PROCEEDINGS

TONGJI-YALE
NETWORKING SYSTEMS GROUP

TONGJI CONTRIBUTIONS

2016 – 2019

Jiading, Shanghai, China

New Haven, Connecticut, USA

Last update: Oct. 12, 2019

Table of Contents

Tongji First Papers

Qiao Xiang (Tongji, Yale), J. Jensen Zhang (Tongji, Yale), X. Tony Wang (Tongji, Yale), Y. Jace Liu (Tongji), Chin Guok (LBNL), Franck Le (IBM), John MacAuley (LBNL), Harvey Newman (California Institute of Technology), Y. Richard Yang (Tongji, Yale)
In Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos.

Official Link: https://dl.acm.org/citation.cfm?id=3234208
Note: SIGCOMM is a flagship conference of computer networking

JMS: Joint Bandwidth Allocation and FlowAssignment for Transfers with Multiple Sources 45
Geng Li (Tongji, Yale), Yichen Qian (Tongji), Lili Liu (Tsinghua), Y. Richard Yang (Tongji, Yale)
In Proceedings of the third International Conference on Data Science in Cyberspace (DSC) 2018.

Official Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8411847

Unicorn: Unified Resource Orchestrationfor Multi-Domain, Geo-Distributed Data Analytics 53
Qiao Xiang (Tongji, Yale), Shenshen Chen (Tongji), Kai Gao (Tsinghua, Yale), Harvey Newman (California Institute of Technology), Ian Taylor (Cardiff, University of Notre Dame), Jingxuan Zhang (Tongji), Y. Richard Yang (Tongji, Yale)
In Proceedings of the Smart Computing Workshop on Distributed Analytics InfraStructure and Algorithms for Multi-Organization Federations (DAIS) 2018.

Official Link: https://dais-ita.org/sites/default/files/IEEE-SWC-DAIS-18.pdf

DDP: Distributed Network Updates in SDN 59
Geng Li (Tongji, Yale), Yichen Qian (Tongji), Chenxingyu Zhao (Peking), Y. Richard Yang (Tongji, Yale), Tong Yang (Peking)
In Proceedings of the third eighth International Conference on Distributed Computing Systems (ICDCS) 2018.

Official Link: https://ieeexplore.ieee.org/document/8416414
Note: ICDCS is a top conference of distributed computing

Unicorn: Unified Resource Orchestration for Multi-Domain, Geo-Distributed Data Analytics ... 65
Qiao Xiang (Tongji, Yale), X. Tony Wang (Tongji, Yale), J. Jensen Zhang (Tongji), Harvey Newman (California Institute of Technology), Y. Richard Yang (Tongji, Yale), Y. Jace Liu (Tongji)
In Proceedings of the forth workshop on Innovating the Network for Data-Intensive Science (INDIS) 2017.

Official Link: https://scinet.supercomputing.org/workshop/sites/default/files/Xiang-Unicorn_0.pdf

Game-Theoretic User Association in Ultra-dense Networks with Device-to-Device Relays 76
Geng Li (Tongji, Yale), Yuping Zhao (Peking), Dou Li (Peking)
Wireless Personal Communications: An International Journal 95 (2017) 2691-2708.

Official Link: https://dl.acm.org/citation.cfm?id=3134961

Auc2Reserve: A Differentially Private Auction for Electric Vehicle Fast Charging Reservation 94
Qiao Xiang (Tongji, Yale), Linghe Kong (McGill, Shanghai Jiaotong), Xue Liu (McGill), Jingdong Xu (Nankai), Wei Wang (Tongji)
In Proceedings of the twenty second International Conference on Embedded and Real-Time Computing Systems and Applications (RTSA) 2016.

Official Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7579930

Tongji Collaboration Papers

Update Algebra: Toward Continuous, Non-Blocking Composition of Network Updates in SDN 104
Geng Li (Yale, Tongji), Y. Richard Yang (Yale), Franck Le (IBM), Yeon-sup Lim (IBM), Junqi Wang (Rutgers)
In Proceedings of the IEEE International Conference on Computer Communications (INFOCOM) 2019.

Official Link: https://ieeexplore.ieee.org/document/8737618
Note: INFOCOM is a flagship conference of computer networking

On Max-min Fair Allocation for Multi-source Transmission 113
Geng Li (Yale, Tongji), Yichen Qian (Tongji), Y. Richard Yang (Yale)
ACM SIGCOMM Computer Communication Review 48 (2018) 2—8.

Official Link: https://ccronline.sigcomm.org/wp-content/uploads/2019/02/sigcomm-ccr-final199.pdf

An Objective-Driven On-Demand Network Abstraction for Adaptive Applications 120
Kai Gao (Tsinghua), Qiao Xiang (Tongji, Yale), Xin Wang (Tongji, Yale), Yang Richard Yang (Yale), Jun Bi (Tsinghua)
IEEE/ACM Transactions on Networking 27 (2019) 805—818

Official Link: https://ieeexplore.ieee.org/document/8674832

Precedence: Enabling Compact Program Layout By Table Dependency Resolution 134
Christopher Leet (Yale), Shenshen Chen (Yale, Tongji), Kai Gao (Sichuan), Y. Richard Yang (Yale, Tongji)
In Proceedings of the ACM Symposium on SDN Research (SOSR) 2019.

Official Link: https://dl.acm.org/citation.cfm?id=3314148.3314348

IEEE Communications Magazine 54.10 (2016) 53-59.

Official Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7588229

Magellan: Generating Multi-Table Datapath from Datapath Oblivious Algorithmic SDN Policies 198
Andreas Voellmy (Yale), Shenshen Chen (Tongji), Xin Wang (Tongji), Y. Richard Yang (Yale)
In Proceedings of the ACM SIGCOMM 2016 Conference on Posters and Demos.

Official Link: https://dl.acm.org/citation.cfm?id=2959064

FAST: A Simple Programming Abstraction for Complex State-Dependent SDN Programming 200
Kai Gao (Tsinghua), Chen Gu (Tongji), Qiao Xiang (Tongji, Yale), Y. Richard Yang (Tongji, Yale), Jun Bi (Tsinghua)
In Proceedings of the ACM SIGCOMM 2016 Conference on Posters and Demos.

Official Link: https://dl.acm.org/citation.cfm?id=2960424

ORSAP: Abstracting routing state on demand 202
Kai Gao (Tsinghua), Chen Gu (Tongji), Qiao Xiang (Tongji, Yale), Xin Wang (Tongji), Y. Richard Yang (Tongji, Yale), Jun Bi (Tsinghua)
In Proceedings of the twenty forth International Conference on Network Protocols (ICNP) 2016 on Poster.

Official Link: https://ieeexplore.ieee.org/document/7784454

Internet Standards

ALTO Extension: Path Vector 204
Kai Gao (Sichuan), Young Lee (Huawei), Sabine Randriamasy (Nokia Bell Labs), Y. Richard Yang (Yale), Jingxuan Zhang (Tongji)
IETF Internet Proposed Standard 2019.

Official Link: https://tools.ietf.org/html/draft-ietf-alto-path-vector-08

Unified Properties for the ALTO Protocol 239
Wendy Roome (Nokia Bell Labs), Sabine Randriamasy (Nokia Bell Labs), Y. Richard Yang (Yale), Jingxuan Zhang (Tongji), Kai Gao (Sichuan)
IETF Internet Proposed Standard 2019.

Official Link: https://tools.ietf.org/html/draft-ietf-alto-unified-props-new-09

Content Delivery Network Interconnection (CDNI) Request Routing: CDNI Footprint and Capabilities Advertisement using ALTO 282
Jan Seedorf (HFT Stuttgart), Y. Richard Yang (Tongji, Yale), Kevin J. Ma (Ericsson), Jon Peterson (NeuStar), Xiao Lin (Tongji), Jingxuan Zhang (Tongji)
IETF Internet Proposed Standard 2019.

Official Link: https://tools.ietf.org/html/draft-ietf-alto-cdni-request-routing-alto-07

Optimizing in the Dark: Learning an Optimal Solution through a Simple Request Interface

Qiao Xiang,[1,2*] **Haitao Yu,**[1] **James Aspnes,**[2] **Franck Le,**[3] **Linghe Kong,**[4] **Y. Richard Yang**[1,2]
[1]Tongji University, [2]Yale University, [3]IBM T.J. Watson Research Center, [4]Shanghai Jiao Tong University
qiao.xiang@cs.yale.edu, haitao.yu@tongji.edu.cn, james.aspnes@yale.edu,
fle@us.ibm.com, linghe.kong@sjtu.edu.cn, yry@cs.yale.edu

Abstract

Network resource reservation systems are being developed and deployed, driven by the demand and substantial benefits of providing performance predictability for modern distributed applications. However, existing systems suffer limitations: They either are inefficient in finding the optimal resource reservation, or cause private information (*e.g.*, from the network infrastructure) to be exposed (*e.g.*, to the user). In this paper, we design BoxOpt, a novel system that leverages efficient oracle construction techniques in optimization and learning theory to automatically, and swiftly learn the optimal resource reservations without exchanging any private information between the network and the user. We implement a prototype of BoxOpt and demonstrate its efficiency and efficacy via extensive experiments using real network topology and trace. Results show that (1) BoxOpt has a 100% correctness ratio, and (2) for 95% of requests, BoxOpt learns the optimal resource reservation within 13 seconds.

1 Introduction

When facing a genie that only tells you whether it can grant a wish or not, how can you find the best wish it can grant?

Although the question may sound like one from fairy tales, people deal with such question in many real world scenarios. For example, modern distributed applications (*e.g.*, (Zaharia et al. 2012; White 2012)) construct complex data flows between end hosts, *e.g.*, in data center networks. The key to supporting these applications is the ability to provide guaranteed network resources (*i.e.*, bandwidth) for performance predictability (Mogul and Popa 2012). As such, many network resource reservation systems are developed and deployed (Campanella et al. 2006; Guok and Robertson 2006; Johnston, Guok, and Chaniotakis 2011; Riddle 2005; Zheng et al. 2005; Sobieski, Lehman, and Jabbari 2004). However, because of the underlying networks' concern of revealing sensitive information, existing reservation systems do not provide applications with an interface to access information of the underlying network infrastructure (*e.g.*, topology, links' available bandwidth). Instead, networks only offer a *simple reservation interface* for applications to submit requests for reserving a specific amount of bandwidths for a set of flows: $\mathrm{request}(\mathrm{flow\_set}, \mathrm{bw\_values})$), and returns either success or failure. A major concern of this design is its inefficiency for the applications/users to find the optimal amount of network resources to reserve. To further illustrate the issues, consider the example in Figure 1, where a user (*e.g.*, application) wants to determine and reserve the maximum achievable bandwidth for two flows from S_1 to D_1, and S_2 to D_2, respectively. Using existing solutions that offer only a *simple reservation interface*, finding the constraint that both flows can collectively get only 100 Mbps of bandwidth is already an instance of the NP-hard membership-query based constraint acquisition problem (Bessiere et al. 2017), letting alone finding the optimal reservation for both flows (*e.g.*, 50 Mbps for each flow).

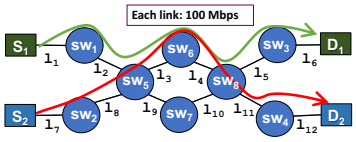


Figure 1: An example network topology: the routes of two flows share bottleneck links, *i.e.*, l_3 and l_4, hence they can only collectively get a 100 Mbps bandwidth.

To address this problem, researchers have proposed several solutions, but all of them suffer limitations, and violate privacy requirements. For example, to determine optimal bandwidth reservations, recent proposals depart from the simple reservation interface, and require either networks to reveal sensitive information to users (Soulé et al. 2014; Subramanian, D'Antoni, and Akella ; Heorhiadi, Reiter, and Sekar 2016; Lee et al.), or vice versa (Gao et al. 2016; Gao et al. 2017; Xiang et al. 2018). These solutions are therefore limited to settings where the level of trust between the applications and the underlying network is high. These solutions cannot be deployed in general settings as malicious parties may use the exposed network information to identify vulnerable links and launch attacks (*e.g.*, DDoS).

In this paper, we explore the feasibility and benefits of learning the optimal network resource reservation for the user without exposing the private information of the network (*i.e.*, bandwidth capacity region) and the user (*e.g.*, resource orchestration policy) to each other. In particular, we tackle the following question: *How can a user learn the optimal*

*The corresponding authors are Q.Xiang and Y. R. Yang.

network resource reservation using only the simple reservation interface? This task is non-trivial due to the extremely limited feedback (*i.e.*, success/failure) provided by simple reservation interface.

Our solution to this problem is BoxOpt, a novel learning system that automatically, and efficiently learns the optimal resource reservations for the user through the simple reservation interface, without exchanging any private information between the network and the user (*e.g.*, bandwidth feasible region of the network and the resource orchestration policy of the user). Specifically, BoxOpt allows users to include their resource reservation objectives as concave utility functions of the requested resources (*e.g.*, bandwidths) in the reservation requests. Upon receiving a reservation request, BoxOpt models the simple reservation interface of network resource reservation systems as a membership oracle over a polytope. It then expands oracle construction techniques (Lovasz, Grotschel, and Schrijver 1993; Lee, Sidford, and Vempala 2017) from optimization and learning theory to construct a separation oracle through invoking the membership oracle in near $O(n)$ iterations (n being the number of flows), which when called upon will accurately infer a search space in which the optimal reservation vector lies. With such a separation oracle, BoxOpt then constructs an optimization oracle based on ellipsoid method, which can learn the optimal reservation vector through $O(n^2)$ calls on the separation oracle.[1] In this way, BoxOpt not only can learn the optimal resource reservation efficiently, but also is privacy-preserving in that no private information is exchanged between the user and the network.

The **main contributions** of this paper are as follows:

• We study the important problem of learning the optimal network resource reservation through the simple reservation interface of network resource reservation systems. In particular, we design BoxOpt, a novel, fast, automatic, privacy-preserving learning system. To the best of our knowledge, BoxOpt is the first working system that solves this problem, and can be extended to other optimization problems.

• We model the simple reservation interface as a membership oracle over a polytope, and expand oracle construction techniques from optimization and learning theory to develop an efficient optimization oracle in BoxOpt, which learns the optimal resource reservation in near $O(n^3)$ of calls on the membership oracle.

• We implement a prototype of BoxOpt and demonstrate both its efficiency and efficacy through extensive experiments using real topologies and traces. Results show that (1) BoxOpt has a 100% correctness ratio, and (2) for 95% cases, it can learn the optimal reservation within 13 seconds.

The remaining of this paper is organized as follows. We present an overview of BoxOpt in Section 2. We give details on how BoxOpt efficiently learns the optimal network resource reservation only using the simple reservation interface in Section 3. We present the evaluation results of BoxOpt in Section 4. We discuss related work in Section 5 and conclude the paper in Section 6.

[1] We choose the ellipsoid method because it is a classic cutting plane method. However, the design of BoxOpt is modular and other cutting plane methods, *e.g.*, analytic center method (Atkinson and Vaidya 1995) and random walk (Bertsimas and Vempala), can also be used to construct an optimization oracle from a separation oracle.

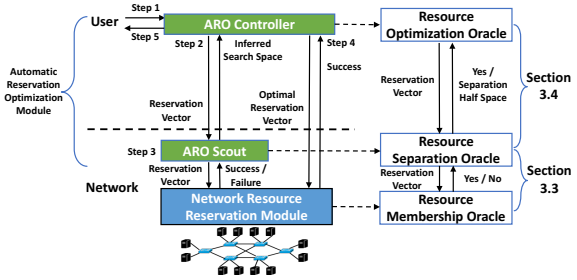


Figure 2: The architecture and workflow of BoxOpt.

2 Overview of BoxOpt

In this section, we first present the architecture and the workflow of BoxOpt. We then give a formal, mathematical formulation of the key technical challenge in BoxOpt: how to find the optimal network resource reservation through the simple reservation interface.

2.1 Architecture

BoxOpt is composed of two components: an automatic reservation optimization (ARO) module for the user, and a network resource reservation (NRR) module for the network (Figure 2). The two components interact with each other through the simple reservation interface commonly used in traditional network resource reservation systems.

Automatic reservation optimization module: The ARO module is a private component belonging to the user, and is composed of two sub-components: an ARO controller, and an ARO scout.

The ARO controller is the main interface for the user to submit the resource reservation requests. A request consists of a set of n flows, $F = \{f_1, f_2, \ldots, f_n\}$, to reserve the resources for, and a concave utility function $util(\mathbf{x})$ to maximize, with $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and each x_i representing the available bandwidth that can be reserved for flow $f_i \in F$. Example utility functions include total throughput and priority-based total throughput. Given a user resource reservation request, the objective of the ARO controller is to infer the optimal resource reservation to maximize $util(\mathbf{x})$. The ARO controller achieves it with the assistance of the ARO scout: Specifically, the ARO controller iteratively selects a vector $\check{\mathbf{x}}$ of bandwidth values for F (called *reservation vector*) and sends it to the ARO scout. For each reservation vector, the ARO scout returns a search space where the optimal reservation vector lies in. With the inferred search spaces returned by the ARO scout, the ARO controller gradually converges to the optimal reservation vector that maximizes $util(\mathbf{x})$.

The ARO scout is the main user entity interacting with the NNR. For each $\check{\mathbf{x}}$ from the ARO controller, the ARO scout infers a search space where the optimal reservation

vector lies in, and returns the inferred search space back to the ARO controller. To infer the search space where the the optimal reservation vector lies in, the ARO scout sends a sequence of reservation vectors to the NRR through the simple reservation interface. As further described in Section 3 and Section 4, for each reservation vector submitted from the ARO controller, the ARO scout might submit tens or hundreds of reservation vectors to the NNR to get an accurately-inferred search space, potentially, leading to a high overhead. As such, to reduce the total latency to find the optimal reservation vector, the ARO scout is placed with the network instead of the user. This design decision reduces the user-network communication latency by 20x as demonstrated in the evaluation section. More importantly, this design does not expose the private information of the user (*i.e.*, $util(\mathbf{x})$) to the network, as the ARO controller does not send such information to the scout.

Network Resource Reservation Module: The NRR module is a private component belonging to the network. Its primary role is to verify whether the reservation vectors submitted by the ARO scout can be satisfied. Upon receiving a reservation vector from the ARO scout, the NRR extracts the relevant constraints from the network. The constraints include both physical network constraints (*e.g.*, if two flows share a same link, their allocated bandwidths cannot exceed the link's available bandwidth), and network policies (*e.g.*, rate limiting, etc.) The constraints are captured as an abstraction of linear inequalities (Gao et al. 2016; Xiang et al. 2018). For example, to capture the physical network constraints, the NNR first retrieves the routes (i.e., sequence of traversed links) for each flow. Then, for each link l in the network, the NNR generates the following linear inequality to ensure that the allocated bandwidths to the flows do not exceed the link's available bandwidth:

$$\sum x_i \leq w_l, \forall f_i \text{ that uses } l \text{ in this route},$$

where w_l is the available bandwidth on link l. Considering the example in Figure 1, the NRR module generates the following linear inequalities:

$$\begin{array}{ll} x_1 \leq 100 & \forall l_u \in \{l_1, l_2, l_5, l_6\}, \\ x_2 \leq 100 & \forall l_u \in \{l_7, l_8, l_{11}, l_{12}\}, \\ x_1 + x_2 \leq 100 & \forall l_u \in \{l_3, l_4\}, \\ x_1, x_2, x_3 \geq 0. & \end{array} \quad (1)$$

Then, the NRR generates additional linear inequalities to represent the network's internal traffic engineering policies, such as load-balancing and bandwidth limiting. For example, suppose the network wants to limit the total bandwidth of flows f_1 and f_2 to be no more than 80 Mbps even if there is no common link in their routes. Then a linear inequality $x_1 + x_2 + x_3 \leq 80$ is generated to represent this policy. Geometrically, the abstraction of linear inequalities represents the *bandwidth feasible region* of the network for providing bandwidths to a set of flows.

Finally, for each generated linear inequality, the NRR checks if it is satisfied by the bandwidth values specified in the reservation vector. If any inequality is violated, it returns a FAILURE signal. Otherwise, it returns SUCCESS.

2.2 Workflow

Having presented the basic components of BoxOpt, we now briefly present its workflow to automatically compute and reserve the optimal network resources for a set of flows as follows (Figure 2):

• Step 1: The user submits a resource reservation request for a set of flows F to the ARO controller. The request also includes a concave utility function $util(\mathbf{x})$ of the bandwidths of F.

• Step 2: In an outer loop, the ARO controller iteratively selects reservation vectors to send to the ARO scout. The selection of the reservation vectors is described in Section 3.4, Algorithm 3. In return, for each reservation vector, the ARO scout determines and replies with an inferred search space.

• Step 3: In an inner loop, upon receiving a reservation vector from the controller, the ARO scout interacts with the NRR, according to Algorithm 1 from Section 3.3, to infer the next search space and send it back to the ARO controller.

The nested iteration of Step 2 and 3 stops when the ARO controller converges to the optimal reservation vector maximizing $util(\mathbf{x})$.

• Step 4: The ARO controller sends the optimal reservation vector to the NRR module to reserve the optimal resources for the user.

• Step 5: The ARO controller confirms with the user that the optimal network resource reservation has been successful.

2.3 Key Challenge

Through the introduction of its architecture and workflow, we show that BoxOpt is *privacy-preserving* by design: neither the user nor the network exposes the private information (*i.e.*, internal optimization objective of the user and the bandwidth capacity region of the network) to the other party. As such, the remaining key challenge for BoxOpt lies in Step 2 and 3: *how can the ARO module interact with the NRR module through the simple reservation interface to compute the optimal network resource reservation?*. To address this challenge, we first give a formal, mathematical formulation.

Specifically, we first model the NRR module as a *resource membership oracle*. Without loss of generality, we use $\mathbf{Ax} \leq \mathbf{b}$ to denote the set of linear inequalities generated by the NRR module, and use $K : \{\mathbf{x} | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ to represent the bandwidth feasible region for a set of flows F. In this way, we give the definition of resource membership oracle:

Definition 1. *[Reservation Membership Oracle (ReMEM)] Given a reservation vector $\check{\mathbf{x}}$, return YES if $\check{\mathbf{x}} \in K$, and return NO otherwise.*

$ReMEM(\check{\mathbf{x}})$ accurately captures the interaction between the ARO scout and the NRR module. Next, we formally define the problem of network resource reservation optimization via simple reservation interface.

Problem 1 (Optimization via Membership Oracle)**.** *Find the optimal solution to the following optimization problem*

$$\textit{maximize} \quad util(\mathbf{x}), \quad (2)$$

subject to,

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{3}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{4}$$

without the knowledge of $\mathbf{A}$ *and* $\mathbf{b}$*, but only using ReMEM defined in Definition 1.*

Maximizing $util(\mathbf{x})$ subject to $K : \{\mathbf{x}|\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ is a classic convex optimization problem. There has been a rich body of literature on how to efficiently solve such problems (Boyd and Vandenberghe 2004). However, most of the existing algorithms require the knowledge of the feasible region (in our case $\mathbf{A}\mathbf{x} \leq \mathbf{b}$). One may think of a strawman to learn K through the ReMEM oracle, and apply the standard optimization techniques to find the optimal $\mathbf{x}$. However, finding the feasible region through a membership oracle is NP-hard (Bessiere et al. 2017), making this strawman impractical. In contrast, as we will present next, BoxOpt resorts to efficient oracle transformation techniques in optimization and learning theory (Lee, Sidford, and Vempala 2017; Lovasz, Grotschel, and Schrijver 1993) to solve this problem (*i.e.*, efficiently learn the optimal resource reservation via membership oracle).

3 Optimizing Network Resource Reservation via Simple Reservation Interface

Having formally defined the key challenge for BoxOpt as a problem of optimization via membership oracle, this section discusses how we solve this problem. For presentation clarity, this section starts by reviewing some concepts in optimization theory. Then, it presents the basic idea of our solution, followed by its details.

3.1 Notations

Unless explicitly noted, we use v to denote a scalar and $\mathbf{v}$ to denote a vector of n dimensions, where n is the number of flows the user wants to reserve bandwidth for (see Section 2.1). We use $\|\mathbf{v}\|_2 = \sqrt{\sum v_i^2}$ to denote the Euclidean norm of $\mathbf{v}$, and use $\|\mathbf{v}\|_\infty = \max|v_i|$ to denote the maximum norm of $\mathbf{v}$. We use $B_2^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_2 \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose Euclidean distance to $\mathbf{m}$ is at most η, and use $B_\infty^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_\infty \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose maximum norm distance to $\mathbf{m}$ is at most η.

3.2 Basic Idea

Our approach to solve Problem 1 utilizes the equivalence and polar relationships between different oracles in optimization theory (Lovasz, Grotschel, and Schrijver 1993). In particular, we focus on the relationships between ReMEM with the following two oracles:

Definition 2. *[Resource Separation Oracle (ReSEP)] Given a reservation vector* $\check{\mathbf{x}}$*, return YES if* $\check{\mathbf{x}} \in K$*, and otherwise return a half space* $\{\mathbf{y}|\mathbf{p}^T(\mathbf{y} - \check{\mathbf{x}}) \leq \delta\}$ *that contains* K *but not* $\check{\mathbf{x}}$*.*

Definition 3. *[Resource Optimization Oracle (ReOPT)] Given a reservation request for a set of flows* F *and the utility function* $util(\mathbf{x})$*, find* $\mathbf{x}^* \in K$ *that maximizes* $util(\mathbf{x})$*.*

Algorithm 1: Resource Separation Oracle $ReSEP(\check{\mathbf{x}})$

```
Select ε ∈ (0, r], ρ ∈ (0, 1);
κ ← R/r;
if ReMEM returns YES for x̌ then
    return x̌ ∈ K ;
else if x̌ ∉ B_2(0, R) then
    return the half space {y|x̌^T(y − x̌) ≤ 0};
else
    r_1 ← rR^{−1/3} ε^{1/3};
    g̃ ← Subgradient(0, r_1, 4ε, 3κ);
    return the half space
      {y|g̃^T(y − x̌) ≤ (40n + 1)ρ^{−1} R^{2/3} κε^{1/3}};
```

Given that there exist efficient algorithms (*e.g.*, ellipsoid method) that can construct an optimization oracle through invoking a separation oracle with a polynomial number of iterations, if we can construct a separation oracle through a polynomial number of calls to a membership oracle, we will be able to solve Problem 1.

One may think classic half space learning techniques can achieve such a construction of separation oracle via membership oracle. However, the problems are different. In half space learning, the goal is to compute a hyperplane to separate a set of given samples (in our case, the reservation vectors) from two predefined classes. In contrast, the goal of ReMEM to ReSEP construction is to compute a hyperplane separating K and a reservation vector not belonging to K by strategically choosing a minimal number of reservation vectors to send to ReMEM.

Specifically, we develop our solution to Problem 1 in two phases. First, we leverage recent progress on geometric algorithms (Lee, Sidford, and Vempala 2017) to develop an efficient algorithm that constructs ReSEP through invoking ReMEM for a polynomial number of times. Specifically, our algorithm expands the weak membership/separation oracle construction in (Lee, Sidford, and Vempala 2017) to strong membership/separation oracle construction. Second, we develop an ellipsoid-method-based algorithm that constructs ReOPT through local feasibility checks and invoking ReSEP for a polynomial number of times. Mapping these two phases to Step 2 and 3 in the workflow of BoxOpt, we see that the ARO scout is essentially the separation oracle ReSEP, and the ARO controller is the optimization oracle ReOPT (Figure 2). Next, we give the details of each phase.

3.3 From Resource Membership Oracle to Resource Separation Oracle

To construct ReSEP from ReMEM, we first define two auxiliary functions on vector $\mathbf{d} \in K$ given a reservation vector $\check{\mathbf{x}}$:

$$\alpha_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow \max_{\mathbf{d}+\alpha\check{\mathbf{x}} \in K} \alpha, \tag{5}$$

$$h_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow -\alpha_{\check{\mathbf{x}}}(\mathbf{d})\|\check{\mathbf{x}}\|_2 \tag{6}$$

We see that given a vector $\mathbf{d} \in K$, $\mathbf{d} + \alpha_{\check{\mathbf{x}}}(\mathbf{d})$ is the last vector on the line from $\mathbf{d}$ to $\mathbf{d} + \check{\mathbf{x}}$ that is in K, and that $-h_{\check{\mathbf{x}}}(\mathbf{d})$

Algorithm 2: Computing the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ $Subgradient(\mathbf{d}, r_1, \tau, L)$

1 $r_2 \leftarrow \sqrt{\frac{\tau r_1}{\sqrt{n}L}}$;
2 Randomly select $\mathbf{y}$ from $B_\infty(\mathbf{d}, r_1)$ following a uniform distribution;
3 Randomly select $\mathbf{z}$ from $B_\infty(\mathbf{y}, r_2)$ following a uniform distribution;
4 **for** $i \leftarrow 1, \ldots, n$ **do**
5 Define line segment $B_\infty(\mathbf{y}, r_2) \cap \mathbf{z} + s\mathbf{e_i}$, where $s \in R$ and $\mathbf{e_i}$ is a vector whose elements are all zeros except the ith one;
6 Denote the endpoints of this line segment as $\mathbf{s_i}$ and $\mathbf{t_i}$, respectively;
7 Evaluate $h_{\check{\mathbf{x}}}(\mathbf{t_i})$ and $h_{\check{\mathbf{x}}}(\mathbf{s_i})$ using binary search and ReMEM;
8 $\tilde{g}_i = \frac{h_{\check{\mathbf{x}}}(\mathbf{t_i}) - h_{\check{\mathbf{x}}}(\mathbf{s_i})}{2r_2}$;
9 **return** $\tilde{\mathbf{g}}$;

is the Euclidean distance from d to this point. Without loss of generality, we assume that $B_2^+(\mathbf{0}, r) \subset K \subset B_2^+(\mathbf{0}, R)$ for some positive numbers r, R and such an assumption can be trivially satisfied in practice. Extending the proof technique in (Lee, Sidford, and Vempala 2017) for an n-dimensional ball to only a partial ball on the first orthant, we get

Lemma 1. *Given $\check{\mathbf{x}}$, $h_{\check{\mathbf{x}}}(\mathbf{d})$ is convex on K, and is $\frac{R+\theta}{R-\theta}$ Lipschitz in $B_2^+(\mathbf{0}, \theta)$ for $0 < \theta < r$.*

With these auxiliary functions and a theorem that for any Lipschitz function, it is linear on a small ball (Bubeck and Eldan 2016), we can construct ReSEP by computing the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ at $\mathbf{d} = \mathbf{0}$, which can be computed by binary search and invoking ReMEM. The constructed separation oracle is presented in Algorithm 1, and the computation of the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ is presented in Algorithm 2.

A key insight in Algorithm 1 is that it expands the applicability of similar construction process from weak membership/separation oracles to strong membership/separation oracles (*i.e.*, ReMEM and ReSEP). In particular, we have

Lemma 2. *There is a random variable ϕ with expectation $E(\phi) \leq 2n\sqrt{\frac{\tau L}{r_1}}$ such that $\forall \mathbf{q} \in K$,*

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{d}) + \tilde{\mathbf{g}}^T(\mathbf{q} - \mathbf{d}) - \phi\|\mathbf{q} - \mathbf{d}\|_\infty - 4nr_1L.$$

With this lemma, we show the correctness of Algorithm 1 in the following theorem.

Theorem 1. *If $\check{\mathbf{x}} \notin K$, Algorithm 1 yields a half space containing K but not $\check{\mathbf{x}}$ with probability $1 - \rho$.*

Proof. When $\check{\mathbf{x}} \notin B_2^+(\mathbf{0}, R)$, it is easy to see that the returned half space $\{\mathbf{y}|\check{\mathbf{x}}(\mathbf{y}) - \check{\mathbf{x}} \leq \mathbf{0}\}$ (Line 6 of Algorithm 1) contains K but not $\check{\mathbf{x}}$. When $\check{\mathbf{x}} \notin K$ but $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, from Lemma 1, we know that $h_{\check{\mathbf{x}}}(\mathbf{d})$ has a Lipschitz constant of 3κ on $B_2^+(\mathbf{0}, \frac{r}{2})$. By selecting $\epsilon \in (0, r]$ and setting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ (Line 1 and 8 of Algorithm 1, respectively), we can get $B_\infty^+(\mathbf{0}, 2r_1) \subset B_2^+(\mathbf{0}, \frac{r}{2})$. As such, we can apply Lemma 2 and get that $\forall \mathbf{q} \in K$

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \mathbf{q} - \phi\|\mathbf{q}\|_\infty - 12nr_1\kappa. \quad (7)$$

Next, because $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, we have $-\frac{1}{\kappa}\check{\mathbf{x}} \in K$ and $h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) = h_{\check{\mathbf{x}}}(\mathbf{0}) - \frac{\|\check{\mathbf{x}}\|_2}{\kappa}$. Then we use Lemma 2 to get

$$h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot -\frac{1}{\kappa}\check{\mathbf{x}} - \frac{\phi}{\kappa}\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa, \quad (8)$$

As such, we then get

$$\tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq \|\check{\mathbf{x}}\|_2 - \phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2. \quad (9)$$

Next, because $\epsilon \in (0, r]$, $\check{\mathbf{x}} \notin K$ and $B_2^+(\mathbf{0}, r) \subset K$, we have $(1 - \frac{\epsilon}{r})K \subset K$. By definition of $h_{\check{\mathbf{x}}}(\mathbf{d})$, we have

$$h_{\check{\mathbf{x}}}(\mathbf{0}) \geq -(1 - \frac{\epsilon}{r})\|\check{\mathbf{x}}\|_2 \geq -\|\check{\mathbf{x}}\|_2 + \epsilon\kappa. \quad (10)$$

Adding Equations (9) and (10) and then subtracting $2\epsilon\kappa$ on the right hand side, we get

$$h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq -\phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2 - \epsilon\kappa. \quad (11)$$

Next, we add Equation (11) to Equation (7) and get that $\forall \mathbf{q} \in K$,

$$\begin{aligned} h_{\check{\mathbf{x}}}(\mathbf{q}) &\geq \tilde{\mathbf{g}}^T(\mathbf{q} - \check{\mathbf{x}}) - \phi\|\mathbf{q}\|_\infty - \phi\|\check{\mathbf{x}}\|_\infty \\ &\quad -12nr_1\kappa - 12nr_1\kappa^2 - \epsilon\kappa \\ &\geq \tilde{\mathbf{g}}^T(\mathbf{q} - \check{\mathbf{x}}) - 2\phi R - 24nr_1\kappa^2 - \epsilon\kappa. \end{aligned} \quad (12)$$

$\forall \mathbf{q} \in K$, we have $h_{\check{\mathbf{x}}}(\mathbf{q}) \leq 0$, and then we can have $\tilde{\phi} \geq \tilde{\mathbf{g}}(\mathbf{q} - \check{\mathbf{x}})$, where $\tilde{\phi}$ is a random scalar independent of $\mathbf{q}$ that satisfies

$$E(\tilde{\phi}) \leq 4\sqrt{\frac{12\epsilon\kappa}{r_1}}nR + 24nr_1\kappa^2 - \epsilon\kappa. \quad (13)$$

Putting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ into Equation (13), we get

$$E(\tilde{\phi}) \leq 40n\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa + \epsilon\kappa. \quad (14)$$

Further leveraging $\epsilon \leq r \leq R$, we get

$$E(\tilde{\phi}) \leq (40n + 1)\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa. \quad (15)$$

Then we can finish the proof using Equation (15) and Markov inequality. □

In addition, observing Algorithm 1 and Algorithm 2, we see that the bottleneck to construct ReSEP is to compute $h_{\check{\mathbf{x}}}$ using binary search and ReMEM (Line 4-8 in Algorithm 2). As such, we give the following theorem on the complexity of Algorithm 1.

Theorem 2. *Algorithm 1 constructs the reservation separation oracle (ReSEP) through an $O(n \log R)$ calls on the reservation membership oracle (ReMEM).*

3.4 From Resource Separation Oracle to Resource Optimization Oracle

Having constructed ReSEP from ReMEM, we next develop an ellipsoid-method algorithm to construct ReOPT from ReSEP, which is summarized in Algorithm 3.

This algorithm adopts a binary search strategy to find the largest $util(\mathbf{x})$ that is feasible on K. In each main iteration (Line 3-24), it constructs a feasible problem

$$\begin{aligned} util(\mathbf{x}) &\geq u_{next}, \\ K : \mathbf{A}\mathbf{x} &\leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \end{aligned} \quad (16)$$

and uses ellipsoid method to test if this problem is feasible. One difference from the classic ellipsoid method is: when evaluating if the center $\mathbf{p}$ of an ellipsoid E_i is a feasible solution, we first evaluate if $\mathbf{p}$ is feasible for $util(\mathbf{x}) \geq u_{next}$, and only invoke ReSEP if $util(\mathbf{p}) \geq u_{next}$. This is because $util(\mathbf{x})$ is kept at the ARO controller, where ReOPT runs, but not shared to ARO scout, where ReSEP resides. This would not affect the correctness of the ellipsoid method for verifying the feasibility of Equation (16) because a half space containing $util(\mathbf{x}) \geq u_{next}$ will also contain the feasible region defined in Equation (16). In the meantime, the privacy of user is also preserved.

Algorithm 3: Reservation Optimization Oracle $ReOPT(util(\mathbf{x}))$.

```
Compute the maximum and minimum of util(x) subject to
  x_i ∈ [0, R] where i = 1, ..., n and denote the value as
  u_max and u_min;
u_l ← u_min, u_r ← u_max;
while u_l < u_r do
    u_next ← (u_l + u_r)/2;
    Build an ellipsoid E_0 bounding B_2^+(0, R);
    V_l ← Vol(B_2^+(0, r)), i ← 0;
    feasible ← false;
    while Vol(E_i) ≥ V_l do
        p ← center of E_i;
        if util(p) < u_next then
            feasible ← false;
            H ← {y|(∇util(p))^T (y − p) >= 0};
        else if ReSEP(p) returns a half space H then
            feasible ← false;
        else
            feasible ← true;
            x* ← p;
            break;
        E_{i+1} ← the minimum-volume ellipsoid containing
          E_i ∩ H;
        i ← i + 1;
    if feasible == false then
        u_r ← u_next;
    else
        u_l ← u_next;
return x*;
```

We present the following theorem on the optimality and efficiency of Algorithm 3.

Theorem 3. *Algorithm 3 finds $\mathbf{x}^*$ that maximizes $util(\mathbf{x})$ subject to K through an $O(n^2)$ calls on the reservation separation oracle (ReMEM).*

Proof. The complexity result in this theorem follows the classic ellipsoid method. For the optimality claim, we observe that even if $util(\mathbf{x})$ is concave, K is still a polytope. As such, $\mathbf{x}^*$ will be a vertex of this polytope. In this way, the optimality of Algorithm 3 can be proved in the same way as the ellipsoid method optimally solves linear programming. □

Putting Theorems 2 and 3 together, we get the following theorem on the optimality and efficiency of BoxOpt.

Theorem 4. *BoxOpt finds $\mathbf{x}^*$ that maximizes $util(\mathbf{x})$ subject to K through an $O(n^3 \log R)$ calls on the reservation membership oracle (ReMEM).*

4 Performance Evaluation

We implement a prototype of BoxOpt and evaluate its performance on an operational federation network supporting large-scale distributed science collaborations, and using real traffic traces from recent science experiments. We first describe our setup, followed by the detailed results.

4.1 Methodology

We evaluate the performance of BoxOpt on the topology from LHC Open Network Environment (LHCONE), a global science network consisting of 62 institutes (Martelli and Stancu 2015). We randomly select a topology for each institute from the Topology Zoo (Knight et al. 2011), and then assemble the connections and topologies from previous steps into a unified large network. We replay the actual trace from the CMS experiment (cms-dashb), a main source of traffic in LHCONE. We focus on a 48-hour trace starting from December 14, 2017, consisting of 716 resource reservation requests. The number of flows in each request varies between 1 and 7. Because the CMS experiment is one of the largest ongoing distributed scientific experiments with complex, distributed analytics across tens of geographically distributed locations, we believe the trace is representative of complex data flow of modern distributed applications.

4.2 Results

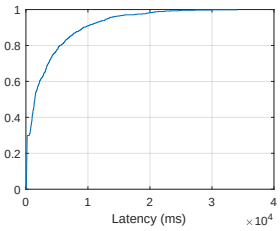
In our experiments, we set R and r in Algorithm 1 to be the maximum and minimum of link bandwidth in the network topology. We run extensive experiments by choosing different values of ϵ and ρ and different utility functions. In what follows, we present the results of one setting: maximizing total throughput when $\rho = 0.001$ and $\epsilon = \frac{1}{5}r$. Results of other settings are highly similar as this setting, hence are omitted due to page limit.

Correctness of BoxOpt: For each reservation request, we compare the optimal resource reservation computed by BoxOpt with the optimal solution to the problem $util(\mathbf{x})$ subject to K computed by a state-of-the-art optimization solver (*e.g.*, CPLEX (CPLEX 2018)). We find that in all 716 requests, BoxOpt outputs the same optimal solution as the solver does, *i.e.*, BoxOpt has a 100% correctness ratio.

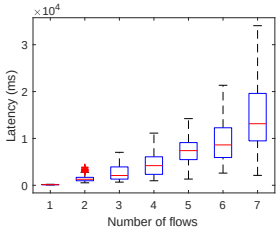
Efficiency of BoxOpt: As illustrated in Figure 2 (Section 2), the main bottleneck of BoxOpt is the communication latency between the optimization oracle at the ARO controller invoking the separation oracle at the ARO scout as the computation latency is ignorable. As such, we use the total communication latency to find the optimal resource reservation for each request to represent the efficiency of BoxOpt. Specifically, we assume the user is located at New York and the network is in Los Angeles. For each invocation of ReSEP at ARO scout, we assign it a round trip time (RTT) randomly chosen from the statistic RTT data collected in (global-ping

). Then the communication latency to find the optimal resource reservation for a given request is the sum of all RTTs incurred by corresponding ReSEP invocations.

Figure 3a plots the CDF of communication latency of all requests in the experiment. We observe that for 95% of the requests, BoxOpt is able to learn the optimal resource reservation within 13 seconds. This demonstrates the efficiency of BoxOpt to swiftly learn the optimal resource reservation via the simple reservation interface. Figure 3b plots the statistics of the communication latency for different sizes of reservation requests. The nonlinear increase of the latency is consistent with conclusion in Theorem 3. However, compared with the lasting time of network resource reservation (*e.g.*, hours and days) and the amount of data being transmitted (*e.g.*, TBs), BoxOpt is highly efficient.



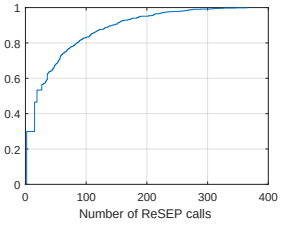
(a) CDF of the communication latency of BoxOpt.



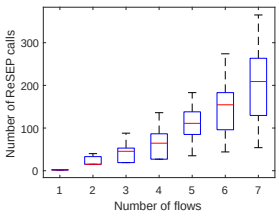
(b) Communication latency of BoxOpt with different sizes of request.

Figure 3: Efficiency of BoxOpt: total communication latency to compute the optimal resource reservation.

Efficiency of ReOPT: We next study the efficiency of the ReOPT oracle to learn the optimal resource reservation. To this end, we count the number of ReSEP invocations (*i.e.*, the bottleneck operation of the ReOPT oracle) for each request. Figure 4a gives the CDF of the number of ReSEP invocations. We observe that for 95% requests, the ReOPT learns the optimal reservation within 200 ReSEP calls. Figure 4b further breaks down the statistics based on the size of requests. We observe that the nonlinear increase of ReSEP invocations is consistent with Theorem 3 and Figure 3b.



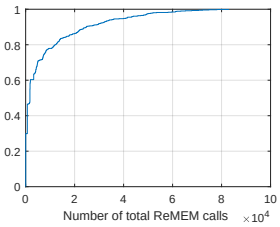
(a) CDF of the number of ReSEP invocations.



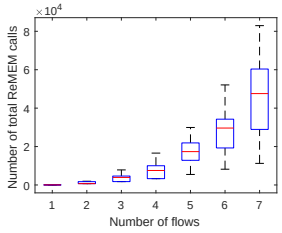
(b) Number of ReSEP invocations with different sizes of request.

Figure 4: Efficiency of ReOPT: number of ReSEP invocations to learn the optimal resource reservation.

Efficiency of ReSEP: In the end, we study the efficiency of the ReSEP oracle to infer the search space for ReOPT. To this end, we count the number of ReMEM invocations (*i.e.*, the bottleneck operation of the ReSEP oracle) for each request. Figure 5a gives the CDF of the number of ReMEM invocations. We observe that for 95% requests, the total ReMEM invocations required is within 30000. This large number demonstrates the necessity and benefits of putting ReSEP (*i.e.*, the ARO scout) with the network. Integrating this observation from Figure 4a and Figure 3a, we can conclude that this design improves the efficiency of BoxOpt (*i.e.*, the communication latency) by 20 times. Figure 5b further breaks down the statistics based on the size of requests. We observe that the almost linear increase of ReMEM invocations is consistent with Theorem 2.



(a) CDF of the number of ReMEM invocations.



(b) Number of ReMEM invocations with different sizes of request.

Figure 5: Efficiency of ReSEP: number of ReMEM invocations to learn the optimal resource reservation.

5 Related Work

Many network resource reservation systems have been developed and deployed (Campanella et al. 2006; Guok and Robertson 2006; Johnston, Guok, and Chaniotakis 2011; Riddle 2005; Zheng et al. 2005; Sobieski, Lehman, and Jabbari 2004). However, existing systems either are inefficient, or cause private information to be exposed. In contrast, BoxOpt adopts a novel approach to efficiently learn the optimal resource reservation through the limited feedback from the simple interface provided by reservation systems.

One area closely related to our problem is *constraint learning* (De Raedt, Passerini, and Teso 2018; Bessiere et al. 2017; Ruggieri 2012; Bessiere et al. 2004; Ruggieri 2013). We refer readers to (De Raedt, Passerini, and Teso 2018) for a comprehensive survey. Instead of learning all linear inequalities that compose the bandwidth feasible region, in BoxOpt, we show that the optimal solution to an optimization problem can be learnt efficiently and accurately without knowing any constraints. One future direction is to integrate constraint learning into BoxOpt to further accelerate the learning of the optimal resource reservation.

BoxOpt leverages several powerful tools from optimization theory (Lovasz, Grotschel, and Schrijver 1993; Boyd and Vandenberghe 2004; Lee, Sidford, and Vempala 2017). We expand the recent theoretical progress on efficient oracle constructions to a broader scenario. To the best of our knowledge, BoxOpt is the first working system that demonstrates the feasibility and benefits of learning the optimal solution of an optimization problem with only membership oracle. In addition to network resource reservation, it also sheds light for other areas such as multi-domain traffic engineering and collaborative data analytics.

6 Conclusion

We design BoxOpt, a novel, automatic learning system to efficiently learn the optimal resource reservations through the simple reservation interface of network resource reservation systems, without exposing the private information of network or user. We demonstrate its efficiency and efficacy through extensive evaluation using real network topology and trace.

Acknowledgment

The authors thank Christian Bessiere, Haizhou Du, Kai Gao, Chin Guok, Max Del Giudice, Chris Harshaw, Amin Karbasi, Yin Tat Lee, Geng Li, Yang Liu, John MacAuley, Harvey Newman, Lam M. Nguyen, Salvatore Ruggieri, Mudhakar Srivatsa, Xin Wang, Jingxuan Zhang and Rui Zhang for their help during the preparation of this paper. The authors also thank the anonymous reviewers for their valuable comments. This research is supported in part by NSFC grants 61702373, 61672385, 61701347, and 61672349; China Postdoctoral Science Foundation 2017-M611618; NSF awards CCF-1637385, CCF-1650596, OAC-1440745; Google Research Award; and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

References

Atkinson, D. S., and Vaidya, P. M. 1995. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming* 69(1-3):1–43.

Bertsimas, D., and Vempala, S. Solving convex programs by random walks. In *Proceedings of the 34th ACM STOC*.

Bessiere, C.; Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Disjoint, partition and intersection constraints for set and multiset variables. In *International Conference on Principles and Practice of Constraint Programming*, 138–152. Springer.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence* 244:315 – 342.

Boyd, S., and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Bubeck, S., and Eldan, R. 2016. Multi-scale exploration of convex functions and bandit convex optimization. In *Conference on Learning Theory*, 583–589.

Campanella, M.; Krzywania, R.; Reijs, V.; Wilson, D.; Sevasti, A.; Stamos, K.; and Tziouvaras, C. 2006. Bandwidth on demand services for european research and education networks. In *IEEE Bandwidth on Demand 06*, 65–72. IEEE.

CMS Task Monitoring. http://dashb-cms-job.cern.ch/.

2018. Ilog cplex.

De Raedt, L.; Passerini, A.; and Teso, S. 2018. Learning constraints from examples. In *Proceedings in Thirty-Second AAAI Conference on Artificial Intelligence, AAAI, New Orleans, USA*, 02–07.

Gao, K.; Gu, C.; Xiang, Q.; Wang, X.; Yang, Y. R.; and Bi, J. 2016. ORSAP: abstracting routing state on demand. In *24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016*, 1–2.

Gao, K.; Xiang, Q.; Wang, X.; Yang, Y. R.; and Bi, J. 2017. Nova: Towards on-demand equivalent network view abstraction for network optimization. In *IWQoS'17*, 1–10.

Global Ping Statistics - WonderNetwork, 2018. https://wondernetwork.com/pings/.

Guok, C., and Robertson, D. 2006. Esnet on-demand secure circuits and advance reservation system (oscars). *Internet2 Joint* 92.

Heorhiadi, V.; Reiter, M. K.; and Sekar, V. 2016. Simplifying software-defined network optimization using sol. In *NSDI*, 223–237.

Johnston, W.; Guok, C.; and Chaniotakis, E. 2011. Motivation, design, deployment and evolution of a guaranteed bandwidth network service. In *Proceedings of the TERENA Networking Conference*.

Knight, S.; Nguyen, H. X.; Falkner, N.; Bowden, R.; and Roughan, M. 2011. The internet topology zoo. 29(9):1765–1775.

Lee, J.; Turner, Y.; Lee, M.; Popa, L.; Banerjee, S.; Kang, J.-M.; and Sharma, P. Application-driven bandwidth guarantees in datacenters. In *SIGCOMM'14*.

Lee, Y. T.; Sidford, A.; and Vempala, S. S. 2017. Efficient convex optimization with membership oracles. *arXiv preprint arXiv:1706.07357*.

Lovasz, L.; Grotschel, M.; and Schrijver, A. 1993. *Geometric algorithms and combinatorial optimization - 2nd corrected edition*. Springer.

Martelli, E., and Stancu, S. 2015. Lhcopn and lhcone: status and future evolution. In *Journal of Physics: Conference Series*, volume 664, 052025. IOP Publishing.

Mogul, J. C., and Popa, L. 2012. What we talk about when we talk about cloud network performance. *SIGCOMM CCR 12* 42(5):44–48.

Riddle, B. 2005. Bruw: A bandwidth reservation system to support end-user work. In *TERENA Networking Conference, Poznan, Poland*.

Ruggieri, S. 2012. Deciding membership in a class of polyhedra. In *ECAI*, 702–707.

Ruggieri, S. 2013. Learning from polyhedral sets. In *23rd Int. Joint Conference on Artificial Intelligence (IJCAI 2013)*, 1069–1075. AAAI Press.

Sobieski, J.; Lehman, T.; and Jabbari, B. 2004. Dragon: Dynamic resource allocation via gmpls optical networks. In *MCNC Optical Control Planes Workshop, Chicago, Illinois*.

Soulé, R.; Basu, S.; Marandi, P. J.; Pedone, F.; Kleinberg, R.; Sirer, E. G.; and Foster, N. 2014. Merlin: A language for provisioning network resources. In *CoNEXT'14*, 213–226. ACM.

Subramanian, K.; D'Antoni, L.; and Akella, A. Genesis: Synthesizing forwarding tables in multi-tenant networks. *ACM POPL'17*.

White, T. 2012. *Hadoop: The definitive guide*. O'Reilly Media, Inc.

Xiang, Q.; Zhang, J. J.; Wang, X. T.; Liu, Y. J.; Guok, C.; Le, F.; MacAuley, J.; Newman, H.; and Yang, Y. R. 2018. Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 27–29. ACM.

Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M. J.; Shenker, S.; and Stoica, I. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI'12*, 2–2. USENIX Association.

Zheng, X.; Veeraraghavan, M.; Rao, N. S.; Wu, Q.; and Zhu, M. 2005. Cheetah: Circuit-switched high-speed end-to-end transport architecture testbed. *IEEE Communications Magazine* 43(8):S11–S17.

Future Generation Computer Systems 93 (2019) 188–197

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics

Qiao Xiang [a,b,*], X. Tony Wang [a,b], J. Jensen Zhang [a], Harvey Newman [c], Y. Richard Yang [a,b,*], Y. Jace Liu [a]

[a] Tongji University, China
[b] Yale University, United States
[c] California Institute of Technology, United States

HIGHLIGHTS

- First unified resource orchestration framework for multi-domain data analytics.
- Resource state abstraction for accurate, minimal resource information discovery.
- Prototype evaluation and full demonstration at SuperComputing 2017.

ARTICLE INFO

Article history:
Received 31 January 2018
Received in revised form 25 June 2018
Accepted 19 September 2018
Available online 1 November 2018

ABSTRACT

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi-organizational, geographically-distributed, collaborative computing, where different organizations contribute various yet scarce resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment, one of the largest scientific experiments in the world, and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of efficient resource utilization, following the autonomy and privacy of different domains, and that the fundamental challenge for designing such a framework is: *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?* Existing resource management systems are designed for single-domain clusters and cannot address this challenge. In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. In Unicorn, we encode the resource availability for each domain into resource state abstraction, a variant of the network view abstraction extended to accurately represent the availability of multiple resources with minimal information exposure using a set of linear inequalities. We then design a novel, efficient cross-domain query algorithm and a privacy-preserving resource information integration protocol to discover and integrate the accurate, minimal resource availability information for a set of data analytics jobs across different domains. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs. We implement a prototype of Unicorn and present preliminary evaluation results to demonstrate its efficiency and efficacy. We also give a full demonstration of the Unicorn system at SuperComputing 2017.

1. Introduction

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi-organizational, geographically-distributed, collaborative computing, where different organizations (also called domains) contribute various yet disaggregated resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze

* Corresponding author at: Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT, 06511, United States.
E-mail addresses: qiao.xiang@cs.yale.edu (Q. Xiang), 13xinwang@tongji.edu.cn (X. Tony Wang), jingxuan.zhang@tongji.edu.cn (J. Jensen Zhang), newman@hep.caltech.edu (H. Newman), yry@cs.yale.edu (Y. Richard Yang), yang.jace.liu@linux.com (Y. Jace Liu).

https://doi.org/10.1016/j.future.2018.09.048
0167-739X/

extremely large amounts of data. One important example of this paradigm is the Compact Muon Solenoid (CMS) experiment at CERN [1], one of the largest scientific experiments in the world. The CMS data analytics system is composed of over 150 participating organizations, including national laboratories, universities and other research institutes. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment over a 7-day period and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of *efficient resource utilization, following the autonomy and privacy of different domains*.

In particular, our trace analysis shows that (1) over 35% of data analytics jobs are *remote jobs*, *i.e.*, jobs that require different types of resources from different domains for execution; (2) the 90% quantile of the job execution time of remote jobs is approximately 38.9% longer than that of local jobs, *i.e.*, jobs that only require resources from a single domain for execution; and (3) the data transfer traffic is saturating the CMS network, leaving limited networking resources (*i.e.*, less than 15%) for data analytics traffic. These observations show that resources in multi-domain, geo-distributed analytics are highly disaggregated, *i.e.*, unbalanced distributed across domains. Although there is much related work on resource management for clusters and data centers, such as [2–12], they are mostly designed for managing resources in single-domain clusters, and cannot accomplish the aforementioned goal for multi-domain, geo-distributed data analytics. In particular, these systems typically adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for two reasons. First, it *compromises the privacy of different domains* by revealing all the details of resources in each domain. Secondly, *the overhead to keep the resource availability graph up to date* is too expensive due to the heterogeneity and dynamicity of resources from different domains. Some systems such as HTCondor [2] adopts a simpler abstraction that only represents computation and storage resources in multi-domain clusters. This approach, however, leaves the orchestration of networking resources completely to the transmission control protocol (TCP), which has long been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, and hence is inefficient. Through trace analysis and related work study, we identify the fundamental design challenge for designing an orchestration framework for multi-domain, geo-distributed data analytics is *the accurate discovery and representation of resources across different domains with minimal information exposure*.

In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo distributed data analytics. In Unicorn, the resource availability of each domain is abstracted into resource state abstraction, a variant of the network view abstraction [13] extended to accurately represent the availability of multiple resources with minimal information exposure using *a set of linear inequalities*. With this intra-domain abstraction, Unicorn uses a novel, efficient cross-domain resource discovery component to find the accurate resource availability information for a set of data analytics jobs across different domains with minimal information exposure, while allowing each domain to make and practice their own resource management strategies. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs.

The **main** contributions of this paper are as follows:

- we study the novel problem of resource orchestration for multi-domain, geo-distributed data analytics and identify the *cross-domain resource discovery challenge* as the fundamental design challenge for this problem through systematic trace-analysis and vigorously related work investigation;
- we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. Unicorn provides the resource state abstraction for each domain to accurately represent its resource availability with minimal information exposure in the form of a set of linear equalities, a novel, efficient cross-domain resource discovery component to provide the accurate, minimal resource availability information across different domains, and a global resource orchestrator to compute optimal resource allocations for data analytics jobs;
- we implement a prototype of Unicorn and perform preliminary evaluations to demonstrate its efficiency and efficacy. We also present a full demonstration of Unicorn at SuperComputing 2017.

The rest of the paper is organized as follows. We analyze the data analytics trace of the CMS experiment, discuss the inadequacy of existing resource management systems and identify the key design challenge for multi-domain, geo-distributed data analytics systems in Section 2. We introduce the system setting and give an overview of the Unicorn framework in Section 3. We then present the details of two key components of Unicorn, cross-domain resource discovery and representation and global resource orchestration, in Section 4 and 5, respectively. We discuss the implementation details in Section 6 and evaluate the performance of Unicorn in Section 7. We conclude the paper and discuss the next steps of Unicorn in Section 8.

2. Motivation and challenge

Analytics trace from the CMS experiment. We collect the trace of approximately 479 thousand data analytics jobs from the CMS experiment, one of the largest scientific experiments in the world, over a period of 7 days. From this trace, we find that over 35% of jobs consumes resources across different domains, *i.e.*, these jobs use the computation node and the storage node located at different domains which are connected by networking resources across multiple domains. We call these jobs *remote jobs*, compared with *local jobs* which only use resources within one single domain. This result indicates the *resource disaggregation* in the CMS network, *i.e.*, the unbalanced distribution of storage and computation resources. We also plot the cumulative distribution function of job execution time for this set of traces as shown in Fig. 1. We observe that the 90% quantile of job execution time for remote jobs has an extra 38.9% higher latency than local jobs. In addition, we observe that the cross-domain networking resources available for data analytics are very limited because the CMS data transfer traffic is saturating the limited networking resources, *e.g.*, the cross-domain data transfer network traffic of the same 7-day period has a total amount of 8785 terabytes while the cross-domain data analytics traffic is only 1404 terabytes. This observation indicates the scarcity of networking resources available for data analytics in the CMS network. All these results demonstrate that in order to support low-latency, multi-domain, geo-distributed data analytics, it is not only necessary, but crucial to design a multi-domain resource orchestration system.

Related work. There exists a rich literature in the field of resource management of clusters [2–12]. YARN [4] is the core resource management framework of Hadoop. Mesos [3] is a platform designed to share resources among multiple cluster computing frameworks, *e.g.*, MapReduce [14], Spark [15], MPI and etc. Google designs a system called Borg [5] to orchestrate the cluster resources for its

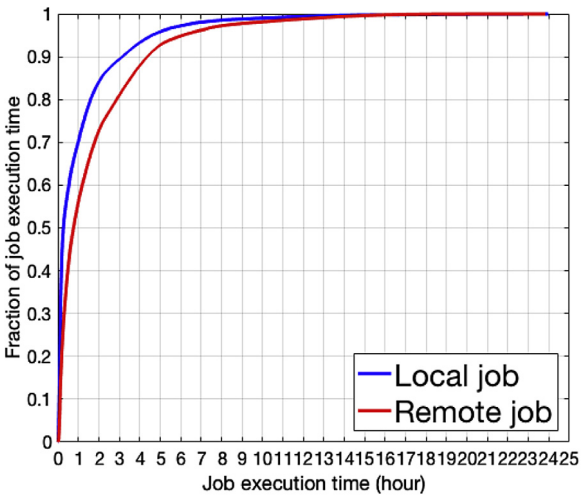


Fig. 1. The CDF of job latency local and remote jobs.

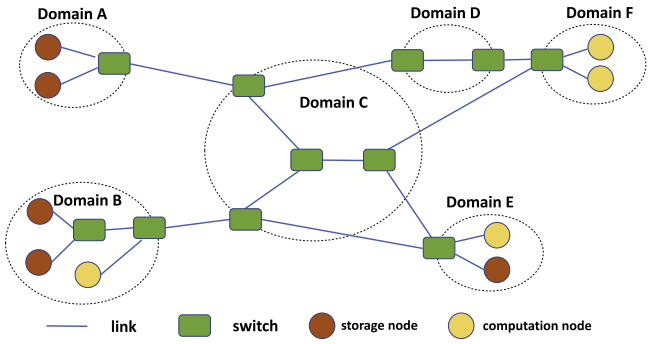


Fig. 2. An example of multi-domain, geo-distributed data analytics system. Domains A, B, E and F are **leaf domains**. Domains C and D are **transmission domains**.

proprietary data analytics frameworks. Microsoft (*i.e.*, Apollo [6]) and Facebook (*i.e.*, Corona [7]) also develop similar systems tailored to their data analytics needs. These systems are all designed for managing resources in single-domain clusters and adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for because (1) it compromises the privacy of different domains by revealing all the details of resources in each domain; and (2) the overhead to keep the resource availability graph up to date is too expensive due to the heterogeneity and dynamicity of resources from different domains.

There are also some efforts towards resource management for multi-domain clusters. HTCondor [2] proposes a ClassAds programming model, which allows different resource owners to advertise their resource supply and the job owners to advertise the resource demand. The CMS [1] experiment currently uses HTCondor and glideinWMS [8] to manage a set of distributively owned computing resources in a globally distributed system. These systems only focus on managing storage and computing resources in clusters, while the recent study shows that computation, storage and networking resources have approximately the same probability to become the bottleneck affecting the performance of data-intensive analytics jobs [16]. By leaving the orchestration of networking resources completely to TCP, which has been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, the abstraction adopted by these systems is also inefficient.

Another line of work called geo-distributed data analytics is also related. Solutions in this field include (1) moving the input dataset to a single data center before the computation [17,18] and (2) placing different amounts of tasks at different sites depending on dataset availability to achieve a better parallelization and hence a lower latency [9–12]. The main focus of these solutions is to optimize the usage of a set of dedicated networking resources. The design of these systems cannot be applied to multi-domain, geo-distributed data analytics where different types of resources owned by different owners need to be orchestrated.

Design challenge. The discussion above shows the urgent need for an efficient resource orchestration framework to support multi-domain, geo-distributed data analytics systems such as CMS. And by investigating the limitations of existing resource management systems, we identify the key design challenge for such a framework is how to achieve resource discovery and representation across different domains with minimal information exposure. To this end, we design the Unicorn framework to manage a large set of distributively-owned, heterogeneous resources for multi-domain, geo-distributed data analytics systems. Unicorn achieves efficient resource utilization while allowing the autonomy and privacy of different domains through a novel resource state abstraction, an efficient cross-domain discovery and representation component and a global resource orchestration component, which will be discussed in the next few sections.

3. Overview

In this section, we introduce the system setting for multi-domain, geo-distributed data analytics and give an overview of the Unicorn framework and its workflow.

System settings. We consider a data analytics system composed of multiple organizations (domains). Each domain contributes a certain amount of computation, storage and networking resources for all the users in the system to store, transfer and analyze large-volume datasets. The storage and computation resources are typically physical servers, virtual machines or containers. The networking resources are typically switches and links. Domains that only contribute networking resources are called *transmission domains* and domains that also contribute computation and storage resources are called *leaf domains*. Fig. 2 gives an example of such a system. In this example, domain A, B, E and F are all leaf domains while domain C and D are transmission domains.

A data analytics task is typically decomposed into a set of jobs J whose precedence relation is specified by a directed acyclic graph (DAG). A task is finished if and only if the last job in the decomposed DAG is finished. Each job j has requirements on storage and computation resources, *e.g.*, number of CPUs, size of memory, input dataset and etc. We use $(stg, comp)$ to denote a pair of candidate storage and computation resources satisfying the requirement of j. The orchestration system is in charge of selecting one $(stg, comp)$ pair for each job j and allocating the selected storage and computation resources and the networking resources connecting them for executing j.

Unicorn architecture. We present the architecture of Unicorn in Fig. 3. On top of all the domains, Unicorn provides a logically centralized controller to orchestrate resources for data analytics jobs. This controller includes a cross-domain resource representation and discovery component and a global resource orchestration component. Residing in each domain are a domain resource manager and a set of job execution agents.

Unicorn provides a novel abstraction called resource state abstraction, a variant of network view abstraction [13]. This abstraction uses a set of linear inequalities to accurately represent the

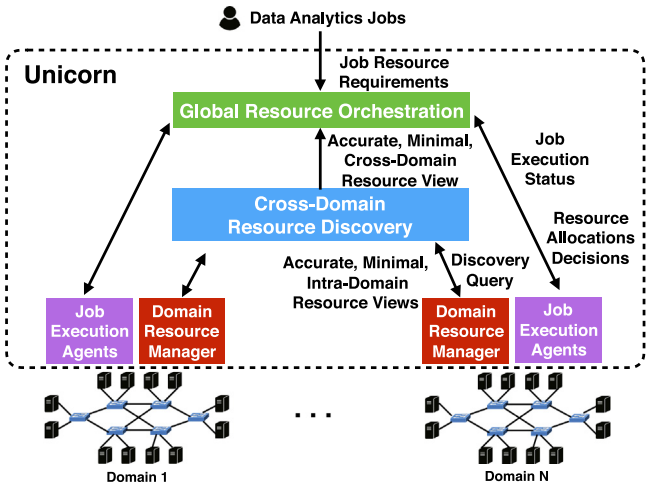


Fig. 3. The architecture of Unicorn.

availability of different resources in each domain with minimal information exposure. When a set of data analytics jobs J are submitted to the Unicorn controller, the cross-domain resource discovery and integration component issues discovery queries, *i.e.*, path queries and resource queries, to the domain resource manager at each domain to retrieve the intra-domain resource view of each domain encoded in the resource state abstraction. It then assembles and compresses the responses into an accurate, minimal cross-domain resource view. This view, together with the resource requirements of j, is then used by the global orchestration component to make global, optimal resource allocation decisions and send to the job execution agents at corresponding domains. The execution agents enforce the received decisions, *e.g.*, starting the corresponding program, rate limiting the data accessing bandwidth and etc., and send the job execution status back to the Unicorn controller as feedback. In the next few sections, we present the design details of key components of Unicorn.

4. Cross-domain resource discovery and representation

In this section, we present our design to address the fundamental challenge of accurately discovering and representing a large set of distributively-owned, heterogeneously resources with minimal information exposure of resource owners. In particular, we introduce a novel abstraction to represent intra-domain resource availability and design an efficient discovery mechanism to discovery resource availability across different domains.

4.1. Intra-domain resource state abstraction

Basic idea. Unicorn framework provides an abstraction called resource state abstraction to accurately represent the availability of multiple resources for a set of data analytics jobs using *a set of linear inequalities*. This is a variant of the network view abstraction [13]. In particular, we consider a set of data analytic jobs J that wants to consume a set of physical resources R (*i.e.*, computation, storage and networking) based on a set of pre-defined policies P. If a resource attribute *attr* is *capacity-bounded*, *i.e.*, a resource r can only provide this attribute with a certain capacity (denoted as $C^{r,attr}$) and each job j consuming r can only get a portion of this attribute (denoted as $c_j^{r,attr}$), the resource availability of R for J on this attribute can be expressed as:

$$\sum_{j \in J(P,r)} c_j^{r,attr} \le C^{r,attr}, \forall r \in R, \tag{1a}$$

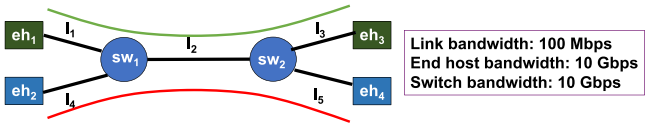


Fig. 4. An example to illustrate the resource state abstraction.

$$c_j^{R,attr} = f(P, attr, c_j^{r,attr}), \forall (j, r \in R), \tag{1b}$$

$$c_j^{r,attr} = g(P, attr, c_j^{r',attr}), \forall (j, r \in R, r' \in R\{r\}). \tag{1c}$$

In this representation. Eq. (1a) indicates that the total amount of *attr* of resource r consumed by all the jobs cannot exceed the supply capacity of r on *attr*, where $J(P, r)$ is the set of jobs that are allowed to consume j based on the policy set P. Eq. (1b) represents the total capacity of *attr* that j can get from the whole set of resources R (denoted as $c_j^{R,attr}$) by a pre-defined linear function of $c_j^{r,attr}$, whose form depends on *attr* and P. Eq. (1c) represents the relation between the amount of *attr* a job j can get from two resources r and r' by a pre-defined linear function, whose form depends on *attr* and P. One of the most common capacity-bounded resource attributes is bandwidth.

If a resource attribute *attr* is *capacity-free*, *i.e.*, each j consuming r who provides this attributes can get the same capacity $C^{r,attr}$ at the same time, the resource availability of R for J on this attribute can be expressed as:

$$c_j^{R,attr} = h(P, R, attr, j), \forall j \in J, \tag{2}$$

where the value of $c_j^{R,attr}$ is computed by a pre-defined function $h(P, R, attr, j)$ whose form depends on *attr* and P. Note that this function does not need to be linear because the value of the right-hand side can be directly computed in this availability representation. Examples of such capacity-free resource attributes include propagation delay, hop-count, and etc.

Example. We use the physical topology in Fig. 4 to illustrate how resource state abstraction works. Suppose two jobs j_1 and j_2 need to read data from storage node eh_1 to computation node eh_3 and from eh_2 to eh_4, respectively. The routing policy for the data flow of each job is also shown in the figure. For simplicity, we only focus on the bandwidth attribute for each resource, *i.e.*, end host, switch and link. Following the definition in Eq. (1), the resource availability of this topology for j_1 and j_2 can be expressed as:

$$\begin{aligned}
c_{j_1}^{l_i} &\le 100 \text{ Mbps}, && i = \{1, 3\},\\
c_{j_2}^{l_i} &\le 100 \text{ Mbps}, && i = \{4, 5\},\\
c_{j_1}^{l_i} + c_{j_2}^{l_i} &\le 100 \text{ Mbps}, && i = \{2\},\\
c_{j_1}^{sw_k} + c_{j_2}^{sw_k} &\le 10 \text{ Gbps}, && k = \{1, 2\},\\
c_{j_1}^{eh_m} &\le 10 \text{ Gbps}, && m = \{1, 3\},\\
c_{j_2}^{eh_m} &\le 10 \text{ Gbps}, && m = \{2, 4\},\\
c_{j_1}^{R} = c_{j_1}^{l_i} &= c_{j_1}^{sw_k} = c_{j_1}^{eh_m}, && i = \{1, 2, 3\}, \forall k, m = \{1, 3\},\\
c_{j_2}^{R} = c_{j_2}^{l_i} &= c_{j_2}^{sw_k} = c_{j_2}^{eh_m}, && i = \{2, 4, 5\}, \forall k, m = \{2, 4\},\\
c_{j_1}^{l_i} &= c_{j_1}^{eh_m} = 0, && i = \{4, 5\}, m = \{2, 4\},\\
c_{j_2}^{l_i} &= c_{j_2}^{eh_m} = 0, && i = \{1, 3\}, m = \{1, 3\},
\end{aligned} \tag{3}$$

Computing minimal, equivalent resource state abstraction. The representation of resource availability defined in Eqs. (1)(2) is accurate and complete, but may result in a large set of linear inequalities with redundant information. In a simple topology in our illustration example, there are already over 20 inequalities. Directly sharing them with a centralized controller or other domains would introduce a large communication overhead and expose unnecessary private information about each domain, *e.g.*, domain topology and policies. We define a metric called *compression ratio*

to measure the exposure of private information of resource state abstraction.

Definition 1 (*Compression Ratio*)**.** Given an original resource state abstraction with M linear inequalities, a compressed, equivalent resource state abstraction with N linear inequalities, which has the same feasible region as the original resource state abstraction, has a compression ratio of $\frac{N}{M}$.

In Unicorn, the domain resource manager adopts a lightweight, optimal algorithm that compresses the original resource state abstraction into an equivalent resource state abstraction with the minimal compression ratio. The basis of this compression algorithm is simple: given an original set of linear inequalities $C : \mathbf{Ax} \leq \mathbf{b}$, we iteratively select one constraint $c \in C : \mathbf{a}^T\mathbf{x} \leq b$ and calculate the optimal solution of problem $y \leftarrow \max \mathbf{a}^T\mathbf{x}$, subject to, $C - \{c\}$. If b is smaller than the resulting y, c is an indispensable constraint in determining the feasible region and will be put into the minimal, equivalent constraint set C'. Otherwise, c is a redundant constraint. We propose the following proposition for this compression algorithm.

Proposition 1. *Given an original resource state abstraction, the proposed compression algorithm computes an equivalent resource state abstraction with the minimal compression ratio.*

This proposition can be proved via contradiction. Applying this algorithm to the example above, we may find that the minimal, equivalent set of linear inequalities has only one inequality: $c^R_{j_1} + c^R_{j_2} \leq$ 100 Mbps. In other words, our compression algorithm achieves the minimal compression ratio of $\frac{1}{31}$.

4.2. Cross-domain resource discovery

The resource state abstraction allows each domain to represent the accurate resource availability for a set of data analytics jobs using a set of linear inequalities with minimal information exposure, but it still requires the knowledge of all available computation, storage and networking resources, *i.e.*, the domain topology, and the domain policy to construct the original abstraction. As a result, it is non-trivial to extend it for resource discovery cross-domains, when a job needs to consume resources located in different domains, *e.g.*, the storage node and computation node assigned to the same job may be located in two different domains and are connected by network links across multiple domains. This is because information such as domain topology and policy is usually private to each domain itself and is not allowed to be passed around different domains. In this subsection, we present the details of our design to tackle this challenge and extend resource state abstraction for cross-domain resource discovery.

Basic idea. The key insight of our design is simple yet powerful: if we can partition the networking resources connecting a $(str, comp)$ candidate pair for job j based on the domains they belong to, as shown in Fig. 5, we can then ask the domain resource manager of each domain to compute and represent the resource availability for j in each domain independently.

With this insight, we design the cross-domain resource discovery process of Unicorn whose workflow is shown in Fig. 6. In particular, Unicorn performs cross-domain resource discovery for a set of candidate $(stg, comp)$ pairs for a set of job J in four key steps. The first step is the path query process, in which the Unicorn controller issues path queries to the domain resource manager to recursively get a *domain path* in the form of

$$(dom_1, srcIP, egress) \rightarrow (dom_2, ingress, egress) \rightarrow \ldots, (dom_N, ingress, dstIP), \quad (4)$$

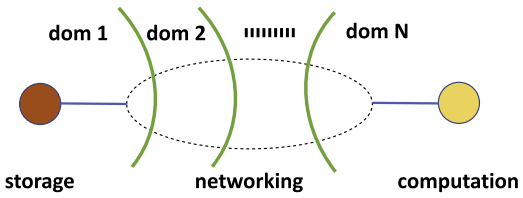


Fig. 5. Partition the networking resources by domain.

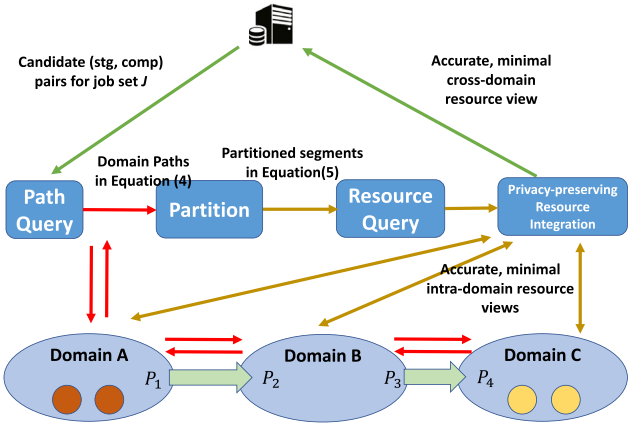


Fig. 6. Workflow of cross-domain resource discovery.

for each candidate $(storage, computation)$ node pair. The path query can be executed either recursively or iteratively. The second step is the partition process, which transforms the domain paths for all the $(stg, comp)$ candidate pairs, into a set of segments, *i.e.*, the partition results, with the form of

$$(dom_i, F_i, F_i.ingress, F_i.egress), \quad (5)$$

for each domain, where F_i denotes the set of all $(stg, comp)$ candidate pairs whose connection use the network resource in domain i. Thirdly, the Unicorn controller sends each partitioned segment to the corresponding domain resource manager to issue one resource query for each segment, which asks each domain to compute the minimal, equivalent single-domain resource state abstraction. Fourthly, a privacy-preserving resource information integration protocol will be executed between all the domains to compute the accurate, minimal cross-domain resource view representing the cross-domain resource availability for a set of candidate $(stg, comp)$ pairs for a set of job J.

Path query. We present the pseudocode of the path query process in Algorithm 1. The path query is a recursive query process. In particular, the path query algorithm requires the input of *domain*, which domain the query should be sent to, F, a set of $(stg, comp)$ candidate pairs whose connection use the network resource in *domain*, and *Ingress*, the set of ingress points each candidate pair is entering *domain* from. It starts from the Unicorn controller group the whole set of F into multiple disjoint subsets based on where the storage resources for this subset of pairs are located, and send one path query for each subset to each corresponding domain. When a domain resource manager receives such a query, it first computes the egress point, the next domain, and the ingress point of next domain for each candidate pair f (Line 3–4). Then the set F is grouped into several disjoint subsets based on the next domain of each pair f (Line 5). For each subset F_i whose next domain is not null, the current resource manager adds the current domain into the domain path for F_i and issues another path query to the domain resource manager at $F_i.nextDom$ to get the remaining part of the whole domain path (Line 8–12). If the next domain of F_i is

null, it means that the computation resources of these (*stg*, *comp*) pairs are in the current domain, *i.e.*, the domain path reaches the destination, and the domain manager simply returns such information to the querying party. During the path query process, each domain only provides the egress points, the next domains and the ingress points for (*stg*, *comp*) candidate pairs without revealing any topology or policy information.

Algorithm 1: The algorithm of path query.

```
1  Function domPathQuery(domain, F, Ingress)
2      domPathResponse ← ∅;
3      foreach f ∈ F do
4          (f.egress, f.nextDom, f.nextDomIngress) ← getNextDomain(f);
5      {F_1, F_2, ..., } ← F.groupBy(f.nextDom);
6      foreach F_i do
7          if F_i.nextDom! = null then
8              domPathResponse ←
9              domPathResponse∪
10             (domain, F_i.egress)⊕
11             {domPathQuery(F_i.nextDom, F_i,
12             F_i.nextDomIngress)};
13         else
14             domPathResponse ←
15             domPathResponse ∪ {(F_i, null)};
16     return domPathResponse;
```

Resource query. For the sake of integrity, we present the pseudocode of partition and resource query together in Algorithm 2. In particular, when the Unicorn controller receives the domain path for each (*stg*, *comp*) candidate pair, it can use this information to partition each path by domains and get the partition results in Eq. (5) (Line 5–12). Then the Unicorn controller can perform efficient resource queries to ask each domain to compute the intra-domain resource view (Line 13–14).

This resource query process is efficient due to the following proposition:

Proposition 2. *Given a set of candidate* (*storage*, *computation*) *node pairs for a job set of J, Unicorn achieves the minimal number of resource queries at each domain.*

Proof. With the domain path for each (*str*, *comp*) candidate pair, the partition process yields a set of segments defined in Eq. (5), one segment for each domain. Hence the Unicorn controller only needs to generate one resource query for each domain if the corresponding F_i is not empty, which completes our proof.

Algorithm 2: The algorithm of partition and resource query and.

```
1  Function resourceQuery(F, F.domainPath)
2      resourceView ← ∅;
3      foreach domain do
4          domain.F ← ∅;
5      foreach f ∈ F do
6          hIdx ← 0;
7          dom ← getDom(f.domainPath, hIdx);
8          do
9              dom.F ← dom.F ∪ {f};
10             hIdx ← hIdx + 1;
11             dom ← getDom(f.domainPath, hIdx);
12         while dom ≠ null;
13     foreach domain do
14         resourceQueryByDomain(domain, F)
```

Privacy-preserving resource information integration. During the resource query phase, each domain d computes the equivalent resource state abstraction that is only minimal to d itself. When the controller collects the resource state abstraction from every domain, a linear inequality that was from domain d_1 may be a redundant one due to the existence of another linear inequality from domain d_2. For instance, d_1 may return $f_1+f_2 \leq 10$ to the controller while d_2 may return $f_1 + f_2 \leq 5$. It is easy to see that the cross-domain minimal, equivalent resource state abstraction would only contain $f_1 + f_2 \leq 5$, not $f_1 + f_2 \leq 10$. A strawman approach to compute the cross-domain minimal, equivalent resource state abstraction is to have the controller run the MECS algorithm with all the resource state abstraction from every domain as input. This approach, however, would force each domain to expose unnecessary resource information, *i.e.*, the redundant linear inequality, to the controller, leading to unnecessary privacy leaks.

In Unicorn, we design a privacy-preserving resource information integration protocol that allows every domain to discover linear inequalities in its own domain that are redundant to the minimal cross-domain resource state abstraction. This protocol involves two steps. In the first step, each domain d uses the classic pivoting algorithm [19] to compute all the vertices of the convex polyhedron defined by all the linear inequalities of its own single-domain resource state abstraction. In the second step, each domain d peers with every other domain $d' \in D$, and uses a customized secure two-party computational geometry protocol to decide if all the vertices computed by d are on the same halfspace defined by a given linear inequality c in the resource state abstraction of d'. If this is true, then c is a redundant inequality in the final cross-domain resource state abstraction, hence will not be sent from domain d' to the controller. The privacy-preserving property of this protocol is summarized in the following proposition.

Proposition 3. *Given two domains d and d′, the proposed protocol ensures that d knows which linear inequalities in its own single-domain resource state abstraction are redundant to the single-domain resource state abstraction of d′ without knowing what the resource state abstraction of d′ has, and vice versa.*

We leave the details of this protocol in [20] due to the space limit.

Schedulability. The cross-domain resource discovery process in Unicorn provides an accurate view of resource availability across domains with minimal exposure of private information. One important question left, however, is whether this view provides a full schedulability of resources for a logically centralized orchestrator. We answer this question with the following theorem.

Theorem 1. *When all the resources represented in the final resource state abstraction queried from the cross-domain discovery process in Unicorn can be fully controlled on the edge, i.e., all the attributes of each resource can be controlled by end host, the resource view provided by resource state abstraction provides a full schedulability of resources to a centralized resource orchestrator.*

We omit the proof of this theorem due to the space limit.

5. Global resource orchestration

With the accurate, minimal cross-domain resource view, Unicorn performs global resource orchestration to compute optimal resource allocation decisions for a given set of jobs J. The modular design of Unicorn allows different allocation algorithms to be deployed. For simplicity, we consider a set of jobs J with no precedence from the same task, *i.e.*, all the jobs can be executed in parallel. We leave a more generic problem formulation as future work. We assume that each computation resource has infinite computation power, *i.e.*, the data accessing delay reading data from storage resources over networking resources to computation resources is the only bottleneck determining the delay for each

workflow. For each job $j \in J$, let Stg_j denote the set of storage resources storing a copy of the input dataset of j, $Comp_j$ denote the set of computation resources that can execute j, v_j denote the volume of input dataset of j, and t_j denote the data accessing delay of j. We also use b_j^{mn} to denote the data access bandwidth for job j from storage resource m to computation resource n, and a binary variable I_j^{mn} to denote if j is assigned storage resource m and computation resource n simultaneously or not. Note that the global resource orchestration component relies heavily on the cross-domain resource discovery component in Section 4. To illustrate this argument, we first give a formulation of the global optimal resource allocation problem *without cross-domain resource discovery* as follows:

$$\text{minimize} \quad max_{j\in J}\{t_j\} \tag{6}$$

subject to

$$\sum_{\{j\in J|n\in Comp_j\}} \sum_{m\in Stg_j} I_j^{mn} \le 1, \qquad \forall n \in N, \tag{7a}$$

$$\sum_{m\in Stg_j} \sum_{n\in Comp_j} I_j^{mn} = 1, \qquad \forall j \in J, \tag{7b}$$

$$\frac{v_j}{\sum_{m\in Stg_j}\sum_{n\in Comp_j} b_j^{mn} I_j^{mn}} = t_j, \qquad \forall j \in J, \tag{7c}$$

$$A_1(BI) \le C_1. \tag{7d}$$

$$A_2(BI) \le C_2. \tag{7e}$$

$$\ldots \tag{7f}$$

$$A_K(BI) \le C_K. \tag{7g}$$

In this formulation, Eq. (6) indicates that the global resource allocation problem aims to minimize the data accessing delay for the whole set of jobs F. Eq. (7a) ensures that for each computation resource, at most one job can be assigned. Eq. (7b) ensures that only one computation resource and one storage resource are assigned for each job j. Eq. (7c) calculates the data accessing delay for each job j. These constraints, *i.e.*, Eqs.(7a)–(7c) are job-specific, *i.e.*, they express the requirements of data analytics jobs and can be changed accordingly based on different job requirements. The constraints in Eqs. (7d)–(7g) are resource-specific, which depends not only on jobs' resource requirements, but also on the attributes provided by resources from each domain.

Though this formulation is accurate itself, its key limitation is that without a cross-domain resource discovery process, it is infeasible to find the resource-specific constraints in Eqs. (7d)–(7g). On the contrary, the cross-domain resource discovery in Unicorn copes with this issue by providing the following constraint to accurately represent the resource availability for a given set of jobs with minimal information exposure.

$$A(BI) \le C. \tag{8}$$

With this formulation, the global optimal resource allocation problem *with cross-domain resource discovery* can be precisely defined as:

$$\text{minimize} \quad max_{j\in J}\{t_j\} \tag{9}$$

subject to

$$\text{Eqs. (7a)(7b)(7c)(8).} \tag{10a}$$

Solution. The multi-domain resource allocation problem defined above is complex in that it involves binary decisions, non-linear constraints and a complex objective function. To solve this problem, we first linearize the binary decision variables, then use a standard optimization solver to find the solution to the relaxed non-linear optimization problem, and then round-up the linearized decision variables back to the {0, 1} feasible space to get the final resource allocation decisions. Because the cross-domain resource discovery process in Unicorn provides the resource view across domains with a minimal set of linear inequalities, the time overhead to solve the relaxed non-linear optimization problem is typically reasonable. We leave the task of finding a more efficient algorithm for this problem as future work.

Table 1
Unicorn resource discovery protocol.

Service	Path Query	Resource Query
HTTP Method	POST	POST
Media Type	`application`	`application`
Accept Subtype	`alto-flowfilter+json`	`alto-flowfilter+json`
Content Subtype	`alto-nextas+json`	`alto-pathvector+json`
Function	Implement *getNextDomain*() in Algorithm 1.	Implement *resourceQueryByDomain*() in Algorithm 2.

6. Implementation

In this section, we discuss the implementation details of the Unicorn framework. The system implementation includes the following components:

Resource discovery protocol. We design and develop a query-based resource discovery protocol by extending the Application-Layer Traffic Optimization (ALTO) protocol [21], to deliver the resource state abstraction from each domain to the Unicorn controller. The protocol provides two major services: *path query service* and *resource query service*. The former is used for delivering next hop information to from domain resource managers the Unicorn controller. The latter is used for executing intra-domain resource queries. Table 1 summarizes the basic view of the two services.

Domain resource manager. We build the prototype implementation of the domain resource manager on top of the OpenDaylight SDN controller [22]. From the view of the Unicorn controller, the domain resource manager works as a web service which provides the resource discovery protocol. From the view of the OpenDaylight controller, the resource manager is a consumer to re-process the topology, the traffic statistics, the intra-domain resource information and the inter-domain routing information.

The implementation includes two sub components: An OpenDaylight application running in the Karaf container; and a Python-based web service to provide the resource discovery protocol. The OpenDaylight application uses the API provided by Model-Driven SAL framework to read the real-time network information from the OpenDaylight DataStore. The two sub components communicate via RPC with each other. So the web service component is decoupled with the OpenDaylight and can be adapted to any other network management platform.

To implement the resource query service, we use the Python web service to look up the raw resource state for the given flow set from the OpenDaylight back end. Our native OpenDaylight application collects the topology and forwarding rules from the `network-topology` and `opendaylight-inventory` model of the DataStore, and computes the intra-domain resource state from these information. In our Python web service, we use GLPK as the underlying LP solver to calculate the minimal equivalent resource state abstraction described in Section 4.1. The solver API is wrapped by PuLP so that we could switch to other LP solvers like CPLEX and Gurobi without many modifications.

We implement the path query service as a BGP compatible service. The domain resource manager reads the inter-domain routing information from the OpenDaylight DataStore and converts it to the *BGP RIB* (Routing Information Base) format to respond the

path query. The native OpenDaylight could support multiple inter-domain routing protocols by implementing their adaptors. In this prototype, we only implement the BGP adaptor which feeds the next-hop information of the inter-domain routing from the `bgp-rib` model.

Cross-domain resource discovery. The cross-domain resource discovery implements the two algorithms, path query (Algorithm 1) and resource query (Algorithm 2) and aggregate resource state abstraction from multiple domains to provide a aggregated resource state abstraction to the Global Resource Orchestration. It provides a high-level API `getGlobalResourceView` which accepts a set of node pairs (*srcIP*, *dstIP*) as the queried flow set, and returns a set of linear inequalities as the global resource view. In addition, it also provides some low-level APIs including: `getDomainPath` that implements the Algorithm 1 and returns the domain path; and `getDomainResource` that retrieves the intra-domain resource view from a domain via resource discovery protocol.

Global resource orchestration. We implement the global resource orchestrator to subscribe to the analytics job management database. Once new jobs are inserted into the database, the orchestrator fetches them, performs cross-domain resource discovery and then make resource allocation decisions. It provides numerous Python APIs for developing new resource allocation algorithms. Therefore it is flexible for administrators to update the resource allocation policy. Our current orchestrator makes resource allocation decisions by solving the optimization problem defined in Section 5.

7. Performance evaluation

We evaluate the performance of Unicorn through trace-based simulations. In particular, we focus on the efficiency of Unicorn in (1) discovering and represent a cross-domain resource view with minimal information exposure; and (2) performing global resource allocation decisions for data analytics jobs. All the simulations are conducted on a laptop with two 1.6 GHz Intel i5 Cores and a 4 GB memory.

7.1. Methodology

We emulate three multi-domain data analytics networks with different number of domains and topologies. For each setting, we first randomly select one topology from Topology Zoo [23] and let that topology be the domain-level topology with each node represent a single domain. And we also generate the intra-domain topology, *i.e.*, switches and the intra-domain links, for each domain. The emulated multi-domain topologies are labeled as Arpanet (composed of 4 domains), Aarnet (composed of 19 domains) and Chinanet (composed of 42 domains). The scale of these multi-domain topologies reflects the scenario of high-energy physics research programs. We leave the evaluation of larger multi-domain topologies (*e.g.*, hundreds or thousands of domains from the CAIDA datasets) as future work. We set the available link bandwidth within each domain to be 0.2–1 Gbps and the available link bandwidth between domains to be 2–4 Gbps. And we assume the I/O bandwidth of storage and computation resources are way larger than the bandwidths of links. We assume each domain's intra-domain and inter-domain routing policies both use the typical routing policies, *i.e.*, the shortest path routing, except that the former is on the router level and the latter is on the domain level. We vary the number of data analytics jobs J from the same task to be from 5 to 30, each of which requires reading 1000 GB of data.

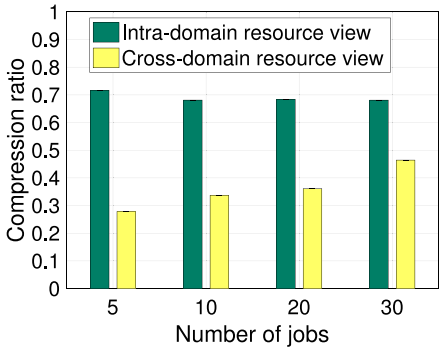


Fig. 7. Compression ratio of intra-domain resource view and cross-domain resource view with varying numbers of jobs.

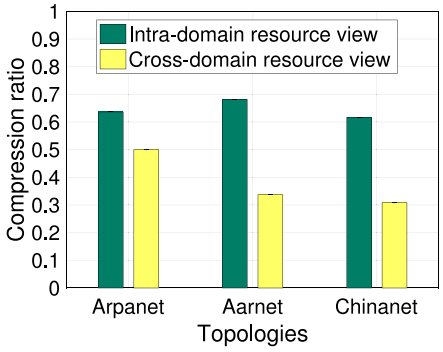


Fig. 8. Compression ratio of intra-domain resource view and cross-domain resource view with different topologies.

7.2. Results

Cross-domain resource discovery and representation. We first present the compression ratio of the Unicorn in discovering and representing the accurate, minimal intra- / cross-domain resource views. This result is computed based on Definition 1 in Section 4.1. Fig. 7 shows this compression ratio in a 19-domain data analytics network derived from the Aarnet topology [23] with different number of data analytics jobs, and Fig. 8 shows this ratio under different number of domains when fixing the number of jobs to be 20. From these results we observe that the average compression ratio of intra-domain resource view is only around 60%–70% while that of the cross-domain resource view is around 25%–45%. These show that Unicorn provides a highly compact view of cross-domain resource availability for data analytics jobs. The higher compression ratio in the cross-domain view is because a multi-domain data analytics network provides more resources for data analytics jobs, *i.e.*, there are fewer jobs sharing the same set of resources. On the other hand, the fact that the highest cross-domain compression ratio is still 45% shows that even with more resources, jobs sharing the same set of resources is still a common situation, indicating the necessity and importance for discovering the accurate, minimal resource availability across domains.

We also plot the number of linear inequalities in the intra-/cross-domain view discovered by Unicorn in Figs. 9 and 10. We see that as the number of domains and the number of jobs grow, the number of linear inequalities in the accurate, minimal resource view computed by Unicorn increases at a very slow rate, which demonstrates the scalability of Unicorn.

Global resource orchestration. We next demonstrate the efficiency of Unicorn in performing global resource orchestration for data analytics jobs. In particular, we focus on the latency of a

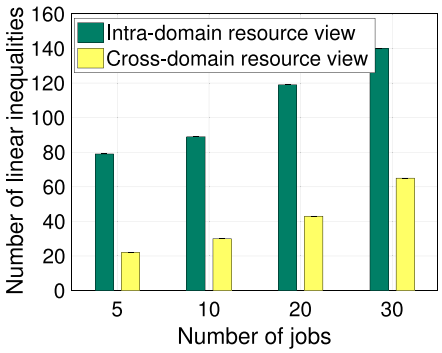


Fig. 9. Number of linear inequalities in intra-domain resource view and cross-domain resource view with varying numbers of jobs.

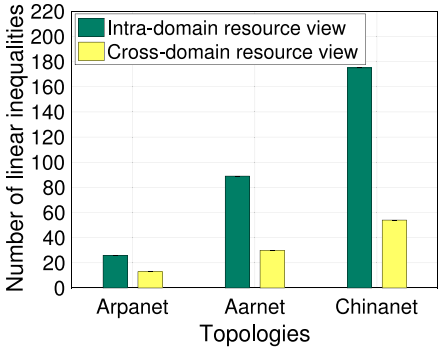


Fig. 10. Number of linear inequalities in intra-domain resource view and cross-domain resource view with different topologies.

Table 2
The reduction of task latency of Unicorn over the domain-path allocation scheme with max–min fairness..

Topology	#Jobs			
	5	10	20	30
Arpanet	31%	24%	27%	65%
Aarnet	27%	46%	55%	10%

task composed of a job set J, which is computed as the longest execution time of all jobs. In our evaluation, we assume all the computation nodes have the same computation power, hence we only need to focus on minimizing the maximal data accessing delay among all jobs, as defined in Eq. (6). We compare the task latency provided by Unicorn with that provided by a domain-path based resource allocation scheme, which allocates computation and storage resources for a job based on the shortest AS path and use the classic max-in fairness mechanism to allocate bandwidth among data accessing flows of analytics jobs. We summarize the results under the combinations of different multi-domain topologies and different numbers of jobs in Table 2. We see that Unicorn provides an up to 65% task latency reduction in all cases. This shows that Unicorn provides a significant latency reduction for multi-domain data analytics.

8. Conclusion and future work

Summary. In this paper, we identify the objective and the fundamental challenge for designing a resource orchestration system for multi-domain, geo-distributed data analytics system through analyzing the data analytics trace from one of the largest scientific experiments in the world and examining the design of existing resource management systems for single-domain clusters. We design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics systems. Unicorn realizes the accurate, cross-domain resource availability discovery with minimal information exposure of each domain through the resource state abstraction and a novel, efficient cross-domain resource availability query algorithm. Unicorn also provides a global resource orchestrator to compute optimal resource allocation decisions for data analytics tasks. We present the implementation details and the preliminary evaluation results of Unicorn.

Prototype and full demonstration at SuperComputing 2017. The source code and more comprehensive evaluation results of Unicorn will be open-sourced at [24]. A full demonstration of the Unicorn prototype has been given at SuperComputing 2017. In this demonstration, we demonstrate the efficiency and efficacy of Unicorn on cross-domain resource discovery and global resource allocation in a multi-domain, geo-distributed data analytics system involving the Caltech booth, the USC booth and the UNESP booth at the conference exhibition, the SCinent network, and the Caltech testbed at Pasadena.

Acknowledgments

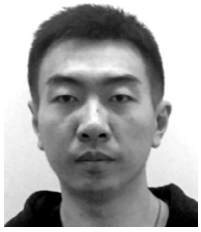
We thank Shenshen Chen, Shiwei Chen, Haizhou Du, and Kai Gao for helpful discussion during the work. The Tongji team is supported in part by NSFC #61702373, #61672385 and #61701347; and China Postdoctoral Science Foundation #2017-M611618. The Yale team is supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions. The Yale team is also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The Caltech team is supported in part by DOE/ASCR project #000219898, SDN NGenIA; DOE award #DE-AC02-07CH11359, SENSE, FNAL PO #626507; NSF award #1246133, ANSE; NSF award #1341024, CHOPIN, NSF award #1120138, US CMS Tier2; NSF award #1659403, SANDIE.

References

[1] The CMS Collaboration, The CMS experiment at the CERN LHC, J. Instrum. 3 (08) (2008) http://dx.doi.org/10.1088/1748-0221/3/08/S08004.
[2] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurr. Comput. Pract. Exp. 17 (2–4) (2005) 323–356.
[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R.H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in: NSDI, 2011.
[4] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache Hadoop YARN: Yet another resource negotiator, in: SoCC, ACM, 2013, p. 5.
[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: EuroSys, ACM, 2015, p. 18.
[6] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, L. Zhou, Apollo: Scalable and coordinated scheduling for cloud-scale computing, in: OSDI, 2014, pp. 285–300.
[7] Under the hood: Scheduling MapReduce jobs more efficiently with Corona, http://on.fb.me/TxUsYN. [Online; accessed: 09-May-2017].
[8] I. Sfiligoi, D.C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, F. Wurthwein, The pilot way to grid resources using glideinWMS, in: CSIE, IEEE, 2009, pp. 428–432.
[9] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, G. Varghese, WANalytics: Analytics for a geo-distributed data-intensive world, in: CIDR, 2015.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, Low latency geo-distributed data analytics, in: SIGCOMM, ACM, 2015, pp. 421–434, http://dx.doi.org/10.475/123.
[11] C.-C. Hung, L. Golubchik, M. Yu, Scheduling jobs across geo-distributed datacenters, in: SoCC, ACM, 2015, pp. 111–124.
[12] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, S. Wang, Rapier: Integrating routing and scheduling for coflow-aware data center networks, in: INFOCOM, 2015.
[13] K. Gao, Q. Xiang, X. Wang, Y.R. Yang, J. Bi, NOVA: Towards on-demand equivalent network view abstraction for network optimization, in: IWQoS 2017, 2017.
[14] D. Jeffrey, G. Sanjay, MapReduce: simplified data processing on large clusters, Commun. ACM (2008).
[15] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: HotCloud'10.
[16] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.G. Chun, V. ICSI, Making sense of performance in data analytics frameworks, in: NSDI, 2015, pp. 293–307.
[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined WAN, in: SIGCOMM'13.
[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: SIGCOMM, ACM, 2013.
[19] D. Avis, K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Comput. Geom. 8 (3) (1992) 295–313.
[20] Privacy-preserving resource information integration: Details, https://www.dropbox.com/sh/6tq5t896etxbvso/AACkQJq_3lMdtOvzhJ0P-6j8a?dl=0.
[21] R. Alimi, Y. Yang, R. Penno, RFC 7285, Application-layer Traffic Optimization (ALTO) Protocol, IETF ALTO, 2014.
[22] J. Medved, R. Varga, A. Tkacik, K. Gray, Opendaylight: Towards a model-driven SDN controller architecture, in: IEEE WoWMoM, 2014.
[23] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The Internet Topology Zoo, 29(9) 1765–1775.
[24] Public Repository of Unicorn, https://github.com/snlab/Unicorn.

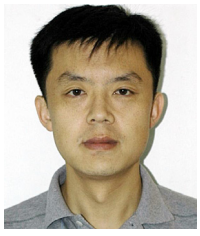
Qiao Xiang is an associate research scientist in the Department of Computer Science at Yale University. His research interests include software defined networking, resource discovery and orchestration in collaborative data sciences, interdomain routing, and wireless cyber-physical systems. From 2014 to 2015, he was a postdoctoral fellow in the School of Computer Science at McGill University. He received his master and Ph.D. degrees in computer science at Wayne State University in 2012 and 2014, respectively, and a bachelor degree in information security and a bachelor degree in economics from Nankai University in 2007.



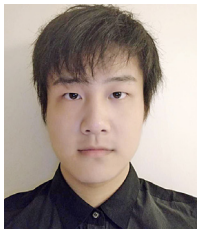
X. Tony Wang is a Ph.D. candidate in the Department of Computer Science and Engineering at Tongji University. His research interests include software defined networking, interdomain routing and distributed computing. He received a bachelor degree in engineering from the Department of Computer Science and Engineering at Tongji University in 2014.

J. Jensen Zhang is a Ph.D. candidate in the Department of Computer Science and Engineering at Tongji University. His research focuses on network resource discovery, abstraction and programming consistency for large-scale data analytics systems. He is also an active member of the IETF ALTO working group and the OpenDaylight open source community. He received a bachelor degree in engineering from the Department of Computer Science and Engineering at Tongji University in 2015.

Harvey Newman (Sc. D, MIT 1974) is the Marvin L. Goldberger Professor of Physics at Caltech, and a faculty member since 1982. In 1973–4 he co-led the team that discovered fourth quark flavor known as "charm". He co-led the MARK J Collaboration that discovered the gluon, the carrier of the strong force in 1979. Since 1994 he has been a member of CMS that discovered the Higgs boson at LHC in 2012. Newman has had a leading role in originating, developing and operating state of the art international networks and collaborative systems serving the high energy and nuclear physics communities since 1982. He served on the IETF and the Technical Advisory Group that led to the NSFNet in 1985–6, originated the worldwide LHC Computing Model in 1996, and has led the science and network engineering teams defining the state of the art in long distance data transfers since 2002.



Dr. Y. Richard Yang is a Professor of Computer Science and Electrical Engineering at Yale University. Dr. Yang's research is supported by both US government funding agencies and leading industrial corporations, and spans areas including computer networks, mobile computing, wireless networking, and network security. His work has been implemented/adopted in products/systems of major companies (e.g., AT&T, Alcatel-Lucent, Cisco, Google, Microsoft, Youku), and featured in mainstream media including Economist, Forbes, Guardian, Chronicle of Higher Education, Information Week, MIT Technology Review, Science Daily, USA Today, Washington Post, and Wired, among others. His awards include a CAREER Award from the National Science Foundation and a Google Faculty Research Award. Dr. Yang's received his B.E. degree in Computer Science and Technology from Tsinghua University (1993), and his M.S. and Ph.D. degrees in Computer Science from the University of Texas at Austin (1998 and 2001).



Y. Jace Liu is a research assistant in the Department of Computer Science and Engineering at Tongji University, China. His research interests include software defined networking, large-scale data analytics systems and high-performance computing. He received a bachelor degree in engineering from the Department of Computer Science and Engineering at Tongji University in 2017.

Fine-Grained, Multi-Domain Network Resource Abstraction as a Fundamental Primitive to Enable High-Performance, Collaborative Data Sciences

Qiao Xiang[♭‡], J. Jensen Zhang[♭], X. Tony Wang[♭], Y. Jace Liu[♭],
Chin Guok[†], Franck Le[◇], John MacAuley[†], Harvey Newman[*], Y. Richard Yang[♭‡],
[♭]Tongji University, [‡]Yale University, [†]Lawrence Berkeley National Laboratory,
[◇]IBM T.J. Watson Research Center, [*]California Institute of Technology,
{qiao.xiang, yry}@cs.yale.edu, {jingxuan.zhang, 13xinwang}@tongji.edu.cn,
yang.jace.liu@linux.com, {chin, macauley}@es.net, fle@us.ibm.com, newman@hep.caltech.edu

***Abstract*—Multi-domain network resource reservation systems are being deployed, driven by the demand and substantial benefits of providing predictable network resources. However, a major lack of existing systems is their coarse granularity, due to the participating networks' concern of revealing sensitive information, which can result in substantial inefficiencies. This paper presents Mercator, a novel multi-domain network resource discovery system to provide fine-grained, global network resource information, for collaborative sciences. The foundation of Mercator is a resource abstraction through algebraic-expression enumeration (*i.e.*, linear inequalities/equations), as a compact representation of the available bandwidth in multi-domain networks. In addition, we develop an obfuscating protocol, to address the privacy concerns by ensuring that no participant can associate the algebraic expressions with the corresponding member networks. We also introduce a super-set projection technique to increase Mercator's scalability. Finally, we implement Mercator and demonstrate both its efficiency and efficacy through extensive experiments using real topologies and traces.**

***Index Terms*—Multi-domain networks, resource discovery, privacy preserving**

I. Introduction

Many of today's premier science experiments, such as the Large Hadron Collider (LHC) [1], the Square Kilometre Array (SKA) [2], and the Linac Coherent Light Source (LCLS) [3], rely on finely-tuned workflows that coordinate geographically distributed resources (*e.g.*, instrument, compute, storage) to enable scientific discoveries. An example of this is the movement of LHC data from Tier 0 (*i.e.*, the data center at European Organization for Nuclear Research, known as CERN) to Tier 1 (*i.e.*, national laboratories) storage sites around the world. This requires deadline scheduling to keep up with the amount of information that is continually generated by instruments when they are online. Another example is the "superfacility" model being developed by LCLS to allow streaming of data from instruments, across the Wide-Area Network (WAN), directly into supercomputers' burst buffers for near real-time analysis. The key to supporting these distributed resource workflows is the ability to reserve and guarantee bandwidth across multiple network domains to facilitate predictable end-to-end network connectivity. As such, several

The corresponding authors are Qiao Xiang and Y. Richard Yang.

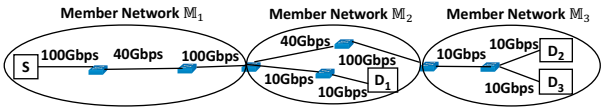


Fig. 1: A motivating example where a user wants to reserve bandwidth for three source-destination pairs: (S, D_1), (S, D_2) and (S, D_3), across 3 member networks $\mathbb{M}_1$, $\mathbb{M}_2$ and $\mathbb{M}_3$.

Research and Education (R&E) networks have deployed inter-domain circuit reservation systems. For example, the Energy Sciences Network (ESnet), a network supporting the LHC experiments, has deployed an On-Demand Secure Circuits and Advance Reservation System called OSCARS [4].

However, due to networks' concern of revealing sensitive information, existing systems do not provide a network interface for users to access network resource information (*e.g.*, network capabilities). Instead, they only allow users to submit requests for reserving a specific amount of bandwidth, and return either success or failure [4]–[10]. This approach, which we call "probe requests" in the rest of this paper, often results in poor performance and fairness. Specifically, while solutions for reserving bandwidth within a single member network, can be very efficient, solutions for discovering and reserving bandwidth for correlated and concurrent flows across multiple member networks face unique challenges. In particular, solutions to reserving bandwidth within a single member network are often provided with the member network's topology, and links' availability. In contrast, because this information is typically considered sensitive, member networks do not reveal internal network details to external parties. As a result, existing multi-domain reservation systems treat each member network as a black box, probe their available resource by submitting varied circuit reservation requests, and receive boolean responses. In other words, current solutions perform a depth-first search on all member networks, and rely on a trial and error approach: to reserve bandwidth, repeated, and varied attempts may have to be submitted until success.

To illustrate the limitations of existing systems, we consider a collaboration network composed of three member networks running OSCARS [4], as shown in Fig. 1. A user may submit a request to reserve bandwidth for three circuits, from source host S to destination hosts D_1, D_2 and D_3. Given the

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00

capabilities of the source host (*e.g.*, the source host may have a 100 Gbps network card), and to ensure fairness across the circuits, the user may request 33.33 Gbps for each circuit. Upon receiving this request, OSCARS processes the circuits sequentially, for example, in the order of (S, D_1), (S, D_2) and (S, D_3). For each circuit, it uses a depth-first search approach to probe if each member network can provide the requested bandwidth. In this example, there is no path with 33.33 Gbps of bandwidth from S to D_1, and hence OSCARS notifies the user that this request fails.

The user can then adjust the requested bandwidth. However, with the limited feedback in OSCARS, the user does not know the amount of available bandwidth from S to D_1. Consequently, the user may use a cut-to-half-until-reserved search strategy. As a result, after 12 attempts, the networks allocate 8.33 Gbps (33.33 $\rightarrow$ 16.67 $\rightarrow$ 8.33) for (S, D_1), 8.33 Gbps (33.33 $\rightarrow$ 16.67 $\rightarrow$ 8.33) for (S, D_2) and 1.04 Gbps (33.33 $\rightarrow$ 16.67 $\rightarrow$ 8.33 $\rightarrow$ 4.17 $\rightarrow$ 2.08 $\rightarrow$ 1.04) for (S, D_3). In addition to requiring a large number of search attempts, the approach may obtain a bandwidth allocation that is far from optimal. For example, given the links' capacities and availability, a fair optimal bandwidth allocation is actually 5 Gbps for each circuit. Without a network interface to provide network resource information, designing an algorithm using existing systems to identify this solution can lead to substantially more complexity and churns.

In addition to multi-domain circuit reservation systems, multiple multi-domain resource discovery systems have been developed and deployed (*e.g.*, [11]–[17]). However, these systems focus on the discovery of endpoint resources (*i.e.*, computation and storage resources) and their availability for different services. They do not provide a network interface for applications to discover the network resource availability and sharing properties [18]–[20].

In this paper, we present Mercator, a novel multi-domain resource discovery system designed to optimize large, multi-domain transfers, and address the limitations of current reservation systems through three main components. The first and core component of Mercator is a resource abstraction through algebraic-expression enumeration (*i.e.*, linear inequalities and equations), which provides a compact, unifying representation of multi-domain network available bandwidth. For example, considering the same example of Fig. 1, the resource abstraction captures the constraints from all networks using the set of linear inequalities depicted in Fig. 2. Specifically, the variables x_1, x_2, x_3 represent the available bandwidth that can be reserved for (S, D_1), (S, D_2) and (S, D_3), respectively. Each linear inequality represents a constraint on the reservable bandwidths over different shared resources by the three circuits. For example, the inequality $x_1 + x_2 + x_3 \leq 100$ indicates that all three circuits share a common resource and that the sum of their bandwidths can not exceed 100 Gbps. With this set of linear inequalities, the user does not need to repeatedly probe the domains, but can immediately derive the bandwidth allocation to satisfy its own objective (*e.g.*, same rate for each transfer, different ratios according to demand ratios, or a fairness allocation such as max-min fairness).

$\mathbb{M}_1$:
$x_1 + x_2 + x_3 \leq 100,$
$x_1 + x_2 + x_3 \leq 40,$
$x_1 + x_2 + x_3 \leq 100,$

$\mathbb{M}_2$:
$x_2 + x_3 \leq 40, \quad x_1 \leq 10,$
$x_2 + x_3 \leq 100, \quad x_1 \leq 10,$

$\mathbb{M}_3$:
$x_2 + x_3 \leq 10,$
$x_2 \leq 10,$
$x_3 \leq 10,$

Fig. 2: Illustration of resource abstraction for the reservation request from Fig. 1.

Second, Mercator introduces a resource abstraction obfuscating protocol to ensure that member networks and other external parties cannot associate an algebraic expression with a corresponding member network, leading to a complete unified aggregation of multiple domains, appearing as much as possible as a single (virtual) network. Although such complete integration may not be needed in all settings, it can be highly beneficial in settings with higher privacy or security concerns. For example, in the scenario of Fig. 1, this protocol ensures that (1) the user cannot infer that the constraint $x_2 + x_3 \leq 10$ comes from network $\mathbb{M}_3$, and (2) that neither network $\mathbb{M}_1$ nor $\mathbb{M}_2$ knows the existence of this constraint. Finally, Mercator also introduces a super-set projection technique, which substantially improves the scalability and performance of Mercator through pre-computation and projection.

The main contributions of this paper are as follows:

- We identify the fundamental reason of the poor performance of current reservation systems for multi-domain data transfers as the lack of visibility of network topology and link availability of each member network, and design Mercator, a novel multi-domain network resource discovery system, to address this issue;
- In Mercator, we propose a novel, compact resource abstraction to represent the network resource availability and sharing, *e.g.*, bandwidth, among virtual circuit requests through algebraic-expression enumeration;
- We design a resource abstraction obfuscating protocol to prevent the user from associating the received algebraic expressions with their corresponding member networks;
- We develop a super-set projection technique to substantially improve the scalability of Mercator;
- We fully implement Mercator and conduct extensive experiments using real network topologies and traces. Results show that Mercator (1) efficiently discovers available networking resources in collaborative networks on average 2 orders of magnitude faster, and allows fairer allocations of network resources; (2) preserves the member networks' privacy with little overhead; and (3) scales to a collaborative network of 200 member networks.

The remaining of this paper is organized as follows. We give an overview of Mercator in Section II. We give the details of the algebraic-expression-based resource abstraction in Section III. We discuss the resource abstraction obfuscating protocol and the super-set projection technique in Section IV and Section V, respectively. We present the evaluation results of Mercator in Section VI. We discuss the related work in Section VII and conclude the paper in Section VIII.

II. Mercator Overview

This section presents the basic workflow and the architecture of Mercator, and a brief overview of its three main components: the resource abstraction through algebraic-expression enumeration, the resource abstraction obfuscating protocol and the super-set projection technique.

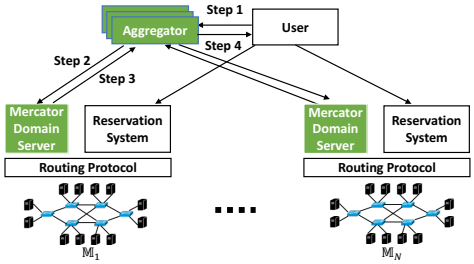


Fig. 3: The architecture and basic workflow of Mercator.

A. Basic Workflow

Mercator introduces and relies on a logically centralized aggregator, and a Mercator domain server in each member network. Consider a multi-domain network of N member networks $\mathbb{M}_i$, where $i = 1, \ldots, N$ (Fig. 3). The basic workflow of Mercator to discover the multi-domain network bandwidth availability and sharing for a set of requested circuits is:

- Step 1: A user (*e.g.*, an application) submits a resource discovery request for a set of circuits to the aggregator by specifying the source and destination endpoints of each circuit.
- Step 2: After authenticating and verifying the authorization of the request, the aggregator determines the member networks that the circuits traverse, and queries the Mercator domain server in each of these member networks to discover their resource abstractions. The determination of the relevant member networks for the aggregator to contact is further described in Section II-B.
- Step 3: Upon receiving the query from the aggregator, each Mercator domain server computes the resource abstraction (Section II-C, Section III) of the corresponding member network, and executes an obfuscating protocol (Section II-C, Section IV) to send the obfuscated resource abstraction to the aggregator.
- Step 4: The aggregator collects the obfuscated resource abstractions from the relevant member networks, and derives the original resource abstractions to present to the user. Based on the received information, the user determines the bandwidth allocation for each circuit, and sends a reservation request to the underlying reservation system.

The above workflow illustrates the main steps for a user to discover the available network bandwidth and properties for a set of circuits traversing multiple member networks. To further improve the scalability of Mercator, Section V introduces the super-set projection technique. It allows the aggregator to proactively discover the resource abstractions for a set of circuits between every pair of source and destination member networks, and project the pre-computed result to get the resource abstraction when receiving actual requests from users. The super-set projection technique can significantly reduce the delay, as well as number of messages, of resource discovery, and allows the aggregator to process multiple requests concurrently.

B. Architecture

This section describes the roles of the aggregator and Mercator domain servers in further details (Fig. 3).

Aggregator: The aggregator is the main interface of Mercator. It is responsible for authenticating and verifying the authorization of users' resource discovery requests (*e.g.*, through PKI [21])[1], querying Mercator domain servers in member networks to discover network resource information, and returning the collected abstractions to users.

The aggregator has connections to Mercator domain servers in all member networks. It also acts as a Border Gateway Protocol (BGP) [24] speaker, and has BGP sessions to all member networks. Consequently, given a request for a set of circuits F, the aggregator can infer the member-network path for each circuit, *i.e.*, the list of member networks a circuit will traverse, and the ingress points of the circuits to each member network[2] (as described in Step 1 of workflow). As such, for this request, the aggregator can also infer the set of circuits traversing and consuming resources in each $\mathbb{M}_i$, denoted as F_i. It can then queries the Mercator domain servers at each $\mathbb{M}_i$ by providing F_i and their ingress points to enter $\mathbb{M}_i$.

Mercator domain server: Given a Mercator domain server in member network $\mathbb{M}_i$, its primary role is to compute the resource abstraction of $\mathbb{M}_i$. To achieve it, Mercator follows the layering design principle to separate the routing protocol and the available network resources. In this way, given a set of circuits sent by the aggregator, their routes in $\mathbb{M}_i$ are computed and provided by the routing protocol in $\mathbb{M}_i$. The Mercator domain server in $\mathbb{M}_i$ takes these routes as inputs, and derives the available bandwidth and shared properties for the requested flows along those routes. After computing the abstraction, the Mercator domain server executes an obfuscating protocol to send the obfuscated resource abstraction to the aggregator, which addresses member networks' privacy concern.

C. Key Design Points

Having illustrated the high-level workflow of Mercator, we next give a brief overview on its key design points.

Resource abstraction through algebraic-expression enumeration (Section III): Mercator follows two important principles in human-computer interaction, *familiarity* and *uniformity*, to design a unifying abstraction that captures the

[1]Mercator may adopt different authentication/authorization systems, *e.g.*, OpenID [22] and SAML [23], depending on the specific requirements of different collaborative science programs. We leave the detailed investigation of this issue in Mercator as future work.

[2]In BGP glossary, such a path is also called an autonomous-system-path, or an AS-path, which is announced in BGP update messages along BGP sessions. The Route View Project [25] relies on a similar architecture with BGP speakers establishing sessions with hundreds of peering networks to collect BGP updates, and provides a real time monitoring infrastructure. In particular, we observe that the AS path for each destination prefix is currently already collected and made publicly available. As such, Mercator does not introduce additional privacy issues.

properties (*e.g.*, available bandwidth) of resources shared – within and between member networks – by a set of requested circuits. This novel, compact resource abstraction is the core component of Mercator, and relies on algebraic expressions (*i.e.*, linear inequalities / equations), a concept familiar to scientists and network engineers [26], to express the available bandwidth sharing for a set of requested circuits to be reserved.

Existing resource abstractions, including graph-based abstractions [27], [28] and the one-big-switch abstractions [29], [30], either fail to protect the private, sensitive information of each member network, or fail to capture the resource sharing between virtual circuit requests. In contrast, the resource abstraction of Mercator, expressed through algebraic-expression enumeration, naturally and accurately captures the available bandwidth of shared resources by a set of circuits without requiring member networks to reveal their network topology. Compared with the Boolean response of current resource reservation systems such as OSCARS, the user receives the complete bandwidth feasible region of the collaboration networks for the requested circuits represented through algebraic expressions. A point in that feasible region represents a feasible allocation of bandwidth for the different circuits in the request. In other words, the user can choose any point in the returned region as the bandwidth parameters for the circuits to be reserved, using his own resource allocation strategy (*e.g.*, max-min fairness [31]).

Resource abstraction obfuscating protocol (Section IV): The algebraic-expression-based abstraction provides a compact, unifying representation of the multi-domain network resource information. It does not require member networks to reveal their network topologies and link availabilities. However, it does expose the bandwidth feasible region of each member network (illustrated by the examples in Section I and Section III). Some member networks might prefer not to expose such information, as malicious parties may use it to identify links where to launch attacks (*e.g.*, DDoS). To address this issue, we develop a resource abstraction obfuscating protocol. More specifically, the protocol prevents the resource discovery aggregator from identifying the source of each received resource constraint. The key idea consists of having each Mercator domain server obfuscate its own set of linear inequalities as a set of linear equations through a private random matrix of its own and a couple of random matrices shared with few other Mercator domain servers from other member networks (*e.g.*, through a consensus protocol), and then sends the obfuscated set of linear equations back to the aggregator using symmetric-key encryption, *e.g.*, Advanced Encryption Standard (AES) [32]. We demonstrate that from the received obfuscated equations, the aggregator can retrieve the actual bandwidth feasible region for the circuits across member networks, but cannot associate any linear inequality with its corresponding member network. As a result, even if a malicious party obtains the bandwidth feasible region across member networks, launching attacks to all member networks is much harder than attacking a particular member network.

Super-set projection (Section V): To improve the scalability of Mercator, we introduce the super-set projection technique. The main idea consists of having the aggregator periodically query Mercator domain servers to discover the resource abstraction for a set of circuits between every pair of source and destination member networks. With these precomputed abstractions, when a user submits a resource discovery request, the aggregator does not need to query the Mercator domain servers to compute the abstraction for each received request. Instead, the aggregator performs a projection on the precomputed abstractions based on the source and destination member networks of each circuit in the actual user request, to get the abstraction for this request. For example, consider a network of 2 member networks $\mathbb{M}_1$ and $\mathbb{M}_2$. Using super-set projection, the aggregator queries the Mercator domain servers at both member networks for a set of 2 circuits, one from $\mathbb{M}_1$ to $\mathbb{M}_2$ and the other from $\mathbb{M}_2$ to $\mathbb{M}_1$, and gets a set of linear inequalities $\{x_{12} + x_{21} \leq 100,\ x_{12} \leq 50\}$. Suppose later a user submits a request for 1 circuit, with the source being an endpoint in $\mathbb{M}_2$ and the destination being an endpoint in $\mathbb{M}_1$, to the aggregator. The aggregator projects the precomputed set of linear inequalities by removing all variables that are not x_{21}, and returns the result $\{x_{21} \leq 100\}$ to the user.

Such projection is much more efficient than having Mercator domain servers compute the abstraction for each received circuit request. With this technique, when a user submits a resource discovery request to the aggregator, the aggregator does not need to query Mercator domain servers (Step 2 in Section II-A), and the Mercator domain servers do not need to compute and obfuscate the resource abstraction for the request (Step 3 in Section II-A). Only when the user fails to reserve the resource based on the projected abstraction will the aggregator query the Mercator domain servers to obtain an up-to-date abstraction for the user. As such, servers in the aggregator pool can process requests concurrently (*e.g.*, using optimistic concurrency control), significantly improving the scalability, fault-tolerance, and performance of Mercator.

After an overview of the key design points in Mercator, we discuss these designs in detail in the next few sections.

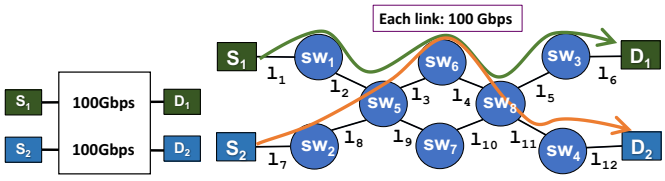
III. Resource Abstraction through Algebraic-Expression Enumeration

In this section, we give the details of the resource abstraction through algebraic-expression enumeration, the core component of Mercator. We first discuss the limitations of existing design options. Then we give the specifications of this abstraction. We also discuss how it handles important use cases, *e.g.*, multicast, multi-path routing and load balancing, in Appendix A.

Basic issue: As illustrated by the example in Section I, the fundamental reason for the poor performance of existing circuit reservation systems is they are lack of the visibility of properties, *e.g.*, bandwidth, of shared network resources for a set of circuits to be reserved. One may think of a strawman to let each member network provide the full topology information to the aggregator in a graph-based abstraction [27], [28]. This design, however, exposes all the sensitive, private information

of each member network, *i.e.*, network topology and links' availability, to external parties, leading to security breaches.

A second strawman is to use a one-big-switch abstraction to provide simplified views of network information [29], [30], which protects the privacy of each member network. However, this abstraction fails to capture the information of shared resource among virtual circuit requests and thus is inaccurate. Consider the example in Fig. 4, where the user wants to reserve two circuits from S_1 to D_1 and S_2 to D_2, respectively. Using the one-big-switch abstraction in the P4P system [29], the user will get the information that each circuit can reserve a bandwidth up to 100 Gbps (Fig. 4a). However, the routes for the two circuits – computed by the underlying routing protocol – share common links l_3 and l_4 (Fig. 4b), making it infeasible for both circuits to each reserve a 100 Gbps bandwidth.



(a) The one-big-switch shows that each circuit can get a 100 Gbps bandwidth. **(b)** The physical topology shows that the route of two circuits share bottleneck links, *i.e.*, l_3 and l_4, hence they can only collectively get a 100 Gbps bandwidth.

Fig. 4: A running example for illustrating the inefficiency of one-big-switch abstraction and the basic idea of resource abstraction through algebraic-expression enumeration, where two circuits (S_1, D_1) and (S_2, D_2) need to be reserved.

In some recent studies [33], [34], a variation of the one-big-switch abstraction was proposed to define the resource sharing among different traffic flows as operations defined in different algebra fields. However, this abstraction is too complex and can only handle single-path routing policies.

Basic idea: Different from the graph-based abstraction and the one-big-switch abstraction, the basic idea of the resource abstraction in Mercator is simple yet powerful: *given a set of requested circuits to be reserved, capture the properties (e.g., available bandwidth) of relevant shared resources, through a set of algebraic expressions.*

Specifically, suppose the Mercator domain server at a member network receives the resource discovery request of a set of circuits F entering this member network. For each circuit $f_j \in F$, we use x_j to denote the available bandwidth the user can reserve for this circuit. Upon receiving this request, the Mercator domain server first checks the intradomain route of each circuit f_j. Then the server enumerates all the links in the member network. For each link l_u, it generates a linear inequality:

$$\sum x_j \leq l_u.bandwidth, \forall f_j \text{ that uses link } l_u \text{ in its route.}$$

Revisit the example in Fig. 4, the Mercator domain server will generate the following set of linear inequalities $\Pi(F)$:

$$\begin{array}{ll} x_1 \leq 100 & \forall l_u \in \{l_1, l_2, l_5, l_6\}, \\ x_2 \leq 100 & \forall l_u \in \{l_7, l_8, l_{11}, l_{12}\}, \\ x_1 + x_2 \leq 100 & \forall l_u \in \{l_3, l_4\}, \end{array} \quad (1)$$

which accurately captures the bandwidth sharing among two circuits' routes.

Removing redundant linear inequalities: Observe the set of linear inequalities in the above example. One may realize that this set has redundancies, *e.g.*, there are 4 same inequalities $x_1 \leq 100$ in this set. Given $\Pi(F)$, a linear inequality $y \in \Pi(F)$ is redundant if and only if the optimal solution of any optimization problem with $\Pi(F)$ as the constraint is the same as that with $\Pi(F) - \{c\}$ as the constraint. In our system, the Mercator domain server adopts a classic compression algorithm [35] to remove the redundant linear inequalities. In this example, the compressed $\Pi(F)$ will only contain one inequality, *i.e.*, $x_1 + x_2 \leq 100$.

Geometric interpretation of resource abstraction: Given $\Pi_i(F)$, the resource abstraction of $\mathbb{M}_i$ for a set of F circuits, from the geometric perspective, represents the bandwidth feasible region of $\mathbb{M}_i$ for providing bandwidths to this set of circuits. Therefore, given a set of F circuits spanning over N member networks, the union of $\Pi_i(F_i)$, where $F_i \subset F$ is the set of circuits that will consume resources in $\mathbb{M}_i$, represents the complete bandwidth feasible region of all N member networks for the requested circuits.

Through algebraic-expression enumeration, the resource abstraction can handle not only unicast, as shown above, but many other settings. In Appendix A, we show how it handles three important use cases in collaborative data sciences.

IV. Privacy-Preserving Resource Abstraction

Given a member network, the algebraic-expression-based resource abstraction accurately captures the shared available bandwidth among virtual circuits without exposing its network topology and links' availability. However, as shown in Section III, the geometric interpretation of a resource abstraction is that it represents the bandwidth feasible region of the corresponding member network for a set of circuits. Such information is still private and sensitive, and a malicious party who acquires it may use it to launch attacks to the corresponding member network. To preserve the privacy of bandwidth feasible region of member networks while still providing the accurate bandwidth sharing information for circuits, we develop a resource abstraction obfuscating protocol in Mercator. In this section, we first formally define the privacy-preserving resource abstraction problem. Next, we present the details of our protocol. We also conduct a rigorous analysis of our protocol in Appendix B.

A. Privacy-Preserving Resource Abstraction Problem

Basic issue: We use the example in Fig. 5 to illustrate the privacy concern of the resource abstraction, where Mercator tries to discover the shared bandwidth of two virtual circuits (S_1, D_1) and (S_2, D_2) across 3 member networks. In this example, all links in black line are 1 Tbps aggregating links. The inter-member-network-paths of two circuits are $[\mathbb{M}_1, \mathbb{M}_2]$ and $[\mathbb{M}_1, \mathbb{M}_3]$, respectively. And two circuits share the same intra-domain path in $\mathbb{M}_1$.

When receiving the resource discovery request, the Mercator domain server at each member network will abstract the bandwidth sharing of both circuits into a set of linear inequalities. After removing the redundant inequalities of each member

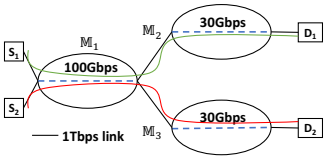


Fig. 5: A running example to illustrate the resource abstraction obfuscating.

network, the resource abstraction of each member network is:

$$\begin{aligned}&\Pi_1(F_1):\{x_1+x_2\leq 100\}\\&\Pi_2(F_2):\{x_1\leq 30\}\\&\Pi_3(F_3):\{x_2\leq 30\}.\end{aligned}\tag{2}$$

If each Mercator domain server directly sends its own resource abstraction to the aggregator, the aggregator will have the knowledge of the bandwidth feasible region of each individual member network. This makes the whole collaboration network vulnerable because the aggregator is a single point of failure possessing the private information of all member networks. In other words, if an attacker gains the control to the aggregator, he can leverage such specific information to attack any member network.

Problem definition: To make Mercator functional and secure, therefore, we need a solution that provides the accurate bandwidth sharing information for the set of virtual circuits to be reserved, and at the same time protects each member network from exposing its private bandwidth feasible region. To this end, we first give a formal definition of privacy-preserving, equivalent resource abstraction:

***Definition** 1 (Equivalent, Privacy-Preserving Resource Abstraction):* Given a set of circuits F that span over $N>1$ member networks, the resource abstraction $\Pi_p(F)$ collected by the aggregator is equivalent and privacy-preserving if (1) the bandwidth feasible region represented by $\Pi_p(F)$ is the same as that represented by $\cup_i\Pi(F_i)$ where $i=1,2,\ldots,N$; and (2) for any linear inequality $c\in\Pi_p(F)$, the aggregator cannot associate it with a particular member network.

In this definition, $\Pi(F_i)\cup\Pi(F_j)$ means the union of two sets of linear inequalities. Geometrically speaking, it means the intersection of the feasible regions represented by $\Pi(F_i)$ and $\Pi(F_j)$. With this definition, we further define the privacy-preserving resource abstraction problem:

***Problem** 1 (Privacy-Preserving Resource Abstraction Problem):* Given a set of circuits F that span over $N>1$ member networks, design a security protocol in the resource discovery system to ensure that (1) the aggregator receives the equivalent, privacy-preserving resource abstraction $\Pi_p(F)$; and (2) for any $\mathbb{M}_i$, it does not know any linear inequality from any other $\Pi_j(F_j)$, where $j\neq i$.

Security model: In this paper, we assume a *semi-honest* security model, *i.e.*, the aggregator and all member networks will not deviate from the security protocol, but merely try to gather information during the execution of the protocol [36]. This is sufficient for collaboration science networks where member networks share resources to collaboratively conduct common tasks such as data transfers, storage and analytics.

B. Resource Abstraction Obfuscating Protocol

There are different design options for Problem 1, *e.g.*, garbled circuit based protocols [37]. However, these designs would incur expensive computation and communication overhead, hence are not suitable for the need of multi-domain resource discovery. In this paper, we tackle this problem by designing a novel resource abstraction obfuscating protocol that only requires simple operations on matrices, *i.e.*, addition and multiplication.

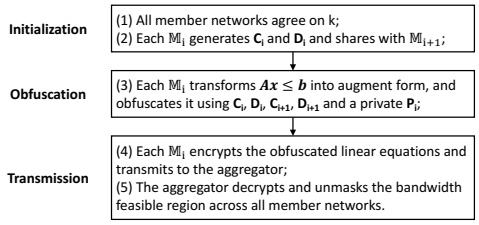


Fig. 6: The resource abstraction obfuscating protocol.

Basic idea: Our protocol leverages random matrix theory [38], [39]. In particular, each $\mathbb{M}_i$ independently computes and sends to the aggregator a set of disguised *linear equations*, which are derived from the private $\Pi_i(F_i)$, a random matrix $\mathbf{P_i}$ known only to $\mathbb{M}_i$, two random matrices $\mathbf{C_i}$ and $\mathbf{D_i}$ known only to $\mathbb{M}_i$ and $\mathbb{M}_{i-1}$, and two random matrices $\mathbf{C_{i+1}}$ and $\mathbf{D_{i+1}}$ known only to $\mathbb{M}_i$ and $\mathbb{M}_{i+1}$.

Protocol: The protocol is composed of three phases: initialization, obfuscation and transmission, as shown in Fig. 6. For the simplicity of presentation, we let $m_i=|\Pi_i(F_i)|$, *i.e.*, the number of linear inequalities in $\Pi_i(F_i)$ after redundancy removal, and $M_i=\sum_{j=1}^{i}m_j$. During the *initialization phase*, all member networks agree on a common $k>\sum m_i$. For each $\mathbb{M}_i$ where $i=1,2,\ldots,N-1$, it generates a k-by-$(|F|+m_i+m_{i+1})$ random matrix $\mathbf{C_i}=[\mathbf{C_i}^{|\mathbf{F}|}\ \mathbf{C_i}^{m_i}\ \mathbf{C_i}^{m_{i+1}}]$, and a k-by-1 random matrix $\mathbf{D_i}$, and sends to $\mathbb{M}_{i+1}$. And we define $\mathbf{C_0}$, $\mathbf{D_0}$, $\mathbf{C_N}$ and $\mathbf{D_N}$ as zero matrices. As we will illustrate in the remaining of this section, these zero matrices are used for presentation completeness and will not affect the correctness of the obfuscating protocol.

During the *obfuscation phase*, each $\mathbb{M}_i$ introduces m_i slack variables, denoted by $\mathbf{x_i^s}$, to transform $\Pi_i(F_i)=\mathbf{A_ix}\leq\mathbf{b_i}$ from the standard form to the augment form and gets the following equivalent linear system:

$$[\mathbf{A_i}\quad\mathbf{I_{m_i}}]\,[\mathbf{x},\ \mathbf{x_i^s}]=\mathbf{b_i}.\tag{3}$$

We then add slack variables introduced by all other member networks with zero coefficients into the linear system in Equation (3) and get the following equivalent linear system:

$$[\mathbf{A_i}\quad\mathbf{0_{M_{i-1}}}\quad\mathbf{I_{m_i}}\quad\mathbf{0}]\,[\mathbf{x},\ \mathbf{x_1^s},\ \ldots,\ \mathbf{x_i^s},\ \ldots,\ \mathbf{x_N^s}]=\mathbf{b_i}.\tag{4}$$

Next, each $\mathbb{M}_i$ generates a private random matrix $\mathbf{P_i}\in R^{k\times m_i}$, and left-multiplies both sides of Equation (4) to get:

$$[\mathbf{P_iA_i}\quad\mathbf{0_{M_{i-1}}}\quad\mathbf{P_i}\quad\mathbf{0}]\,[\mathbf{x},\ \mathbf{x_1^s},\ \ldots,\ \mathbf{x_i^s},\ \ldots,\ \mathbf{x_N^s}]=\mathbf{P_ib_i}.\tag{5}$$

Then each $\mathbb{M}_i$ adds

$$[\mathbf{C_i}^{|\mathbf{F}|}-\mathbf{C_{i-1}}^{|\mathbf{F}|}\quad\mathbf{0_{M_{i-2}}}\quad-\mathbf{C_{i-1}}^{\mathbf{m_{i-1}}}\quad-\mathbf{C_{i-1}}^{\mathbf{m_i}}+\mathbf{C_i}^{\mathbf{m_i}}\quad\mathbf{C_i}^{\mathbf{m_{i+1}}}\quad\mathbf{0}],$$

to the coefficient matrix of the left-hand-side (LHS) of Equation (5), and adds $-\mathbf{D_{i-1}}+\mathbf{D_i}$ to its right-hand-side (RHS) to get Equation (6) where it can be observed that for each $\mathbb{M}_i$,

$$\left[\mathbf{P_i A_i} + \mathbf{C_i^{|F|}} - \mathbf{C_{i-1}^{|F|}} \quad \mathbf{0_{M_{i-2}}} \quad -\mathbf{C_{i-1}^{m_{i-1}}} \quad \mathbf{P_i} - \mathbf{C_{i-1}^{m_i}} + \mathbf{C_i^{m_i}} \quad \mathbf{C_i^{m_{i+1}}} \quad \mathbf{0}\right] \cdot \left[\mathbf{x},\ \mathbf{x_1^s},\ \ldots,\ \mathbf{x_i^s},\ \ldots,\ \mathbf{x_N^s}\right] = \mathbf{P_i b_i} - \mathbf{D_{i-1}} + \mathbf{D_i}, \quad (6)$$

the coefficient matrix of LHS of Equation (6) is of dimension k-by-$|F| + M_N$, and the RHS is of dimension k-by-1.

In the *transmission phase*, each $\mathbb{M}_i$ encrypts the set of linear equations in Equation (6) using a symmetric-key algorithm, *e.g.*, AES, and sends the cypher text to the aggregator. After collecting the linear equations from all member networks, the aggregator decrypts them and computes the sum of all LHS matrices and RHS matrices of all member networks, respectively. After simple elimination, the LHS sum is expressed as: $[\sum \mathbf{P_i A_i} \quad \mathbf{P_1} \quad \ldots \quad \mathbf{P_N}]$. Similarly, the sum of all RHS matrices of all member networks can be expressed as $\sum \mathbf{P_i b_i}$. Denoting $[\mathbf{x_1^s}, \ldots, \mathbf{x_N^s}]$ as $\mathbf{x^s}$, the aggregator can get the privacy-preserving abstraction $\Pi_p(F)$:

$$[\textstyle\sum \mathbf{P_i A_i} \quad \mathbf{P_1} \quad \ldots \quad \mathbf{P_N}.] \left[\mathbf{x},\ \mathbf{x^s}\right] = \sum \mathbf{P_i b_i}. \quad (7)$$

Example: We use the example in Fig. 5 to illustrate the resource abstraction obfuscating protocol. For simplicity, we assume three member networks agree on $k = 4$. The private random matrices $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ are generated as $\mathbf{P_1} = [11, 49, 95, 34]$, $\mathbf{P_2} = [58, 22, 75, 25]$, and $\mathbf{P_3} = [50, 69, 89, 95]$. The resource abstraction $\Pi_p(F)$ obtained by the aggregator is:

$$\begin{aligned} 69x_1 + 61x_2 + 11x_{11}^s + 58x_{21}^s + 50x_{31}^s &= 4340, \\ 71x_1 + 118x_2 + 49x_{11}^s + 22x_{21}^s + 69x_{31}^s &= 7630, \\ 170x_1 + 184x_2 + 95x_{11}^s + 75x_{21}^s + 89x_{31}^s &= 14420, \\ 59x_1 + 129x_2 + 34x_{11}^s + 25x_{21}^s + 95x_{31}^s &= 7000, \end{aligned}$$

where x_{11}^s, x_{21}^s and x_{31}^s are slack variables. Assume the user's objective is to maximize the throughput, *i.e.*, $x_1 + x_2$. Using this set of linear inequalities as the constraint, it can get the optimal solution where $x_1 = x_2 = 30$ Gbps, the same as when using Equation (2) as the constraint.

We conduct rigorous analysis on different properties (*e.g.*, correctness, security and efficiency) of the proposed obfuscating protocol, which can be found in Appendix B.

V. Super-Set Resource Abstraction Projection

One concern of the resource discovery is its scalability, as the number of resource discovery requests may be large in collaboration networks and each request would trigger a resource discovery procedure. This procedure requires the communication between the aggregator and the user, and between the aggregator and every Mercator domain server in member networks. Furthermore, the introduction of resource abstraction obfuscating further increases the communication and computation overhead of resource discovery. To address this issue, we develop a novel super-set projection technique. We describe its basic idea in this section, and leave the details of this technique in Appendix C.

Basic idea: The intuition of super-set projection is simple: to have the aggregator proactively discover the resource abstraction for a set of circuits between every pair of source and destination member networks, and use these pre-computed abstractions to quickly project to get the resource abstraction for user's requests.

In particular, in a collaboration network of N member networks, the super-set projection technique first simulates the need of $N(N-1)$ artificial circuits, where each circuit f_{ij} represents an artificial circuit from $\mathbb{M}_i$ to $\mathbb{M}_j$. With this artificial resource discovery request, the aggregator follows the normal resource discovery process to discover the shared bandwidth of all these $N(N-1)$ circuits across the whole collaboration network, represented by Π_{full}. When a user sends an actual resource discovery request for a set of F circuits, the aggregator checks the source and destination member networks of each circuit, and uses the stored Π_{full} to derive $\Pi(F)$ by removing unrelated inequalities and unrelated artificial circuits, instead of starting a new resource discovery procedure. In this way, the overhead of resource discovery is reduced to a single round of message exchange between the aggregator and the user.

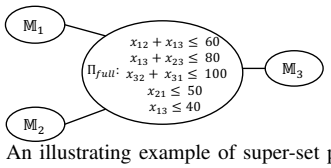


Fig. 7: An illustrating example of super-set projection.

Example: Consider an example of 3 member networks in Fig. 7. With the super-set projection, the aggregator discovers the bandwidth sharing of all $3 \times 2 = 6$ network-to-network artificial circuits as Π_{full} in the figure. When a user submits a resource discovery request for two circuits (S_1, D_1) and (S_2, D_2), where S_1 is in $\mathbb{M}_1$, S_2 and D_1 are in $\mathbb{M}_2$ and D_2 is in $\mathbb{M}_3$. The aggregator first maps the (S_1, D_1) to the artificial circuit from $\mathbb{M}_1$ to $\mathbb{M}_2$, and (S_2, D_2) to the artificial circuit from $\mathbb{M}_2$ to $\mathbb{M}_3$. Next, it projects Π_{full} to these two circuits to get the resource abstraction for these two circuits by (1) removing all linear inequalities that do not contain x_{12} or x_{23}, and (2) for every remaining linear inequality, remove all the items on the LHS that are not x_{12} or x_{23}. Finally, it returns the resource abstraction: $\{x_{12} \leq 60,\ x_{23} \leq 80\}$, to the user.

VI. Evaluation

We implement Mercator on commodity servers (*i.e.*, equipped with Intel(R) Xeon(R) E5-2609 2.50GHz 4-core CPU and 32 GB memory) and evaluate its performance based on a member-network-level topology from a large federation of networks supporting large-scale distributed science collaborations, and using real traffic traces from recent science experiments. After describing our experimental setup, we first demonstrate the benefits of resource abstraction through algebraic-expression enumeration. Second, we demonstrate the efficiency of the proposed resource abstraction obfuscation protocol. Finally, we demonstrate that the super-set projection technique substantially increases the scalability of Mercator.

A. Experimental Setup

We evaluate Mercator on the member-network-level topology from LHC Open Network Environment (LHCONE), a global science network consisting of 62 member networks, where scientists conduct large-scale distributed analytics. Because inter-member-network routing typically is not based

on shortest path routing, but follows business relationships (*e.g.*, customer, peer, provider), we label the connections between every pair of connected member networks with their business relationship using the CAIDA network relationships dataset [40], and we compute the inter-member-network paths according to conventional policies for selecting and exporting routes. For member networks' intradomain topologies, we randomly select a topology for each network from the Topology Zoo [41], which provides a collection of real intradomain topologies. The topology of transit member networks varies from 31 switches/routers with 33 links to 49 switches/routers with 85 links. The topology of stub member networks (*e.g.*, campus science networks) ranges from 7 switches/routers with 6 links to 21 switches/routers with 44 links.

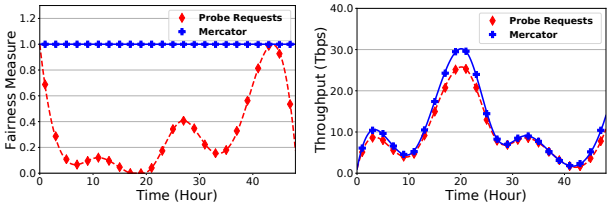
B. Benefits of Resource Abstraction Through Algebraic-Expression Enumeration

The first set of experiments demonstrate the benefits of the resource abstraction through algebraic-expression enumeration. We show that this abstraction reduces the time to discover network resources by up to 3 orders of magnitude, and allows fairer allocations of network resources.

1) Methodology: To evaluate the benefits of this resource abstraction, we replay the trace from a large-scale distributed experiment, and submit network resource reservations for the corresponding flows. More specifically, we use the actual trace from the CMS experiment [42], a major scientific experiment in LHC, and a main source of traffic in LHCONE. We extract the traffic flows, with their source member network, destination member network and the time. We focus on the 48-hour trace starting from December 14, 2017 and slice the data trace into 24 continuous 2-hour time windows. We apply the resources reservation once every time window. In other words, resources for traffic flows starting at the same time window are reserved in the same request, and we assume all resources will be released in the next time window.

We compare the performance of Mercator with that of existing reservation systems. In particular, for existing systems, we consider one that adopts a probe-requests based approach:

- **Mercator**: As described in Section II, for every resource discovery request, the aggregator queries the relevant member networks for their resource abstraction, and then derives the feasible bandwidth allocation region.
- **Probe requests**: As described in Section I, existing resource reservation systems such as OSCARS process each circuit in the request one at a time and in a sequential order. For each circuit, the resource reservation system initiates a depth-first search to probe if each member network can provide the requested bandwidth. We set the initial requested bandwidth for a circuit as C/N where C is the source host's capacity, and N is the number of flows from that host. In the event of a failure, the resource reservation system performs a binary search of the available bandwidth repeatedly halving the requested bandwidth until success. The process is repeated for each circuit in the request.



(a) Obj. = max-min fairness **(b)** Obj. = max total throughput

Fig. 8: Comparison of performance between the probe-requests approach and Mercator in different objectives.

2) Results: First, we consider that the goal of the resource allocation policy is to maximize the minimum throughput of all the requested flows (max-min fairness). Such a policy is commonly desired as it ensures high throughput and fairness across the circuits. We compare the fairness of the network resource allocations obtained with Mercator to that obtained with the probe-requests based solution. We adopt Jain's fairness index [43] to measure the fairness [31]:

$$J(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

where x_i is the ratio of the actual allocation and the optimal fair allocation for a single flow. Fig. 8a shows that with resource abstraction, Mercator can always compute the optimal max-min fairness allocation. Hence its fairness index is always 1. In contrast, the fairness index of the probe-requests based solution has an average of 0.37, and even drops to 0 at times.

Second, we consider the case where the objective is to maximize the total throughput. Fig. 8b shows that the total throughput of Mercator is larger than that obtained by the probe-requests based solution, by 15% on average, and up to 20%. The results are noteworthy given that Mercator assumes the routes for each circuit to be completely determined by the underlying intradomain routing protocol. In contrast, the probe-requests approach sequentially explores every possible route for each circuit until it finds an available one. In other words, even with much less exploration, Mercator outperforms the probe requests. Allowing Mercator to consider not only the routes provided by the underlying routing protocols, but also all other available routes, could lead to significant additional improvements. We leave the extension of Mercator to consider all possible routes in the network as future work.

Fig. 9 presents the total resource discovery latency for completing all circuits resource reservations in a time window. We assume the aggregator to be in New York, and consider network latencies as measured in [44]. The figure shows the total resource discovery latency with Mercator can reduce the time to discover network resources by two orders of magnitude on average and up to three orders of magnitude at times. This is because resource abstraction allows users to query the information from different member networks in parallel. In contrast, existing probe-requests based solutions process requests sequentially, and continuously probe to discover the available network resources.

Finally, we highlight that the probe-requests based solution can suffer high request failure ratio, *i.e.*, a large number of requests cannot succeed: We define a failure of a request as the

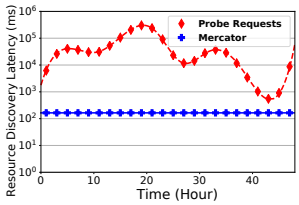


Fig. 9: Resource discovery latency of the probe-requests approach and Mercator.

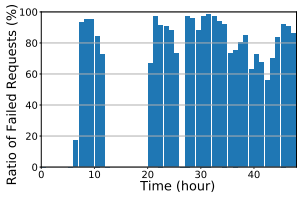
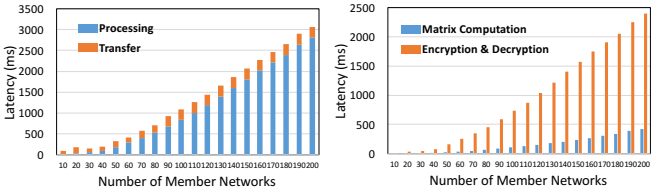


Fig. 10: Ratio of failed requests in the probe-requests approach.



(a) Overall latency. **(b)** Processing latency.

Fig. 11: The latency of the resource abstraction obfuscating protocol.

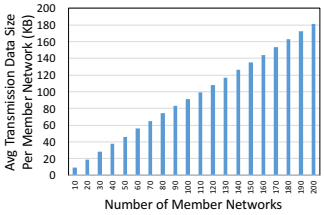


Fig. 12: The data transmission overhead of the resource abstraction obfuscating protocol.

inability to reserve resource for the circuit, due to the lack of remaining capacity despite the gradually decreasing requested bandwidth. Fig. 10 shows that during the 48-hour period Mercator is running, the probe-requests based solution has an average request failure ratio of 73%. In other words, more than 70% of the circuits cannot reserve network resources. This is because the probe-requests approach processes the request for each circuit sequentially. Therefore, the first few circuits may successfully reserve network resources and saturate the network. As such, despite achieving a total throughput similar to Mercator, the majority of the latter requests may fail as the links do not have any spare resources. In contrast, the request failure ratio of Mercator is null because Mercator returns a feasible region for the set of circuits so that the user can make optimal reservation decisions for all circuits.

C. Efficiency of Resource Abstraction Obfuscating Protocol

This second set of experiments evaluate the performance of the resource abstraction obfuscating protocol. We show that this protocol efficiently scales for collaboration networks of 200 member networks, with a maximal overall latency around 3 seconds and an average data transmission overhead between the aggregator and member networks of only around 180 KB.

1) Methodology: We conduct our experiment by using the member-network-level topology from the LHC Open Network Environment (LHCONE). In each round of the experiment, we randomly select a set of member networks from the topology. For each chosen member network, we randomly select a set of m linear inequalities, where m is randomly chosen between 5 and 15, to represent the bandwidth feasible reason for 10 circuits in this member network. For the encryption and decryption operations in the obfuscating protocol, we use the AES algorithm, provided by the Python Cryptography Toolkit (pycrypto) [45]. The parameters k, $\mathbf{C_i}$ and $\mathbf{D_i}$ are pre-configured as discussed in Appendix B.

We consider two metrics, *i.e.*, the latency and the data transmission overhead of the resource abstraction obfuscating protocol. First, the overall latency of the protocol is measured from the beginning of the obfuscation phase, when each member network independently starts to obfuscate its own set of linear inequalities, to the end of the transmission process, when the aggregator obtains $\sum \mathbf{P_i A_i x} = \mathbf{b}$. We use the field statistic results measured in [44] as the communication latencies between the aggregator and the Mercator domain servers at the different member networks. Second, the data transmission overhead is measured as the size of the set of encrypted, obfuscated linear equations transferred from each member network to the aggregator. We vary the number of member networks from 10 to 200, in a step size of 10. For each number of member networks, we repeat the experiment 10 times and measure the average values of these metrics.

2) Results: We present the results of our experiments in Fig. 11 and Fig. 12. In particular, Fig. 11a shows the overall latency of the obfuscating protocol under different numbers of member networks, together with a break down on processing delay and transmission latency. We observe that even for a large collaboration network with 200 member networks, which is larger than most existing operational collaboration networks, the overall latency of the resource abstraction obfuscating protocol is only slightly over 3 seconds, which demonstrates that the latency of this protocol is reasonably low. We also observe that the processing latency takes a much higher percentage than the transmission latency and that the processing latency has a linear growth as the number of member networks increases. We further plot the breakdown of the processing latency. Fig. 11b shows that both the cryptography operations of AES and the matrix operations in the resource abstraction obfuscating protocol increases linearly as the number of member networks increases, but the AES encryption and decryption operations are the most expensive operations in the protocol (*i.e.*, up to 2.4 seconds for federations of 200 member networks). More importantly, although the obfuscating protocol may take over 3 seconds for a federation of 200 member networks, we emphasize that with the super-set projection technique, the Mercator domain servers do not need to execute the obfuscating protocol for each individual request.

Next, we present the average data transmission overhead of the obfuscating protocol at each member network in Fig. 12. We see in this figure that even after the encryption, the size of data to be transmitted from member networks to the aggregator is still very small. For example, for a collaboration network with 200 member networks, the average size of data transmitted from a member network to the aggregator is only 180 KB. As discussed in Appendix B, this is because most of the columns of the LHS coefficient matrix are zero-columns and each member network only needs to send nonzero-

columns to the aggregator. The linear scaling of the data transmission overhead (*i.e.*, the ciphertext) at each member network comes from the linear increase of the number of disguised linear equations (*i.e.*, the plaintext), which is caused by the linear increase of k due to the increased number of member networks. This is consistent with Proposition 3 in Appendix B.

D. Efficiency of Super-Set Projection

In this experiment, we evaluate the efficiency of the super-set projection technique in improving the scalability of Mercator. We show that this mechanism improves the resource discovery delay of Mercator by 2 times, and that its update latency is within seconds in a collaborative network with 200 member networks.

1) Methodology: We conduct our experiments by using the same settings as in Section VI-B1. We focus on two metrics. The first one is the resource discovery latency. When Mercator uses super-set projection, the resource discovery latency is reduced to only the round-trip time from the user to the Mercator aggregator because the aggregator can derive the resource abstraction for a request from the precomputed Π_{full}.

To have a comprehensive understanding on the scalability of super-set projection, we are also interested in a second metric, the update latency. This is measured as the resource discovery latency of from the time the aggregator starting the artificial resource abstraction discovery procedure to the time the aggregator receives the latest Π_{full}. In particular, we measure this latency under different collaboration scales by varying the number of member networks and the number of stub member networks in the collaborative network. For each setting, we repeat the experiment 10 times and compute the average update latency. In each repetition, we also randomly choose different sizes of intradomain topologies from the Topology Zoo dataset for each member network.

2) Results: Fig. 13 compares the resource discovery latency of Mercator with and without super-set projection. We observe that the super-set projection technique decreases the average resource discovery latency by around 2 times. Fig. 14 presents the update latency of this mechanism. It shows that even in a collaborative network with 200 member networks, the update latency of Π_{full} is still less than 10 seconds. Most importantly, although computing Π_{full} may take up to ten seconds for a federation of 200 member networks, we emphasize that resource discovery requests do not get blocked at the aggregator because servers from the aggregator pool can still process incoming requests using the previously computed resource abstraction, which is continuously locally updated (*e.g.*, available resources are continuously reduced as incoming requests reserve resources).

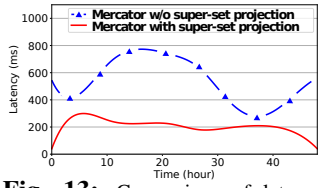


Fig. 13: Comparison of latency between Mercator with and without super-set projection.

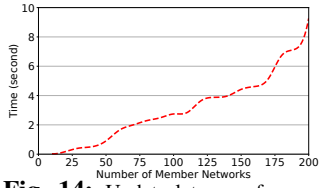


Fig. 14: Update latency of super-set projection.

VII. Related Work

Many multi-domain network resource information and reservation systems [4], [5], [8]–[10], [46] have been developed to support collaborative data sciences. Multiple multi-domain resource discovery systems (*e.g.*, [11]–[17]) are also designed to discover endpoint resources (*i.e.*, computation and storage resources) and their availability for different services across multiple domains. In contrast, there has been little progress on multi-domain network resource discovery systems that provide fine-grained, global network resource information, to support high-performance, collaborative data sciences.

Many cluster / grid resource management systems [15], [17], [27], [28], [47]–[53] adopt a graph-based abstraction to discover and manage network resources. However, in a multi-domain collaborative network, this abstraction would reveal the network topology and link availabilities of member networks, leading to security breaches. Some systems [29], [30] use a one-big-switch abstraction to provide a simplified view of network resources, which protects the privacy of member networks but cannot provide accurate information of shared network resource for concurrent traffic flows. Some recent studies [26], [33], [34], [54], [55] propose variations of the one-big-switch abstraction to represent the resource availability and sharing among different data traffic flows using operations defined on different algebra fields. However, this abstraction (1) cannot handle complex routing and traffic engineering policies, *e.g.*, WCMP, and (2) will raise security concern when applied to multi-domain science collaborations. In contrast, Mercator provides fine-grained, global network resource information, to support high-performance, collaborative data sciences, through a unifying representation and composition framework to reveal compact, complete multi-domain network resource information.

VIII. Conclusion

We present Mercator, a novel multi-domain network resource discovery system to provide fine-grained, global network resource information, to support high-performance, collaborative data sciences. The core of Mercator is a unifying representation resource abstraction using algebraic expressions to capture multi-domain network available bandwidth. We develop a resource abstraction obfuscating protocol and a super-set projection technique to ensure the privacy-preserving and the scalability of Mercator. Evaluation using real data demonstrates the efficiency and efficacy of Mercator.

Acknowledgment

The authors thank Kai Gao, Geng Li, Linghe Kong, Ennan Zhai, Alan Liu, Yeon-sup Lim and Haizhou Du for their help during the preparation of this paper. The authors also thank the anonymous reviewers for their valuable comments. This research is supported in part by NSFC grants #61702373, #61672385 and #61701347; China Postdoctoral Science Foundation #2017-M611618; NSF awards #1440745, #1246133, #1341024, #1120138, and #1659403; DOE award #DE-AC02-07CH11359; DOE/ASCR project #000219898; Google Research Award, and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

[1] "The Large Hadron Collider (LHC) Experiment," https://home.cern/topics/large-hadron-collider.
[2] "The Square Kilometre Array," https://www.skatelescope.org/.
[3] "The Linac Coherent Light Source," https://lcls.slac.stanford.edu/.
[4] "Oscars: On-demand secure circuits and advance reservation system," https://www.es.net/engineering-services/oscars/.
[5] M. Campanella, R. Krzywania, V. Reijs, D. Wilson, A. Sevasti, K. Stamos, and C. Tziouvaras, "Bandwidth on demand services for european research and education networks," in Bandwidth on Demand, 2006 1st IEEE International Workshop on. IEEE, 2006, pp. 65–72.
[6] C. Guok, E. N. Engineer, and D. Robertson, "Esnet on-demand secure circuits and advance reservation system (oscars)," Internet2 Joint, vol. 92, 2006.
[7] W. Johnston, C. Guok, and E. Chaniotakis, "Motivation, design, deployment and evolution of a guaranteed bandwidth network service," in Proceedings of the TERENA Networking Conference, 2011.
[8] B. Riddle, "Bruw: A bandwidth reservation system to support end-user work," in TERENA Networking Conference, Poznan, Poland, 2005.
[9] J. Sobieski, T. Lehman, and B. Jabbari, "Dragon: Dynamic resource allocation via gmpls optical networks," in MCNC Optical Control Planes Workshop, Chicago, Illinois, 2004.
[10] X. Zheng, M. Veeraraghavan, N. S. Rao, Q. Wu, and M. Zhu, "Cheetah: Circuit-switched high-speed end-to-end transport architecture testbed," IEEE Communications Magazine, vol. 43, no. 8, pp. S11–S17, 2005.
[11] Y. Deng, F. Wang, and A. Ciura, "Ant colony optimization inspired resource discovery in p2p grid systems," The Journal of Supercomputing, vol. 49, no. 1, pp. 4–21, 2009.
[12] S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith, and S. Tuecke, "A directory service for configuring high-performance distributed computations," in IEEE HPDC 1997.
[13] A. Iamnitchi and I. Foster, "A peer-to-peer approach to resource location in grid environments," in Grid resource management. Springer, 2004, pp. 413–429.
[14] T. Kocak and D. Lacks, "Design and analysis of a distributed grid resource discovery protocol," Cluster Computing, vol. 15, no. 1, pp. 37–52, 2012.
[15] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, "The pilot way to grid resources using glideinWMS," in CSIE. IEEE, 2009, pp. 428–432.
[16] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Transactions on Networking (TON), vol. 11, no. 1, pp. 17–32, 2003.
[17] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," Concurrency and computation: practice and experience, vol. 17, no. 2-4, pp. 323–356, 2005.
[18] R. Ahmed, N. Limam, J. Xiao, Y. Iraqi, and R. Boutaba, "Resource and service discovery in large-scale multi-domain networks," IEEE Communications Surveys & Tutorials, vol. 9, no. 4, pp. 2–30, 2007.
[19] A. Hameurlain, D. Cokuslu, and K. Erciyes, "Resource discovery in grid systems: a survey," International Journal of Metadata, Semantics and Ontologies, vol. 5, no. 3, pp. 251–263, 2010.
[20] N. J. Navimipour, A. M. Rahmani, A. H. Navin, and M. Hosseinzadeh, "Resource discovery mechanisms in grid systems: A survey," Journal of Network and Computer Applications, vol. 41, pp. 389–410, 2014.
[21] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, "Internet x. 509 public key infrastructure (pki) proxy certificate profile," Tech. Rep., 2004.
[22] N. Sakimura, J. Bradley, M. Jones, and B. de Medeiros, "C. mortimore," openid connect core 1.0", november 2014."
[23] O. S. S. T. Committee et al., "Security assertion markup language (saml) 2.0," ht tp://www. oasis-open. org/committees/tc home. php, 2012.
[24] Y. Rekhter, S. Hares, and D. T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4271.txt
[25] "Route views project," http://www.routeviews.org/routeviews/.
[26] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using sol." in NSDI, 2016, pp. 223–237.
[27] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in NSDI, 2011.
[28] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in EuroSys. ACM, 2015, p. 18.
[29] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4p:provider portal for applications," Acm Sigcomm Aug, vol. 38, no. 4, pp. 351–362, 2008.
[30] R. Alimi, Y. Yang, and R. Penno, "Application-layer traffic optimization (ALTO) protocol."
[31] J. Y. Boudec, "Rate adaptation, congestion control and fairness: A tutorial," Web Page, no. Oct, 2000.
[32] F. P. Miller, A. F. Vandome, and J. McBrewster, "Advanced encryption standard," 2009.
[33] K. Gao, C. Gu, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "ORSAP: abstracting routing state on demand," in IEEE ICNP 2016.
[34] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "Nova: Towards on-demand equivalent network view abstraction for network optimization," in ACM/IEEE IWQoS 2017, 2017.
[35] J. Telgen, "Identifying redundant constraints and implicit equalities in systems of linear constraints," Management Science, vol. 29, no. 10, pp. 1209–1222, 1983.
[36] M. Raykova, Secure Computation in Heterogeneous Environments: How to Bring Multiparty Computation Closer to Practice? Columbia University, 2012.
[37] A. C.-C. Yao, "How to generate and exchange secrets," in IEEE FOCS 1986.
[38] X. Feng and Z. Zhang, "The rank of a random matrix," Applied mathematics and computation, vol. 185, no. 1, pp. 689–694, 2007.
[39] O. L. Mangasarian, "Privacy-preserving horizontally partitioned linear programs," Optimization Letters, vol. 6, no. 3, pp. 431–436, 2012.
[40] "The CAIDA AS Relationships Dataset, 2016," http://www.caida.org/data/as-relationships/.
[41] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," vol. 29, no. 9, pp. 1765–1775.
[42] "CMS Task Monitoring," http://dashb-cms-job.cern.ch/.
[43] R. Jain, D.-M. Chiu, and W. R. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984, vol. 38.
[44] "Global Ping Statistics - WonderNetwork, 2018," https://wondernetwork.com/pings/.
[45] "Python Cryptography Toolkit," https://pypi.python.org/pypi/pycrypto.
[46] "Network service interface," https://redmine.ogf.org/projects/nsi-wg.
[47] "Under the hood: Scheduling MapReduce jobs more efficiently with Corona," http://on.fb.me/TxUsYN, [Online; accessed: 09-May-2017].
[48] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in OSDI, 2014, pp. 285–300.
[49] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in SoCC. ACM, 2015, pp. 111–124.
[50] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy:fair scheduling for distributed computing clusters," in IEEE International Conference on Recent Trends in Information Systems, 2009, pp. 261–276.
[51] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-distributed Data Analytics," in SIGCOMM. ACM, 2015, pp. 421–434.
[52] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in Usenix Conference on Operating Systems Design and Implementation, 2016, pp. 435–450.
[53] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "WANalytics: Analytics for a geo-distributed data-intensive world," in CIDR, 2015.
[54] Q. Xiang, S. Chen, K. Gao, H. Newman, I. Taylor, J. Zhang, and Y. R. Yang, "Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics," in 2017 IEEE SmartWorld, DAIS Workshop.
[55] Q. Xiang, X. Wang, J. Zhang, H. Newman, Y. R. Yang, and Y. J. Liu, "Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics," in INDIS Workshop. IEEE, 2017.
[56] Q. Xiang, J. J. Zhang, X. T. Wang, Y. J. Liu, C. Guok, F. Le, J. MacAuley, H. Newman, and Y. R. Yang, "Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences," in Technical Report.

APPENDIX A
USE CASES

In this appendix, we show how resource abstraction handles three important use cases in collaborative data sciences.

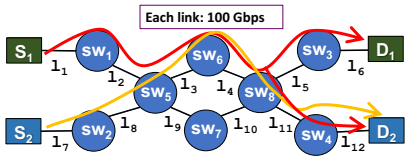


Fig. 15: A running example illustrating how the resource abstraction handles multicast through algebraic-expression enumeration, where two circuits $(S_1, \{D_1, D_2\})$ and (S_2, D_2) need to be reserved.

Use case 1 - multicast: Consider the example in Fig. 15, where the first circuit is a multicast circuit from S_1 to D_1 and D_2, and the second one is a unicast circuit from S_2 to D_2. The routes for these circuits, computed by the underlying routing protocol, are marked in red and yellow, respectively. The resource abstraction captures the bandwidth sharing between these two circuits by introducing auxiliary variables x_{11} and x_{12} for the multicast circuit. Because the traffic duplication for the first circuit happens at switch 8, we use x_{11} to represent the traffic from switch 8 to D_1, and x_{12} to represent the traffic from switch 8 to D_2. In this way, the Mercator domain server will generate the following set of linear inequalities:

$$\begin{array}{ll} x_{11} = x_1,\ x_{12} = x_1, & \\ x_1 \le 100 & \forall l_u \in \{l_1, l_2\}, \\ x_{11} \le 100 & \forall l_u \in \{l_5, l_6\}, \\ x_2 \le 100 & \forall l_u \in \{l_7, l_8\}, \\ x_1 + x_2 \le 100 & \forall l_u \in \{l_3, l_4\}, \\ x_{12} + x_2 \le 100 & \forall l_u \in \{l_{11}, l_{12}\}, \end{array} \tag{8}$$

Use case 2 - multi-path routing: Consider the example in Fig. 16, where the user wants to discover the bandwidth sharing for two circuits $f_1 : (S_1, D_1)$ and $f_2 : (S_2, D_2)$, and $\mathbb{M}_1$ uses multi-path routing for the circuit f_1, *i.e.*, routing to two egresses e_1, e_2.

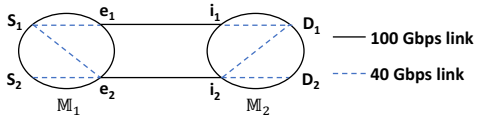


Fig. 16: A running example illustrating how resource abstraction handles complex routing and traffic engineering policies through algebraic-expression enumeration and how resource abstractions from different member networks are stitched, where two circuits (S_1, D_1) and (S_2, D_2) need to be reserved.

In particular, the Mercator domain server at $\mathbb{M}_1$ introduces variables x_{11} and x_{12} to represent the available bandwidth from S to egresses e_1 and e_2, respectively, and share the introduction of these variables to $\mathbb{M}_2$. Then $\mathbb{M}_1$ independently adds an equation $x_1 = x_{11} + x_{12}$ into its set of linear inequalities $\Pi_1(F)$. The resulting resource abstraction at both member networks are then expressed as

$$\begin{array}{lll} \Pi_1(F): & x_1 = x_{11} + x_{12}, & \Pi_2(F): \quad x_{11} \le 40, \\ & x_{11} \le 40, & \qquad\quad x_{12} \le 40, \\ & x_{12} \le 40, & \qquad\quad x_2 \le 40. \\ & x_2 \le 40, & \\ & x_{11} \le 100, & \\ & x_{12} + x_2 \le 100. & \end{array} \tag{9}$$

Using $\Pi_1(F)$ and $\Pi_2(F)$ as the constraint, the user can then make reservation requests based on the optimization of her own objective function. For example, to achieve the max-min fairness between two circuits, the user will reserve $x_1 = 80$ Gbps for (S_1, D_1) and $x_2 = 40$ Gbps for (S_2, D_2), where internally $\mathbb{M}_1$ can allocate $x_{11} = x_{12} = 40$ Gbps.

Use case 3 - load-balancing: In the same example in Fig. 16, assume $\mathbb{M}_1$ uses weighted-cost-multi-path (WCMP) and has an internal policy to allocate bandwidth for the circuit (S_1, D_1) along two path $S_1 \to e_1$ and $S_1 \to e_2$ in a ratio of 1:2. With this policy, the previous reservation request with $x_1 = 80$ Gbps and $x_2 = 40$ Gbps is no longer valid as x_{11} and x_{12} cannot reach 40 Gbps simultaneously. To capture this policy so that the user does not make the invalid reservation request, the Mercator domain server at $\mathbb{M}_1$ introduces an additional equation $x_{12} = 2x_{11}$ into $\Pi_1(F)$ and sends to the user. And the user can compute the valid, optimal reservation decisions, *e.g.*, $x_1 = 60$ Gbps and $x_2 = 40$ Gbps, to achieve max-min fairness.

APPENDIX B
ANALYSIS OF THE RESOURCE ABSTRACTION OBFUSCATING PROTOCOL

In this appendix, we conduct rigorous analysis on different properties of the proposed obfuscating protocol.

Correctness: We first study the correctness of this protocol. In particular, we start by proposing and proving the following propositions.

***Proposition** 1 (Resource Abstraction Equivalence):* The bandwidth feasible region of the set of circuits F over N member networks represented by Equation (7) is the same as the bandwidth feasible region represented by $\mathbf{Ax} \le \mathbf{b}$ where $\mathbf{A} = [\mathbf{A_1}, \mathbf{A_2}, \ldots, \mathbf{A_N}]$ and $\mathbf{b} = [\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_N}]$.

Proof: To prove this proposition, we first observe that the bandwidth feasible region of $\mathbf{Ax} \le \mathbf{b}$ is the same as that of

$$\begin{bmatrix} \mathbf{A} & \mathbf{I_{M_N}} \end{bmatrix} \begin{bmatrix} \mathbf{x}, & \mathbf{x^s} \end{bmatrix} = \mathbf{b} \tag{10}$$

Representing $\mathbf{P} = [\mathbf{P_1}, \ldots, \mathbf{P_N}] \in R^{k \times M_N}$, we first observe that $\begin{bmatrix} \sum \mathbf{P_i A_i} & \mathbf{P_1} & \ldots & \mathbf{P_N} \end{bmatrix} = \mathbf{P} \begin{bmatrix} \mathbf{A} & \mathbf{I_{M_N}} \end{bmatrix}$, and that $\sum \mathbf{P_i b_i} = \mathbf{Pb}$ [39]. It is easy to see that when $\begin{bmatrix} \mathbf{x} & \mathbf{x^s} \end{bmatrix}$ satisfies Equation (10), it also satisfies Equation (7).

Next, from the results in [38] and that $\mathbf{P} \in R^{k \times M_N}$, we have $rank(\mathbf{P}) = M_N < k$. As a result, $\mathbf{P}$ has a left inverse matrix $\mathbf{P}^{-1}_{left}$ where $\mathbf{P}^{-1}_{left}\mathbf{P} = \mathbf{I}_{M_N}$. Hence when $\begin{bmatrix} \mathbf{x} & \mathbf{x^s} \end{bmatrix}$ satisfies Equation (7), *i.e.*, $\mathbf{P} \begin{bmatrix} \mathbf{A} & \mathbf{I_{M_N}} \end{bmatrix} \begin{bmatrix} \mathbf{x}, & \mathbf{x^s} \end{bmatrix} = \mathbf{Pb}$, we have

$$\mathbf{P}^{-1}_{left}\mathbf{P} \begin{bmatrix} \mathbf{A} & \mathbf{I_{M_N}} \end{bmatrix} \begin{bmatrix} \mathbf{x}, & \mathbf{x^s} \end{bmatrix} = \mathbf{P}^{-1}_{left}\mathbf{Pb},$$

which then transforms into Equation (10). Therefore, Equations (7) and (10) represent the same bandwidth feasible region, which completes the proof. ■

In addition, we also have an interesting lemma, which serves as an alternative correctness proof of the equivalence proposition.

***Lemma** 1:* Given the set of linear equations in Equation (7), the aggregator can use Gaussian Elimination to get $\mathbf{Ax} \le \mathbf{b}$. The complete proof can be found in our technical report [56].

Security: Next, we give the following proposition on the privacy-preserving property of the proposed protocol.

***Proposition** 2 (Resource Abstraction Privacy-Preserving):* In the semi-honest security model, the proposed resource abstraction obfuscating protocol ensures that (1) the aggregator cannot associate any linear equation it receives in $\Pi_p(F)$ with any particular member network, and (2) for any $\mathbb{M}_i$, it does not know any linear inequality from any other $\Pi_j(F_j)$ $(j \neq i)$.

The complete proof can be found in [56]. Even with Lemma 1 and the inter-member-network-path information of each circuit, the aggregator still cannot associate any linear inequality in $\mathbf{Ax} \leq \mathbf{b}$ with the corresponding member network or any networking device (*i.e.*, switch or link). This is because (1) the set of linear equations sent by each member network do not represent its original feasible region, and (2) the inter-member-network-path does not reveal any topology information inside member networks.

With both propositions, we can get the following theorem.

***Theorem** 1:* Given a set of circuits F that span over N member networks, the proposed resource abstraction obfuscating protocol ensures that the aggregator receives equivalent, privacy-preserving resource abstraction and each member network only knows its own bandwidth feasible region.

As stated in Section IV-A, the resource abstraction obfuscating protocol was designed for the semi-honest security model. In a malicious setting (*e.g.*, some member networks may collude or be breached by one same attacker), the colluded member networks or the attacker still cannot associate a linear inequality to any unbreached member network, as long as the aggregator is not breached.

Efficiency: We next analyze the efficiency of our protocol at different phases. During the initialization phase, the main overhead comes from the process each member network agreeing on k, and each $\mathbb{M}_i$ share $\mathbf{C_i}$ and $\mathbf{D_i}$ with $\mathbb{M}_{i+1}$. The first part can be efficiently realized using leader-election algorithms in ring topology or pre-configured. For the second part, it can be efficiently realized by sharing random seeds between $\mathbb{M}_i$ and $\mathbb{M}_{i+1}$. In the obfuscating phase, the computation overhead is also low because it only involves simple, cheap matrix operations, *e.g.*, addition and multiplication.

One may have concern on the transmission overhead of our protocol in the transmission phase because we disguise the set of linear inequalities of each member network into a larger set of linear equations. However, observing the set of equations sent by each $\mathbb{M}_i$ in Equation (6), we can see that most of the columns of the LHS coefficient matrix are zero-columns. Therefore, each $\mathbb{M}_i$ only needs to send nonzero-columns to the aggregator and specifies the indice of these columns, substantially reducing the amount of data needs to be transmitted from $\mathbb{M}_i$ to the aggregator. We quantify the transmission overhead of our obfuscating protocol as follows:

***Proposition** 3 (Transmission Overhead):* Given a resource discovery procedure for a set of circuits F spanning over N member networks, the transmission overhead of the resource abstraction obfuscating protocol at each member network is $O(k|F|)$, where $k > \sum m_i$.

APPENDIX C
PRACTICAL ISSUES OF SUPER-SET RESOURCE ABSTRACTION PROJECTION

In this appendix, we discuss practical issues of the super-set projection technique.

Update of Π_{full}: We ensure the freshness of Π_{full} via two mechanisms. First, the Mercator domain servers at member networks periodically send updated information to the aggregator. Second, when the reservation system receives and successfully executes a resource reservation request from the user, it sends a notification to the aggregator with the reservation details so that the aggregator can update Π_{full}. The aggregator will only query the Mercator domain servers to obtain an up-to-date abstraction for the user when the user fails to reserve the resource based on the projected abstraction.

Handling heterogeneous flows: One may notice that the super-set projection technique is designed based on the assumption that given a source-destination member network pair, all the traffic flows between these two member networks will be treated homogeneously by all other member networks. In practice, flows between the same source-destination member network pair may be handled differently by other member networks, *i.e.*, they are heterogeneous flows. To address this limitation, we use traffic classes to differentiate heterogeneous flows. In particular, for each source-destination member network pair with G different traffic classes, the super-set projection technique considers these classes as G separate artificial circuits and proactively discovers the bandwidth sharing among these G circuits and other artificial circuits.

SFP: Toward Interdomain Routing for SDN Networks

Qiao Xiang, Tongji/Yale
Chin Guok, LBNL
Franck Le, IBM
John MacAuley, LBNL
Harvey Newman, Caltech
Y. Richard Yang, Tongji/Yale

ABSTRACT

Interdomain routing using BGP is widely deployed and well understood. The deployment of SDN in BGP domain networks, however, has not been systematically studied. In this paper, wefi rst show that the *use-announcement inconsistency* is a fundamental mismatch in such a deployment, leading to serious issues including unnecessary blackholes, unnecessary reduced reachability, and permanent forwarding loops. We then design SFP, thefi rstfi ne-grained interdomain routing protocol that extends BGP withfi ne-grained routing, eliminating the aforementioned mismatch. We develop two novel techniques, automatic receiverfi ltering and on-demand information dissemination, to address the scalability issue brought byfi ne-grained routing. Evaluating SFP using real network topologies and traces for intended settings, which are not global Internet but tens of collaborative domains, we show that SFP can reduce the amount of traffic affected by blackholes and loops by more than 50%, and that our proposed techniques can reduce the amount of signaling between ASes by 3 orders of magnitude compared with naive fine-grained routing.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Routing protocols**; **Programmable networks**;

KEYWORDS

Interdomain Routing, SDN, On-Demand Information Dissemination

ACM Reference Format:
Qiao Xiang, Chin Guok, Franck Le, John MacAuley, Harvey Newman, and Y. Richard Yang. 2018. SFP: Toward Interdomain Routing for SDN Networks. In *SIGCOMM Posters and Demos '18: ACM SIGCOMM 2018 Conference Posters and Demos, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234200.3234207

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

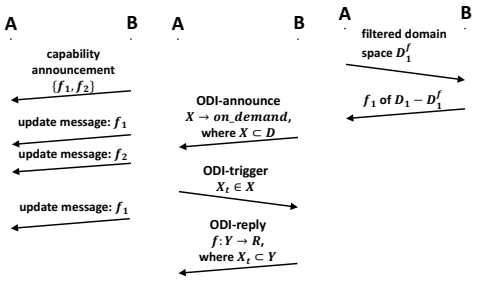
ACM ISBN 978-1-4503-5915-3/18/08…$15.00
https://doi.org/10.1145/3234200.3234207

1 INTRODUCTION

There are multiple important settings where multiple anonymous systems (ASes) interconnect to form collaborative networks (also called federations) to improve network performance of large scientific collaboration across member networks [2]. The *de facto* protocol to interconnect these ASes is the Border Gateway Protocol (BGP) [6]. Meanwhile, to support efficient usage of their network resources, members of federation networks commonly deploy software defined networking (SDN) [1, 3]. Though interdomain routing using BGP is well understood, the deployment of SDN in BGP-connected networks has not been systematically studied.

Specifically, we show that such a deployment reveals a fundamental mismatch between thefi ne-grained control by SDN and the coarse-grained routing by BGP, *i.e.*, the *use-announcement inconsistency*, which leads to serious issues. To illustrate these issues, consider two networks A, and B, connected using BGP. We focus on a prefix P. Assume that B drops all traffic sent to P with TCP destination port 22. If B still announces, through BGP, routes to P to its neighbor A, a subset of traffic (more specifically, traffic with destination port 22) may result in blackholes. Instead, if B does not announce any route to P to its neighbor A, such policy can result in reduced reachability for traffic to P with destination port other than 22. To further illustrate the issues, we assume that instead of dropping traffic with destination port 22, B decides to redirect this traffic to another network C and still announce the reachability of P to A via BGP. Such behavior can lead to permanent forwarding loops. Several solutions have been proposed to providefi ne-grained routing. A particularly elegant one is SDX [5]. An issue of SDX, however, is that it requires a trusted third party to conduct integration.

Different from existing solution approaches, this paper investigates a simple, novel protocol named SFP (SDN Federation Protocol) that maintains the compatibility with BGP and at the same time provides *fine-grained routing*, where each network decides interdomain routes based on common packet headerfi elds instead of destination IP only. We prove that SFP avoids all issues caused by use-announcement inconsistency. We develop two novel techniques, on-demand routing information dissemination and automatic receiver filtering, to address the messaging scalability issue brought byfi ne-grained routing. Evaluating SFP using real network topologies and traces, we show that by guaranteeing blackhole / loop-free routing, SFP can reduce the traffic affected



(a) Functional mapping dissemination. **(b)** On-demand information dissemination. **(c)** Automatic receiverfiltering.

Figure 1: The message-time diagrams of SFP protocol.

by blackholes and forwarding loops by more than 50%, and the amount of signaling between ASes is reduced by 3 orders of magnitude compared with naivefi ne-grained routing.

2 SFP HIGH-LEVEL PROTOCOL DESIGN

Functional mapping information dissemination: SFP uses a functional mapping from the *general packet space* domain space – where each point is defined as a vector of common packet headerfi elds (*e.g.*, the TCP/IP 5-tuple) and the address of an ingress, representing a packet entering the ingress of a network – to the *AS-path space*, where each point represents an AS-path, as the representation of the routing information base (RIB) of each network.

Property 1. *SFP with general packet space announcements has the same space partition structure as SDN, and hence can avoid unnecessary blackholes, reachability, or loops caused by use-announcement inconsistency.*

A major concern offi ne-grained routing is its messaging scalability, as the RIB can become extremely large due to the multiple packet headerfi elds, and the ensuing cross-product when composingfi ne-grained routes. To achieve scalability, we develop two novel techniques in SFP.

On-demand information dissemination (ODI): This technique allows a neighbor AS B to send incomplete functional mappings to AS A, and later send updated mappings to A based on A's demand triggers. In Figure 1b), B first sends an *ODI-announce* message $X \rightarrow on\_demand$ to A to indicate that the mapping of the subspace X is on-demand to A. When A needs the missing information, it sends an on-demand trigger X_t to B, where $X_t \subset X$, to ask for the information of the domain subspace X_t. Upon receiving the trigger, B looks in its local RIB for the mapping of X_t, invokes its SDN program to compute the mapping if it is not in RIB, sends other triggers to query other ASes if its SDN program does not provide the mapping, and returns an *ODI-reply* message $Y \rightarrow R$, the mapping information of a subspace $Y \supset X_t$. We prove that ODI does not introduce any new convergence issue. Given a single point in the general packet space, under the Gao-Rexford conditions [4], ODI always converges.

Automatic receiverfiltering : Consider a packet pkt. If A can already make the decision for pkt without the information from a neighbor B, then the mapping sent from B to

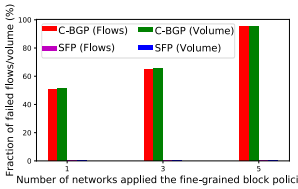
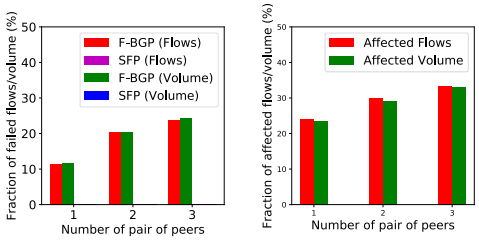


Figure 2: Loss when blocking one unused destination TCP port.



(a) Failed traffic **(b)** Traffic affected by loops

Figure 3: Loss when deflect traffic between neighboring peers.

A does not need to include pkt. With this observation, SFP allows the receiver AS A to provide *receiverfiltering* to notify the neighbor B which part of a mapping is of no use to A. In Figure 1c, A sends $\mathcal{D}_1'$, afi lter on the domain space $\mathcal{D}_1$ of mapping f_1, to B. When B sends f_1 to A, it only needs to send the mapping from a smaller space $\mathcal{D}_1 - \mathcal{D}_1'$, reducing the amount of exchanged routing information.

3 PERFORMANCE EVALUATION

We evaluate SFP on the topology of LHCONE using real trace from the CMS experiment [2], a main source of traffic in LHCONE. We compare SFP with two variations of BGP: C-BGP, where a route to a destination IP will not be announced by an AS if the AS only uses this route for a subset of traffic for this IP, and F-BGP, where such a route will still be announced. Figure 2 shows that with 1 transit network deploying one fine-grained policy blocking traffic to one unused destination transport port, 50% of thefl ows, and 51% of the traffic volume can be dropped in C-BGP. In contrast, SFP ensures all 100% offl ows to successfully reach their destinations. Figure 3a shows that with 1 pair of providers deploying one fine-grained policy to deflect large data transfers sent to their customer, 11% of thefl ows result in loops in F-BGP. In addition,fl ows traversing the affected links – although not resulting in loops – may still suffer high packet losses. Figure 3b show that 23% of thefl ows can get affected in F-BGP, whereas nofl ow is affected in SFP.

ACKNOWLEDGMENTS

The authors thank Jingxuan Zhang, Xin Wang and Yang Liu (Tongji University) for their help during the preparation of the paper. This research is supported in part by NSFC grants #61672385, #61472213 and #61702373; China Postdoctoral Science Foundation #2017-M611618; NSF grant CC-IIE #1440745; NSF award #1246133, ANSE; NSF award #1341024, CHOPIN; NSF award #1120138, US CMS Tier2; NSF award #1659403, SANDIE; DOE award #DE-AC02-07CH11359, SENSE; DOE/ASCR project #000219898; Google Research Award, and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

[1] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 87–95.

[2] Collaboration, T. C. The CMS experiment at the CERN LHC. *Journal of Instrumentation 3*, 08 (2008).

[3] Foundation, O. N. Openflow switch specification 1.4.0. Open Networking Foundation (on-line), Oct. 2013.

[4] Gao, L., andRexford , J. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking (TON)* (2001).

[5] Gupta, A., Vanbever, L., Shahbaz, M., Schlinker, S. P. D. B., Feamster, N., Rexford, J., Shenker, S., Clark, R., andK atz-Bassett, E. SDX: A Software Defined Internet Exchange. In *SIGCOMM'14*.

[6] Rekhter, Y., Hares, S., andLi , D. T. A Border Gateway Protocol 4 (BGP-4). RFC 4271, Jan. 2006.

Fine-Grained, Multi-Domain Network Resource Abstraction as a Fundamental Primitive to Enable High-Performance, Collaborative Data Sciences

Qiao Xiang
Tongji/Yale

J. Jensen Zhang
Tongji

X. Tony Wang
Tongji

Y. Jace Liu
Tongji

Chin Guok
LBNL

Franck Le
IBM

John MacAuley
LBNL

Harvey Newman
Caltech

Y. Richard Yang
Tongji/Yale

ABSTRACT

Recently, a number of multi-domain network resource information and reservation systems have been developed and deployed, driven by the demand and substantial benefits of providing predictable network resources. A major lacking of such systems, however, is that they are based on coarse-grained or localized information, resulting in substantial inefficiencies. In this paper, we present Explorer, a simple, novel, highly efficient multi-domain network resource discovery system to provide fine-grained, global network resource information, to support high-performance, collaborative data sciences. The core component of Explorer is the use of linear inequalities, referred to as resource state abstraction (ReSA), as a compact, unifying representation of multi-domain network available bandwidth, which simplifies applications without exposing network details. We develop a ReSA obfuscating protocol and a proactive full-mesh ReSA discovery mechanism to ensure the privacy-preserving and scalability of Explorer. We fully implement Explorer and demonstrate its efficiency and efficacy through extensive experiments using real network topologies and traces.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Signaling protocols**; **Network privacy and anonymity**; *Network resources allocation*;

KEYWORDS

Interdomain, Collaborative Data Sciences, Resource Discovery, Resource State Abstraction

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5915-3/18/08…$15.00
https://doi.org/10.1145/3234200.3234208

ACM Reference Format:
Qiao Xiang, J. Jensen Zhang, X. Tony Wang, Y. Jace Liu, Chin Guok, Franck Le, John MacAuley, Harvey Newman, and Y. Richard Yang. 2018. Fine-Grained, Multi-Domain Network Resource Abstraction as a Fundamental Primitive to Enable High-Performance, Collaborative Data Sciences. In *SIGCOMM Posters and Demos '18: ACM SIGCOMM 2018 Conference Posters and Demos, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234200.3234208

1 INTRODUCTION

Many emerging large-scale data science projects (*e.g.*, the Large Hadron Collider experiment [1]), are based on a collaborative, distributed-networks design, where massive datasets have to be moved from storage facilities to large computing clusters distributed at multiple autonomous member networks, and analyzed by multi-stage distributed systems. Such large, correlated and parallel flows (*e.g.*, petabytes of data) have evolved to dominate science networks' traffic. To ensure the completion of those transfers within the application time constraints, users require the ability to reserve and guarantee bandwidth across networks. As such, a number of on-demand circuits reservation systems have been developed and deployed (*e.g.*, the OSCARS system [2] in LHC).

However, such existing systems are based on localized design and hence can suffer poor performance for correlated and concurrent flows across multiple ASes. For privacy reasons, existing multi-domain reservation systems treat each AS as a black box. They probe their available resource by submitting varied circuit reservation requests, and receive Boolean responses. In other words, current solutions perform a depth-first search on all ASes, and rely on a trial and error approach: to reserve bandwidth, repeated, and varied attempts may have to be submitted until success. In addition to requiring a large number of search attempts, this approach may obtain a bandwidth allocation that is far from optimal (*e.g.*, max-min fairness).

This paper presents Explorer, a system designed to optimize large, multi-domain transfers, and address the limitations of current reservation systems through three main components. The first and core component of Explorer is

the use of linear inequalities, referred to as *resource state abstraction* (ReSA), as a compact, unifying representation of multi-domain network available bandwidth. Second, Explorer introduces a ReSA obfuscating protocol to ensure that ASes and other external parties cannot associate an inequality with a corresponding AS. Finally, Explorer includes a proactive full-mesh ReSA discovery component for scalability purposes. This component improves the latency of resource discovery via AS-level ReSA pre-computation and projection. We implement Explorer. Extensive experiments using real network topologies and traces show that Explorer (1) efficiently discovers available networking resources in collaborative networks on average 2 orders of magnitude faster, and allows fairer allocations of network resources, (2) preserves the private information of ASes with little overhead; and (3) scales to a collaborative network with 200 ASes.

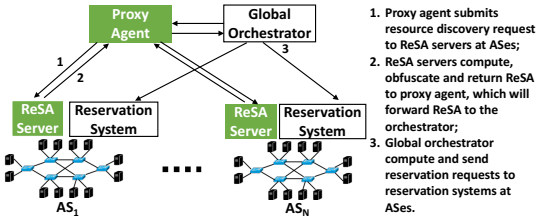


Figure 1: The architecture and workflow of Explorer.

2 EXPLORER OVERVIEW

Architecture: Explorer introduces a resource discovery proxy agent, and a ReSA server in each AS (Figure 1). The resource discovery proxy agent is the main interface for the orchestrator to submit the resource discovery requests and provides a unified view of the resources across different the ASes. The proxy agent has BGP sessions to all participating ASes, and given a request for a set of circuits, can thus infer the AS paths for each circuit in the request. It also has connections to the ReSA servers in each AS, which upon receiving requests provided from the proxy agent, compute the ASes' ReSA abstractions.

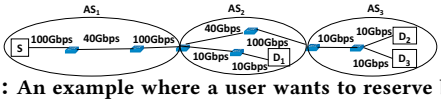
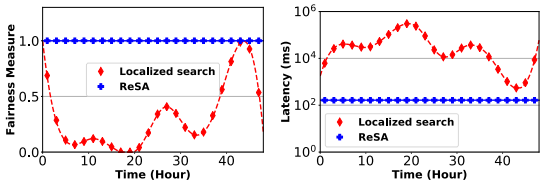


Figure 2: An example where a user wants to reserve bandwidth for three source-destination pairs: (S, D_1), (S, D_2) and (S, D_3).

Resource state abstraction (ReSA): ReSA is a unifying representation that captures the properties (e.g., available bandwidth) of resources shared – within and between ASes – by a set of requested circuits. It relies on linear inequalities to express the available bandwidth of shared resources for a set of requested circuits to be reserved, and can also support more complex traffic engineering policies (*e.g.*, multi-path routing and multicast), with the use of auxiliary variables and additional inequalities. As an example, consider a collaboration network composed of three ASes, where a user wants to reserve bandwidth for three circuits, from source host S to destination hosts D_1, D_2 and D_3 (Figure 2). The ReSA abstraction captures all constraints from all networks using linear inequalities as follows:



(a) Max-min fairness of resource utilization. **(b) Resource discovery latency.**

Figure 3: Comparison of performance between Explorer and localized search (*e.g.*, OSCARS).

$$\begin{array}{ccc} \text{AS}_1: & \text{AS}_2: & \text{AS}_3: \\ x_1 + x_2 + x_3 \le 100, & x_2 + x_3 \le 40,\ x_1 \le 10, & x_2 + x_3 \le 10, \\ x_1 + x_2 + x_3 \le 40, & x_2 + x_3 \le 100,\ x_1 \le 10, & x_2 \le 10, \\ x_1 + x_2 + x_3 \le 100, & & x_3 \le 10, \end{array}$$

where x_1, x_2 and x_3 each represents the available bandwidth that can be reserved for each circuit. For example, $x_1 + x_2 + x_3 \le 100$ means that three circuits share a common resource and the sum of their bandwidths cannot exceed 100 Gbps.

ReSA obfuscating protocol: The key idea of this protocol is to have each AS obfuscate its own set of linear inequalities as a set of linear equations through a private random matrix and a couple of random matrices shared with few other ASes, and send the obfuscated equations to the proxy agent. In this way, the proxy agent can reconstruct the original bandwidth feasible region for the circuits across the ASes, but cannot associate any linear inequality with its corresponding AS.

Proactive full-mesh ReSA discovery: The main idea of this component consists in having the proxy agent periodically query ReSA servers to discover inter-domain ReSA between every pair of source and destination ASes. As such, when a user submits a resource discovery request, the proxy agent does not need to send any query to the ReSA servers. Instead, using the AS-level ReSA information, the proxy agent can immediately perform projection operations to get the ReSA for the request. This mechanism substantially improves the scalability of Explorer.

3 PERFORMANCE EVALUATION

We implement Explorer and evaluate its performance on the topology from LHCONE, a science network with 78 ASes. We replay an actual trace from the CMS experiment [3]. We compare the performance of Explorer with that of existing resource reservation systems (*e.g.*, OSCARS), which we refer as *localized search*. Figure 3(a) shows that Explorer can always compute the optimal max-min fairness allocation, while that of the localized search based solution is 0.37 on average and even drop to 0 at times. Figure 3(b) shows that Explorer can reduce the total time to discover network resources by 2 orders of magnitude on average.

ACKNOWLEDGMENTS

This research is supported in part by NSFC grants #61672385, #61472213 and #61702373; China Postdoctoral Science Foundation #2017-M611618; NSF grant CC-IIE #1440745; NSF award #1246133, ANSE; NSF award #1341024, CHOPIN; NSF award #1120138, US CMS Tier2; NSF award #1659403, SANDIE; DOE award #DE-AC02-07CH11359, SENSE; DOE/ASCR project #000219898; Google Research Award, and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

[1] The large hadron collider. https://home.cern/topics/large-hadron-collider.
[2] Oscars: On-demand secure circuits and advance reservation system. https://www.es.net/engineering-services/oscars/.
[3] CMS Task Monitoring. http://dashb-cms-job.cern.ch/.

2018 IEEE Third International Conference on Data Science in Cyberspace

JMS: Joint Bandwidth Allocation and Flow Assignment for Transfers with Multiple Sources

Geng Li[†*], Yichen Qian[†], Lili Liu[‡], Y.Richard Yang[*]
[†]Tongji University, China, [*]Yale University, USA, [‡]Tsinghua University, China

***Abstract*—The increasing prevalence of data-intensive applications has made large-scale data transfers more important in datacenter networks. Excessive traffic demand in oversubscribed networks has caused serious performance bottlenecks. Data replicas, with the advantage of source diversity, can potentially improve the transmission performance, but current work focuses heavily on best replica selection rather than multi-source transmission. In this paper, we present JMS, a novel traffic management system that optimizes bulk multi-source transfers in software-defined datacenter networks. With a global network view and consistent data access, JMS conveys data in parallel from multiple distributed sources and dynamically adjusts the flow volumes to maximize network utilization. The joint bandwidth allocation and flow assignment optimization problem poses a major challenge with respect to nonlinearity and multiple objectives. To cope with this, we design a fair allocation algorithm that derives a novel transformation with simple equivalent canonical linear programming to efficiently achieve global optimality. Simulation results demonstrate that JMS outperforms other transmission approaches with substantial gains, where JMS improves the network throughput by up to 52% and reduces the transfer completion time by up to 44%.**

I. Introduction

During the last decade, datacenters have been continuing to thrive with the emergence of cloud-related services. Companies like Google, Microsoft and Amazon utilize datacenters to accomplish various application functions, including web search, storage, e-commerce, streaming media and large-scale computations [1], [2]. Datacenters nowadays contain up to hundreds of thousands of servers, running multiple data-intensive distributed services. Many of them relying on low-latency and high-throughput data transmission, require network operators to carefully orchestrate the large-scale transfers among servers. Sub-optimal flow routing and transfer scheduling will cause network congestion and slow transmission time.

A number of traffic engineering (TE) solutions have been developed to improve the transfer efficiency in datacenter networks. Traditional TCP-based transfer protocols reactively adjust the flow rate, which is far from optimal for satisfying transfer requirements [3], [4]. Centralized TE solutions, aiming at maximizing network utilization or minimizing flow completion time, are well investigated [5], [6]. By dynamically changing routing and rate allocation with a global network view, centralized TEs achieve better optimality. Recently, the concept of software-defined networking (SDN) that separates the control and data planes, has been increasingly exploited in datacenter networks/WANs [1], [2], [7]–[9]. SDN allows operators to directly program on the open hardware under central control, thereby making routing and engineering protocols more customized for a variety of requirements.

Data replication, amending data availability and access efficiency, can also potentially improve the transmission performance in datacenter networks [10]–[12]. However, current solutions focus heavily on best replica selection and data replication placement, instead of multi-source transmission [13], [14]. The single-source transmission with multiple data replicas results in two major shortcomings. i) When several sources have almost the same transmission performance to the destination, choosing a slightly better one and discarding all the others fails to fully utilize the network resources. ii) Selecting only one source for the whole transfer does not adapt to the dynamics because, if there are flows entering/exiting or network state changing during the transmission, the pre-selected best replica may no longer remain as the best.

In this paper, we introduce JMS, a novel traffic management system for bulk multi-source transfers in datacenter networks. Leveraging SDN principles, JMS orchestrates bulk transfers in a centralized manner. JMS's key insight is to concurrently convey data from multiple sources while dynamically adjusting the flow volumes to maximize network utilization. The joint bandwidth allocation and flow assignment optimization problem poses a major challenge stemming from a nonlinear and multi-objective formulation. To cope with this, JMS runs a novel fair allocation algorithm, which derives an transformation with simple canonical linear programming (LP) to achieve global optimality. The allocation decisions are enforced through the SDN controller to reconfigure the network and transfer sessions.

This paper presents the first approach that optimizes bulk transfers with multiple sources in software-defined datacenter networks. The **major contributions** of this paper are summarized as follows:

1) Leveraging concepts from SDN, we build a new traffic management system for bulk multi-source transfers. In particular, each transfer is accomplished by retrieving data from all available replica sources to make the best of network utilization.
2) We propose a unified data access mechanism to realize dynamic flow assignment from different sources. With data consistently divided into partitions, JMS can re-distribute the data volume and transmission rate across multiple flows in response to network dynamics.
3) We design an optimized algorithm to jointly determine bandwidth allocation and flow assignment in a max-min fair manner. The algorithm solves the nonlinear and multi-

978-1-5386-4210-8/18/$31.00 ©2018 IEEE
DOI 10.1109/DSC.2018.00026

objective problem via a novel transformation with simple equivalent canonical LP.

We perform extensive simulations, which show that JMS leads to better performance on network throughput with a gain of up to 52% and reduces transfer completion time by up to 44% compared to other transmission approaches. Furthermore, the results validate that JMS can optimize large-scale bulk transfers by continuing to provide good performance for large files and high traffic loads.

The following section briefly introduces the background and the motivation of JMS. Sec. III presents the detailed design of JMS with its constituent components. The optimal MultiSource Fair Allocation algorithm is discussed in Sec. IV. We evaluate JMS in Sec. V and introduce related work in Sec. VI. Finally, we conclude this paper in Sec. VII.

II. Background and Motivation

In this section, we outline some background, and give two motivating examples to show the benefits of leveraging multiple sources for transmission.

A. Background

Large files and bulk data transfers. There are many big-data applications generating large files and bulk data in datacenter networks, *e.g.,* scientific experiments and streaming media. These applications heavily rely on frequent data transfers for storing and retrieving large datasets. Bulk transfers have large size (terabytes) and account for a big proportion of traffic, *e.g.,* 85-95% for some datacenter networks [1], [2], [7]. High throughput and low transfer completion time is essential for their service qualities.

Centralized TE and SDN. The inefficiency of traditional TCP-based protocols motivates the work on proactive rate allocation in datacenter networks [3], [4]. Centralized TE solutions schedule packet-level flows with a global network view. Some of them aim at maximizing the aggregate network utilization with minimal scheduler overhead [5]. Priority-based flow scheduling considers not only network utilization, but also some fine-grained transfer metrics, such as minimizing flow completion time and meeting deadlines [6], [15]. Further more, SDN that leverages the network programmability, can help datacenters make the best routing decision and achieve customized scheduling [1], [2], [7]–[9].

Multiple data replicas. Data replication, a technique that amends both data availability and access efficiency, are frequently used in datacenter networks. Distributed filesystems including Google File System (GFS), Hadoop Distributed File System (HDFS) are typically deployed with a replication factor of three [10], [13], [14]. Media companies supply video replicas in different areas to allow clients have closer data access. Data-intensive scientific applications require large amounts of data in a distributed computing environment, so the experimental data is usually stored in different servers [12]. A number of approaches have been proposed for selecting the best data replica based on various criteria [11], [12]. However, those approaches only allow users to specify one replica in each selection, while discarding the others.

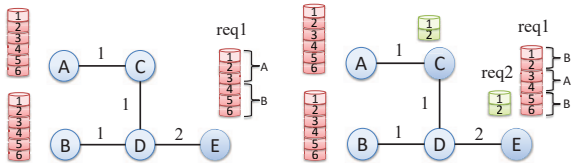


Fig. 1. Motivation examples of multi-source transfers, where bandwidth capacity is labeled on each link. (a) An example with 1 request to demonstrate that multi-source transmission outperforms the single-source one. (b) An example with 2 requests to demonstrate that dynamic flow assignment outperforms the fixed one.

B. Motivating Examples

Multiple data replicas open a new opportunity for optimizing bulk transfers in datacenter networks. Existing approaches assume a given and fixed data source for every transfer, and schedule bulk transfers by controlling the routing and the flow rate of each one. We provide two simple motivating examples to demonstrate that leveraging multi-source transmission with dynamic flow assignment outperforms single-source approaches in terms of network utilization and transfer completion time.

As the example shown in Fig. 1(a), a client sends a request req1 to download a certain file with size of 6 data units to the destination node E, and both node A and B have the source file. If we conduct the transport by the traditional single-source approach, only B is chosen as the data source with the minimum cost and $B-D-E$ as the best shortest path. Here source A is unused, and it takes 6 time units for the whole transfer task. But if we control A to send 50% of the source file and B to send the other 50%, the transfer is thus split into two flows $A-C-D-E$ and $B-D-E$, which are transmitted simultaneously. *By making complete use of the available sources, the network is fully utilized and it takes only 3 time units for the whole task, which is much faster than single-source transmission.*

Fig. 1(b) shows another example of dynamic flow assignment from multiple sources. We assume that there is another request req2 to the destination node E, and only C has the corresponding file with size 2. At the beginning, transfer 1 comes as a single flow $B-D-E$ and transfer 2 comes as the flow $C-D-E$ for transfer-level fairness. Then 2 time units later when transfer 2 is finished, we can re-adjust the flow assignment of transfer 1 and start to use both A and B to complete the last 4 units of file (3-6). In total, it takes 4 time units to finish the two transfer tasks by dynamic flow assignment from multiple sources. Whereas by the fixed flow assignment approach for multi-source transfer (1/2 from source A and 1/2 from source B), the total completion time is 2+3=5 units, and by the traditional single-source approach, the completion time is 6. *As we show, better transmission performance can be achieved by dynamically adjusting the flow volumes from feasible sources.*

III. JMS Design

JMS is a centralized system with a series of components which orchestrate bulk transfers and enforce bandwidth allo-

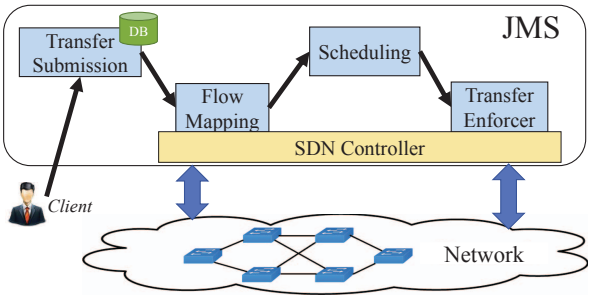


Fig. 2. JMS architecture.

cations in the datacenter network. The primary design goal is to provide high transmission performance for large files by leveraging all the available replica sources. The basic system architecture of JMS consists of four main functional components as shown in Fig. 2. *Transfer Submission* monitors clients' transfer requests and searches the candidate data sources; *Flow Mapping* calculates flow paths according to high-level routing policies then maps the flows and links according to the network state; *Scheduling* decides the data rate and flow assignment of each transfer; *Transfer Enforcer*, running within the SDN controller, reconfigures the network to start/continue transfer sessions.

In JMS, time is divided into time slots with equal length. A time slot (minutes) is much longer than the time (seconds) of reconfiguring the network and adjusting transmission rates. There is a stream of new transfers arriving at the system. So in each time slot, Flow Mapping and Scheduling will recompute the bandwidth allocation in response to the dynamic network. Transfer Enforcer will update the network configuration to orchestrate bulk transfers. Additionally, any new transfer request received in Transfer Submission can also trigger the allocation and update.

Each file in JMS is partitioned into large data blocks, where the block size is an adjustable system parameter for different files. To offer data consistency, the block number and identifiers for the same file must remain consistent in different source servers. JMS system adopts dynamic flow assignment for each multi-source transfer. By "dynamic flow assignment", we mean that the flow rate proportions from different sources are dynamically changed in every scheduling slot. JMS informs source servers about the calculated assignment in terms of data block numbers. This way, the multiple flows of the same transfer task can be finished at the same time. For instance, in the time slot with a flow assignment of (1/3, 2/3), JMS retrieves data block 1 from source A and data block 2 and 3 from source B; in the next slot with an updated flow assignment of (2/3, 1/3), JMS retrieves data block 4 and 5 from source A and block 6 from source B.

The details of each component are described as below.

1) Transfer Submission: Transfer Submission provides an interface to clients, and is responsible for real-time monitoring clients' bulk transfer requests. A request req$i$ submitted by clients is a tuple ($file_i$, dst_i) that denotes the file ID and destination address of transfer i. Transfer Submission then queries a persistent key-value database (DB) according to the file ID $file_i$. Note that there might be multiple servers storing the file replicas, so the DB can return a set of source addresses which will then be attached in req$i$ as ($file_i$, dst_i, $\{src_{ik}\}$). Transfer Submission collects all the requests and hands them to the next component.

2) Flow Mapping: Flow Mapping has a global view of the physical topology and on-going transfers with the help of SDN controller. It periodically queries for the bandwidth utilization of links and keeps track of the existing flows in each time slot. The measured bandwidth information is used as an instantaneous snapshot of the network state to compute the routing paths of new flows according to high-level network policies. Here a flow is defined as a (src, dst) pair, so the transfer with multiple available sources is decomposed into multiple parallel data flows. Flow Mapping integrates the routing paths of new flows, together with those of existing flows, into a flow-link mapping matrix (described in Section IV-B) as an output.

3) Scheduling: Taking the calculated flow paths and mapping results from Flow Mapping as the inputs, Scheduling executes the optimized MultiSource Fair Allocation algorithm in each time slot. The algorithm that computes the joint bandwidth allocation and flow assignment for each transfer, will be introduced in Section IV.

4) Transfer Enforcer: Transfer Enforcer is the direct manipulator for conducting data transmission in the datacenter network. Through SDN controller, it can install/update the flow rules on corresponding switches and set up transfer sessions on related servers. Specifically, Transfer Enforcer periodically queries for the unfinished data blocks from sources, and keeps recording the transfer state. In the mean time, it can control and manage the detailed assemblage of data blocks to be conveyed. According to the rate allocation and flow assignment results from component Scheduling, Transfer Enforcer carefully redistributes the pending data volumes among the source servers in each time slot. In brief, Transfer Enforcer instructs the servers more precisely about which data blocks to transfer, and at what data rate.

IV. Joint Bandwidth Allocation and Flow Assignment for Multi-source Transfers

One of the biggest challenges for taking best advantage of the data sources is to optimally assign flows among the sources and to allocate each flow's bandwidth. First, we must change the optimized object from the individual 5-tuple flow into the group of flows that belong to the same transfer. Hence, multiple potential bottlenecks might be considered simultaneously. Second, we need a joint optimization algorithm that computes bandwidth allocation and flow assignment at once.

A. Network Model and Problem Formulation

Consider a telecommunications network composed of a set of nodes and a set of links $\mathcal{L}$. The capacity of link L_j ($j \in [1, M]$) is defined as C_j. Suppose there are a number of data transfer requests, each of which may come from multiple

sources, with the paths pre-calculated and given. Thus each transfer i ($i \in [1, N]$) is assigned with a set of flow paths $\{P_{i1}, ..., P_{ik}\}$, and each such path is identified with the set of links that it traverses, *i.e.,* $P_{ik} \subseteq \mathcal{L}$. Now let r_i denote the data rate of transfer i, which is the sum bandwidth of the constituent flows from all its sources. We use a variable set $\mathcal{X}_i = \{x_{i1}, ..., x_{ik}\}$ to express the flow assignment proportions from different sources for transfer i, and $\sum_{k=1}^{K_i} x_{ik} = 1$, where K_i is the total source number of transfer i.

We are interested in solving the joint bandwidth allocation and flow assignment problem in each time slot, *i.e.,* finding the transmission rate r_i of each transfer, together with the assignment proportions $\mathcal{X}_i$ from all its sources. The solution is required to provide a fair and efficient allocation result, and its precise objective and constraints are described as below.

Objective. When computing allocated bandwidth, our goal is to maximize network utilization while in a max-min fair manner. A vector of transfer rate allocations $\{r_i\}$ in a slot is said to be max-min fair if, for any other feasible allocation $\{r'_i\}$, the following has to be true: if $\exists r'_p > r_p$ for the data transfer p, then there exists another transfer q such that $p, q \in [1, N]$, $r'_q < r_q$, $r_q \leq r_p$. In other words, increasing some components must be at the expense of decreasing some other existing smaller or equal components.

Constraints. The constraints of this problem are given as below. Constraint (1) is called the capacity constraint, which assures that for any link L_j, its load does not exceed its capacity C_j. Constraint (2) promises the sum fractions of a transfer from all available sources equal to 1.

$$s.t. \quad \sum_{L_j \subseteq P_{ik}} r_i \cdot x_{ik} \leq C_j \qquad \forall j, \qquad (1)$$

$$\sum_{k=1}^{K_i} x_{ik} = 1 \qquad \forall i, \qquad (2)$$

$$0 \leq x_{ik} \leq 1 \qquad \forall i, k. \qquad (3)$$

Fig. 3 shows an example of the network consisting of six links $\{L_1, ..., L_6\}$, with capacities $\{10, 7, 12, 8, 4, 8\}$ respectively. Assume all bandwidth numbers are in *Gbps* here. Suppose there are three data transfer requests in the current time slot, among which transfer 1 and 2 have a single data source while transfer 3 has two available sources. In total, there are four potential flows in the network with given paths, including $f_1 : A \to B \to C$, $f_2 : A \to E \to F$, $f_{31} : A \to B \to C \to D$ and $f_{32} : E \to F \to D$. Given the topology and values of link capacities, we aim at finding the max-min fair solution of the rate vector $\{r_1, r_2, r_3\}$ and the flow assignment $\{x_{31}, x_{32}\}$ for transfer 3.

B. Flow-link Mapping (FL) Matrix and Single Source Case

To solve the problem, we propose a multi-source allocation algorithm that maximizes network utilization while providing global max-min fairness. The proposed algorithm is fundamentally based on a flow-link mapping matrix (FL matrix) with solvable variables, which is the output of component Flow Mapping and the input of Scheduling in system JMS. In this section, we define the FL matrix, and illustrate the traditional water-filling algorithm for single source case with this matrix.

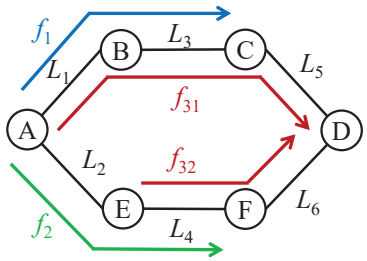


Fig. 3. An example with 3 transfer requests. Transfer 3 comes from two feasible sources A and E, corresponding to two parallel flows f_{31} and f_{32}.

Algorithm 1 Traditional Water-Filling Algorithm

Input:
Flow-link mapping matrix: $FL = \{fl_{ij}\}$;
Capacity of each link: $\{C_j\}, 1 \leqslant j \leqslant M$;
Output:
Transmission rate of each transfer: $\{r_i\}, 1 \leqslant i \leqslant N$;

```
while num of rows in FL ≠ 0 do
    n_j ← Σ_i fl_ij,   ∀1 ⩽ j ⩽ M;
    τ_j ← C_j / n_j,   ∀1 ⩽ j ⩽ M;
    Find τ* ← min{τ_j}, j* ← j|τ_j = τ*;
    Set r_i ← τ*, for i|fl_ij* = 1;
    Update FL;
end while
return {r_i}.
```

Definition 1 (Flow-link Mapping Matrix)**.** The flow-link mapping matrix *(FL matrix)* $\{fl_{ij}\}$ expresses the flow paths and the traffic shares of each transfer in matrix form. The matrix element fl_{ij} is defined as the proportion of flow i in its belonging transfer that traverses link L_j.

For single source case, since every transfer comes as an individual flow, the matrix elements are either 0 or 1. In this simple case, the bandwidth allocation can be plainly obtained by the traditional water-filling algorithm (**Algorithm 1**). Using the FL matrix as an input, we first denote the saturated average bandwidth allocation as $\tau_j = C_j/n_j$, where n_j is the total number of flows that use link L_j. The algorithm iteratively finds the minimum τ^* and the corresponding bottleneck link L_{j^*}. Set the bandwidth of the flows that use link L_{j^*} to τ^*. Then update the FL matrix by subtracting those flows and the bottleneck link to calculate a new set of $\{\tau_j\}$. Such process iterates until all transfers obtain their allocated rates, *i.e.,* all flows have bottleneck links.

Consider a single-source version of Fig. 3, where we assume transfer 3 only uses source A, so there are 3 flows over the network. Fig. 4 illustrates the allocation algorithm for this single-source example. In the first iteration, L_5 is first saturated because τ_5 is of the minimum value $\tau^* = 4$. We allocate the bandwidth of f_{31} to 4, and then remove the row of f_{31} and column L_5. The FL matrix is updated for the next iteration, where τ_1 and τ_3 are changed accordingly. By the same token, L_1 and L_3 are then found as the bottleneck links in the

	L_1	L_2	L_3	L_4	L_5	L_6
f_1	1	0	1	0	0	0
f_2	0	1	0	1	0	0
f_{31}	1	0	1	0	1	0
n_j	2→1	1	2→1	1	1	0
C_j	10→6	7	12→8	8	4	8
τ_j	5→6	7	6→8	8	4	*NA*

Fig. 4. An example of the traditional water-filling algorithm with the FL matrix, where C_j is the bandwidth capacity, $n_j = \sum_i fl_{ij}$ is the total number of flows that use link L_j, and $\tau_j = C_j/n_j$ is the saturated average bandwidth share. In the single-source example, L_5 is first found as the bottleneck link, and then the matrix is updated for the next iteration by removing f_{31} and its bandwidth.

following two rounds of iteration respectively. The ultimate solution to the rate allocation is (6, 7, 4) for the 3 transfers.

The traditional water-filling algorithm succeeds in obtaining the max-min fair allocation in the single-source case, yet it is incapable of dealing with multi-source transfers. This principally stems from the fact that the algorithm is based on flows, rather than on transfers. For the example in Fig. 3, transfer 3 will have double weights by using the water-filling algorithm, which is unfair to the others. A naïvely improved solution is to normalize the weight of each transfer, and to assign an equal share to the flows from different sources. With regard to the example, the elements for f_{31} and f_{32} become 1/2 and 1/2 in the FL matrix, such that each transfer has the same sum weight of 1. The allocation result turns into (20/3, 16/3, 6), which is slightly better than the single-source solutions, which are (6, 7, 4) for using only source A and (10, 4, 4) for using only source E. However, as we will soon learn, equally sharing the flow weight is still not the optimal solution. *The transfer-level max-min fair allocation is conditioned by the optimal flow assignment.*

C. MultiSource Fair Allocation (MSFA) Algorithm

Given the limitation of the water-filling algorithm, we propose a MultiSource Fair Allocation (MSFA) algorithm to handle the optimization with multi-source transfers. For clear illustration, we consider only one multi-source transfer m who has K_m available sources and the flow assignment $\mathcal{X}_m$ is to be solved. Arbitrary multi-source transfers can be simply extended from it. The MSFA algorithm continues to use the FL matrix as an input, but the elements are no longer 0 and 1 as in the single-source case. Instead, the matrix elements related to transfer m are replaced by the unknown variables in $\mathcal{X}_m$. The MSFA algorithm (**Algorithm 2**) can be described in the following high-level steps:

1) Start from zero allocation with the whole link set, and build the FL matrix with variables $\mathcal{X}_m$.
2) Calculate the saturated average bandwidth $\tau_j(\mathcal{X}_m)$ on each link L_j.
3) Find the bottleneck fair share $\tau^* = min\{\tau_j(\mathcal{X}_m)\}$ by solving $\mathcal{X}^* = \mathcal{X}_m | max\, min\{\tau_j(\mathcal{X}_m)\}$.
4) Set the data rate to τ^* for the flows that traverse the bottleneck links, and update the FL matrix by removing those flows and links.

Algorithm 2 MSFA Algorithm

Input:
Flow-link mapping matrix: $FL = \{fl_{ij}\}$;
Capacity of each link: $\{C_j\}, 1 \leqslant j \leqslant M$;
Output:
Transmission rate of each transfer: $\{r_i\}, 1 \leqslant i \leqslant N$;
Flow assignment of transfer m: $\mathcal{X}_m = \{x_{m1}, ..., x_{mk}\}$;

- **Step 1:** ▷ Initiation
  $r_i \leftarrow 0, \quad \forall i = 1, ..., N$;
  $\mathcal{L} \leftarrow \{L_1, L_2, ..., L_M\}$;
- **Step 2:** ▷ Calculate the saturated average bandwidth
  $n_j(\mathcal{X}_m) \leftarrow \sum_i fl_{ij}, \quad \forall j \in \mathcal{L}$;
  $\tau_j(\mathcal{X}_m) \leftarrow C_j/n_j(\mathcal{X}_m), \quad \forall j \in \mathcal{L}$;
- **Step 3:** ▷ Find the bottleneck fair share τ^*
  if $min\{\tau_j(\mathcal{X}_m)\}$ is a constant **then**
    $\mathcal{X}^* \leftarrow \varnothing$;
  else
    $\mathcal{X}^* \leftarrow \mathcal{X}_m | max\ min\{\tau_j(\mathcal{X}_m)\}$;
  end if
  $\tau^* \leftarrow min\{\tau_j(\mathcal{X}_m)\}$;
- **Step 4:** ▷ Set data rate and update the FL matrix
  $\mathcal{L}_{j^*} \leftarrow \{L_j | \tau_j(\mathcal{X}_m) = \tau^*\}$;
  $\mathcal{L} \leftarrow \complement_{\mathcal{L}_{j^*}} \mathcal{L}$;
  for $i | P_i \cap \mathcal{L}_{j^*} \neq \varnothing$ **do**
    $r_i \leftarrow r_i + \tau^*$;
    Remove f_i from FL;
  end for
  Update FL;
- **Step 5:** ▷ Iteration
  if No transfers left **then**
    return $\{r_i\}$ and $\mathcal{X}_m$;
  else
    goto **Step 2**.
  end if

5) If there are no transfers left then stop, otherwise return to Step 2.

In Step 2, similar to the traditional water-filling algorithm, we first calculate $n_j(\mathcal{X}_m)$ by summarizing the elements of column j in the FL matrix. Here $n_j(\mathcal{X}_m)$ denotes the number of transfers that use link L_j. Due to the fact that transfer m comes from multiple parallel flows, there might be only a fraction of the transfer using link L_j. As a result, n_j becomes a function of $\mathcal{X}_m$ instead of an integer value as in the single-source case. Next, we compute the average bandwidth $\tau_j(\mathcal{X}_m) = C_j/n_j(\mathcal{X}_m)$, which is also a function of $\mathcal{X}_m$.

In Step 3, given $\{\tau_j(\mathcal{X}_m)\}$ on the current link set, one or several bottleneck links are found. Specifically, since all the variables $\mathcal{X}_m$ are within a certain range $[0, 1]$, $min\{\tau_j(\mathcal{X}_m)\}$ is sometimes a constant value. In that case, no flow assignment variable is calculated, *i.e.,* $\mathcal{X}^* = \varnothing$. Otherwise, the flow assignment is determined by $\mathcal{X}^* = \mathcal{X}_m | max\, min\{\tau_j(\mathcal{X}_m)\}$. We use τ^* to denote the minimum bandwidth share, and the set of $\{L_{j^*}\}$ is found as the bottleneck links.

Back to the example in Fig. 3, transfer 3 has two available sources to access, resulting in two parallel flows f_{31} and f_{32}, respectively. For simplicity, we use only one variable x to denote the proportion of f_{31}, and that of f_{32} will thus be $1-x$. Fig. 5 illustrates the FL matrix, followed by the saturated average bandwidths $\{\tau_j(\mathcal{X}_m)\}$.

From the example, we can tell that *Step 3, finding* $\mathcal{X}^*$

	L_1	L_2	L_3	L_4	L_5	L_6
f_1	1	0	1	0	0	0
f_2	0	1	0	1	0	0
f_{31}	x	0	x	0	x	0
f_{32}	0	0	0	1-x	0	1-x
n_j	1+x	1	1+x	2-x	x	1-x
C_j	10	7	12	8	4	8
τ_j	$\frac{10}{1+x}$	7	$\frac{12}{1+x}$	$\frac{8}{2-x}$	$\frac{4}{x}$	$\frac{8}{1-x}$

Fig. 5. An example of MSFA, where x is the flow assignment variable of transfer 3. Two bottleneck links (L_1 and L_4) are found in the first iteration, where $x = 2/3$ is solved by a simple equivalent LP. The rate allocation is determined in one shot as (6, 6, 6).

that maximizes $min\{\tau_j(\mathcal{X}_m)\}$, is the major challenge in MSFA. Accordingly, it is to find $x^* = x|max\, min(10/(1+x), 7, 12/(1+x), 8/(2-x), 4/x, 8/(1-x))$ in Fig. 5. First, as formulated in **Problem 1**, it is a nonlinear programming problem, which can not be solved directly. Second, even if we can find a linear expression, we still need long sequences of LPs for the max-min objective (multi-objective), which is computationally intense in real implementation.

Problem 1 (The optimization problem in Step 3)**.**

$$max \quad min\{\tau_j(\mathcal{X}_m)\}, \tag{4}$$

$$s.t. \quad \sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{5}$$

$$0 \le x_{ik} \le 1 \qquad \forall i,k. \tag{6}$$

To cope with this, we transform this nonlinear optimization problem into a canonical form of LP problem based on **Theorem 1**. Accordingly, the equivalent canonical LP problem expressed as **Problem 2** can be solved efficiently with limited computational complexity.

Problem 2 (The equivalent LP problem in Step 3)**.**

$$min \quad t \tag{7}$$

$$s.t. \quad \sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{8}$$

$$0 \le x_{ik} \le 1 \qquad \forall i,k, \tag{9}$$

$$t \ge \tau'_j(\mathcal{X}_m) \qquad \forall j. \tag{10}$$

Theorem 1. *Problem 1 is equivalent to Problem 2 as a canonical LP problem, where $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$.*

Proof: Given an arbitrary instance of the FL matrix, $\tau_j(\mathcal{X}_m)$ satisfies two conditions: i) $\tau_j(\mathcal{X}_m) \ge 0$, ii) the inverse of $\tau_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$. So we let $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$, and the objective of $max\, min\{\tau_j(\mathcal{X}_m)\}$ is then equivalent to $min\, max\{\tau'_j(\mathcal{X}_m)\}$, which becomes linear accordingly. Next, we introduce a temporary variable $t = max\{\tau'_j(\mathcal{X}_m)\}$, and use a sequence of constraints $t \ge \tau'_j(\mathcal{X}_m)$ for all j to express t. Since $\tau'_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$, the constraint (10) as a set of inequalities is also linear. In the end, the optimization problem in Step 3 (**Problem 1**) turns into an equivalent canonical LP problem (**Problem 2**), where the decision variables to be solved are the flow assignment set $\mathcal{X}_m$ and t. ■

As a result, the optimization problem of the example in Fig. 5 is well transformed into a simple equivalent LP problem, which is expressed as below.

$$min \quad t \tag{11}$$

$$s.t. \quad 0 \le x \le 1. \tag{12}$$

$$\begin{pmatrix} 10 & 7 & 12 & 8 & 4 & 8 \\ -1 & 0 & -1 & 1 & -1 & 1 \end{pmatrix}^T \begin{pmatrix} t \\ x \end{pmatrix} \ge \begin{pmatrix} 1 & 1 & 1 & 2 & 0 & 1 \end{pmatrix}^T \tag{13}$$

The results of the above LP come out as $x = 2/3$ and $t = 1/6$. So $\tau^* = 1/t = 6$ is the minimum fair share for the bottleneck links L_1 and L_4. We set $r_1 = r_2 = 6$ as the bandwidth of f_1 and f_2, and $r_3 = 4+2$ as the sum bandwidth of f_{31} and f_{32}. The bandwidth of all flows is solved in one shot, and the final rate allocation (6, 6, 6) is more max-min fair than the allocation (6, 7, 4) where transfer 3 uses only source A (as in Fig. 4), as well as the allocation (10, 4, 4) where transfer 3 uses only source E.

The MSFA algorithm is scalable and extensible for more complex use cases. For instance, differentiated qualities of service lead to transfers with variations in priority or other requirements, such that MSFA is capable of supporting weighted fair allocation by taking priority factors into account. The max-min fair principle can also be applied to different optimization objectives (*e.g.,* transfer completion time), and the adaption of MSFA with consideration of data size can likewise yield the optimal results for multi-source transfers. The computation complexity is significantly reduced in MSFA by transforming a nonlinear multi-objective problem into a single LP, and it can be further reduced by removing the redundant constraints in implementation. Thus MSFA with limited complexity is scalable to handle a datacenter network with 100s of switches.

V. Performance Evaluation

To evaluate how JMS works on a large-scale network, we implement a flow-level datacenter network simulator.

A. Simulation Methodology

Topologies: We conduct our experiments by emulating a 3-tier datacenter network topology with 8:1 oversubscription. The topology contains 64 servers, whereby each edge link is of $1Gbps$ capacity, and aggregated link is of $10Gbps$ capacity.

Workloads: We synthesize a stream of transfer requests with a total number of 1000. The request arrival is modeled as a Poisson process, where the arrival rate λ is defined as the average number of new transfers per time slot. We set the slot length as one second for fast simulation. A transfer has multiple sources with probability ρ. A multi-source transfer is assumed to have a random number of replicas between [2,5],

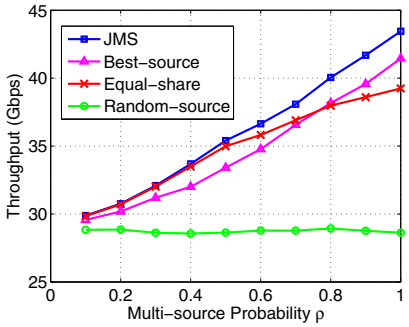


Fig. 6. Network throughput vs. multi-source probability ρ, where the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$.

and the replicas are randomly placed in servers. In simulations, we do not consider the fluctuation of transfer size, and assume all transfers have an uniform size of V.

Performance metrics: We use *network throughput* and *average transfer completion time* to show the improvements of JMS over other approaches.

Traffic engineering: We compare the following TE approaches, each of which computes per-flow bandwidth allocation with max-min fairness.

- **JMS:** The approach in this paper, runs MSFA algorithm to dynamically assigns flows from different sources.
- **Best-source:** This approach selects a best replica source based on the algorithm in [12] for multi-source transfers.
- **Equal-share:** This approach equally splits the transfer across different sources. For example, if a transfer has 3 replicas, each replica will send 1/3 of the data.
- **Random-source:** This approach randomly selects an available source to transmit data.

B. Simulation Results

Fig. 6 shows the simulation results of network throughput for various TE approaches. Here we set the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$ for all of the 1000 transfers. As the results show, Random-source approach disregards the source dissimilarity, therefore performs the worst with a constant throughput value. Equal-share approach takes advantage of source diversity simplistically. When the diversity is limited to a small number of multi-source transfers (at low multi-source probabilities), equal flow sharing approximates the optimal assignment, and thus obtains the similar performance as JMS. But as the multi-source proportion increases, there are more flows entering the network. The effect of "bad flows" enlarges, and hence drags down the overall throughput improvement. Accordingly, Best-source approach begins to outperform Equal-share. By jointly optimizing the bandwidth allocation and flow assignment, JMS achieves a much higher throughput than the others, resulting in higher utilization of the network. When all the transfers have multiple sources ($\rho = 1$), JMS obtains a substantial throughput gain of up to 52% compared to the single-source transmission.

Fig. 7 compares the transfer completion time versus three factors respectively: the multi-source probability ρ, the transfer size V and the transfer arrival rate λ. It is shown that, JMS achieves the smallest transfer completion time across all configurations. This improvement is derived from two aspects: 1) optimized transmission rate from a more max-min fair allocation by MSFA, 2) dynamic flow assignment to remain optimal as new transfers arrive and existing transfers complete.

Fig. 7(a) focuses on the impact of multi-source probability ρ. Approximately, ρ equals to the proportion of multi-source transfers. The gap between Random-source and the other approaches verifies that, leveraging multiple sources is more efficient for data transmission. Moreover, the sustained decline in completion time with the multi-source probability implies that, the multi-source transmission obtains more performance gains by placing more replicas in datacenters. Compared to single-source transmission, JMS reduces the average completion time by up to 44%.

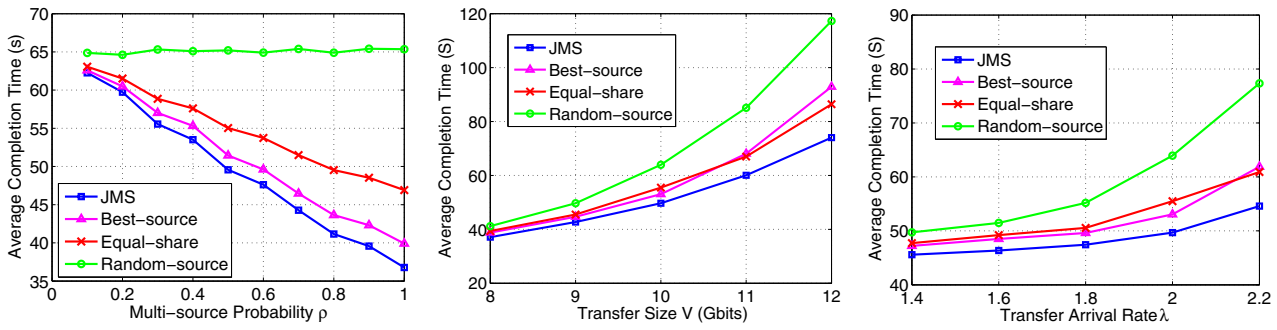
Fig. 7(b) shows the relationship between completion time and the transfer size V. As the transfer size increases, the transfer backlog starts to cause more bottleneck links, which leads to the degradation of transmission rates. Therefore the completion time increases superlinearly along with the transfer size for all the approaches. But the almost linear completion time growth of JMS suggests that, by completing transfers as quick as possible, JMS is capable of optimizing bulk transfers for small files as well as large files.

Fig. 7(c) illustrates the impact of transfer arrival rate λ. As expected, at higher rates, the number of transfers over the network potentially increases, and links are more likely to become congested. Accordingly, the performance degrades quickly for all the approaches except JMS. The smaller growth in completion time demonstrates that, by effectively avoiding the congestion point, JMS manages to handle relatively a larger amount of traffic without degrading the performance.

VI. Related Work

Datacenter traffic management: Most of the recent work in datacenter networks aims at scheduling flows with centralized TE to maximize aggregate network utilization [5], [9]. DevoFlow [9] is a modification of the OpenFlow model to reduce the number of switch-controller interactions. Some work focuses on the optimization of some fine-grained transfer metrics, such as minimizing flow completion time and meeting deadlines [6], [15]. PDQ achieves better deadline-meeting rate by allocating different priorities [6]. However, none of them takes into consideration that data replication provides more sources to improve transmission performance.

Distributed filesystems: Several high-performance distributed filesystems with sufficient data replicas have been developed, including HDFS [13] and Quantacast File System [14]. Sinbad [11] is the first distributed filesystem that utilizes replica placement flexibility to avoid congested links for write operations. Leveraging SDN, Mayflower [12] performs global optimizations to make intelligent replica selection and flow scheduling decisions. Nevertheless, all the systems completely rely on single-source transmission, instead of conveying data in parallel from multiple distributed sources to adapt to the variability of network bandwidths as in JMS.



(a) (b) (c)

Fig. 7. Impact of (a) the multi-source probability ρ, (b) the data size V and (c) the transfer arrival rate λ on average transfer completion time. In each subfigure, we adjust one factor and fixed the other two, with three default values $\rho = 0.5$, $V = 10$ and $\lambda = 2$.

Task-based flow scheduling: The approaches that schedule parallel flows have been developed to optimize transfers at the level of coflow. Coflow [16] and Varys [17] improve application-level performance by minimizing coflow completion times. However, their basic assumption is the flow volumes for different data are designated in advance, so they can easily predict the completion time and allocate the rate to meet their deadlines. JMS's improvement over them is the dynamic flow volume assignment, which enables redistribution of the pending data among available sources for the same data. Moreover, this assignment is jointly optimized with bandwidth allocation to achieve global optimality.

VII. Conclusion

We present JMS, a novel traffic management system that orchestrates bulk transfers with multiple sources in software-defined datacenter networks. JMS conveys data in parallel from multiple sources and dynamically adjusts the flow volumes to maximize the network utilization. The core of JMS is an online fair allocation algorithm that jointly computes the bandwidth allocation and flow assignment with simple equivalent canonical LP to achieve global optimality. Extensive simulations validate that, compared to other transmission approaches, JMS achieves a better throughput gain of up to 52% and decreases transfer completion time by up to 44% for large-scale bulk transfers.

Acknowledgment

This research was supported in part by NSFC #61701347, NSFC #61702373 and NSFC #61672385; NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM CCR*, 2013.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *SIGCOMM CCR*, 2013.

[3] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *INFOCOM*, 2013.

[4] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *ACM SIGCOMM CCR*, 2012.

[5] S. Radhakrishnan, M. Tewari, R. Kapoor, G. Porter, and A. Vahdat, "Dahu: Commodity switches for direct connect data center networks," in *ANCS*, 2013.

[6] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *SIGCOMM CCR*, 2012.

[7] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *Proceedings of SIGCOMM 2016 Conference*.

[8] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," in *SIGCOMM CCR*, 2015.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM CCR*, 2011.

[10] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *IEEE Transactions on Cloud Computing*, 2017.

[11] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *ACM SIGCOMM CCR*, 2013.

[12] S. Rizvi, X. Li, B. Wong, F. Kazhamiaka, and B. Cassell, "Mayflower: Improving distributed filesystem performance through sdn/filesystem co-design," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, 2016.

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, 2010.

[14] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, "The quantcast file system," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, 2013.

[15] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM CCR*, 2011.

[16] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of HotNet*, 2012.

[17] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *ACM SIGCOMM CCR*, 2014.

Unicorn: Unified Resource Orchestration for Multi-Domain, Geo-Distributed Data Analytics

Qiao Xiang[+‡], Shenshen Chen[+], Kai Gao[*‡], Harvey Newman[◇],
Ian Taylor[○†], Jingxuan Zhang[+], Yang Richard Yang[+‡]
[+]Tongji University, [‡]Yale University, [*]Tsinghua University,
[◇]California Institute of Technology, [○]Cardiff University, [†]University of Notre Dame
{qiao.xiang, kai.gao, yang.r.yang}@yale.edu, {jingxuan.zhang, cs9091}@tongji.edu.cn,
newman@hep.caltech.edu, taylorij1@cardiff.ac.uk

***Abstract*—Data-intensive analytics is entering the era of multi-organizational, geographically-distributed, collaborative computing, where different organizations contribute various resources, *e.g.*, sensing, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. This new paradigm calls for a framework to manage a large set of distributively owned heterogeneous resources, with the fundamental objective of efficient resource utilization, following the autonomy and privacy of resource owners. In this paper, we design Unicorn, the first unified framework that accomplishes this goal. The foundation of Unicorn is RSDP, an autonomous, privacy-preserving resource discovery and representation system to provide accurate resource availability information. Its core is a novel abstraction called resource vector abstraction which describes the resource availability in a set of linear constraints. In addition, Unicorn also provides a series of advanced solutions to support automatic, efficient management of resource dynamics on both supply and demand sides, including an automatic workflow transformer, an intelligent resource demand estimator and an efficient, scalable multi-resource orchestrator. Being the first unified framework for this new paradigm, Unicorn plays a fundamental role in next-generation data-intensive collaborative computing systems.**

***Keywords*—multi-domain, data analytics, resource allocation.**

I. Introduction

As the data volume increases exponentially over time, data-intensive analytics is transiting from single-domain computing to multi-organizational, geographically-distributed, collaborative computing, where different organizations contribute various resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. Examples of this paradigm include the Compact Muon Solenoid (CMS) experiment at CERN [1], coalitions between different combating units, etc. Figure 1 summarizes the general settings of collaborative computing: data-intensive analytics workflows consume resources supplied by participating sites/resource owners, coordinated by a logically centralized orchestrator. Collaborative computing calls for a framework to manage a large set of distributively owned heterogeneous resources, with the fundamental objective of *efficient resource utilization, following the autonomy and privacy of resource owners*. To achieve this goal, this framework must be capable of the following functionalities.

- **Managing resource supply dynamic.** This requires a system for joint computation, storage and networking resource discovery and representation, which allows resource owners to make and practice their own resource supply strategies without exposing private information. Without such a component, it is infeasible to manage a large set of distributively owned, heterogeneous, dynamic resources.
- **Managing resource demand dynamic.** This requires a system for automatic, effective resource demand estimation, which automatically transforms high-level data analytics workflows (*e.g.*, Spark) to low-level task workflows (*e.g.*, HTCondor ClassAds) and finds the optimal configuration, *i.e.*, resource demand, for each task. Without such a component, users have to manually configure the low-level workflows for data analytics. On one hand, users might request more than necessary resources for workflows, resulting in inefficient use of resources. On the other hand, if users request insufficient resources for workflows, the system may need more resource distribution transactions, resulting in larger overhead.
- **Matching demand with supply.** This requires a system for efficient, scalable multi-resource orchestration, which makes efficient resource allocation decisions for analytics workflows based on resource supply and demand. This component is essential for achieving efficient resource utilization.

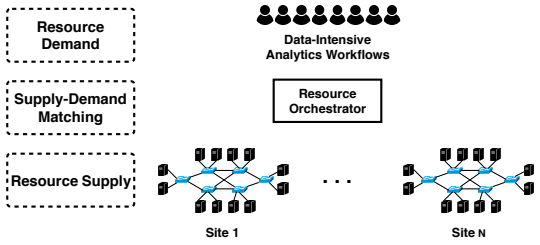


Fig. 1. General settings of multi-organization, geo-distributed, data-intensive collaborative computing: (1) users submit analytics workflows to produce resource demand; (2) different sites/resource owners provide resource supply; (3) a logically centralized orchestrator matches demand with supply, *i.e.*, allocating resources to analytics workflows.

In this paper, we propose Unicorn, the first unified framework providing all these functionalities for multi-organizational, geographically-distributed collaborative computing.

The fundamental design challenge for Unicorn is: *how to accurately discover and represent resource availability in a large set of distributively owned heterogeneous resources*? Although there is much related work on resource management of cluster

computing [2]–[11], current systems are mostly designed for a single administrative domain and focus on computation and storage resources by assuming the networking resource is not a bottleneck. Such settings usually lead to easier designs. In particular, current systems typically adopt a graph representation to describe resource availability, where each node is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource. In the multi-domain collaborative computing, where resources are distributively owned and all resources (*i.e.*, computation, storage and networking) have the same probability of becoming the bottleneck of analytics [12], this graph representation has quite a few drawbacks. First, it could lead to race conditions between resource suppliers and consumers. Secondly, multi-resource interferences would lead to inefficient use of allocated resources. Thirdly, this representation hides the underlying resource sharing between nodes or edges in the physical topology, which leads to the over-provisioning of resources.

To address these drawbacks and the design challenge, we developed RSDP, an autonomous, privacy-preserving resource discovery and representation system, to accurately represent available resources in collaborative computing systems. This is achieved through a novel abstraction called *resource vector abstraction*, which describes the resource availability using a set of linear constraints.

With RSDP managing the resource supply dynamic, the Unicorn framework also provides the Handyman system to manage the resource demand dynamic, which automatically transforms high-level analytics workflows to low-level task workflows and finds them the optimal configurations. Between supply and demand, Unicorn designs an efficient, scalable multi-resource orchestrator called Miro to achieve efficient resource utilizations.

The rest of the paper is organized as follows. We first give an overview of the Unicorn framework and introduce its core components in Section II. We then present the details of RSDP, the core resource discovery and representation system of the Unicorn framework in Section III and its preliminary evaluation results in Section IV. We briefly discuss related work in Section V and conclude the paper in Section VI.

II. System Architecture of Unicorn

Unicorn aims to achieve two design goals simultaneously for data-intensive collaborative computing: (1) achieve efficient utilization of a large set of distributively owned heterogeneous resource, and (2) allow each participating site to practice policies and protocols at its own choices without revealing private information. The first goal ensures that the available resource supply is efficiently matched to the resource demand of data-intensive analytics workflows. The second goal ensures the autonomy, privacy and security of each participating site.

Figure 2 gives the overall architecture of the Unicorn framework, which consists of three components. The foundation of Unicorn is RSDP, an autonomous, privacy-preserving resource discovery and representation system to accurately represent availability information of a large set of distributively owned resources. On the resource demand side, Unicorn provides Handyman, an analytics demand automation system, which automatically converts high-level analytics workflows into low-level task workflows and finds the optimal configuration (*i.e.*, resource demand) for each task. Between resource demand and supply, an efficient, scalable multi-resource orchestrator called Miro is provided in Unicorn to efficiently utilize resources in the system for data-intensive analytics workflows.

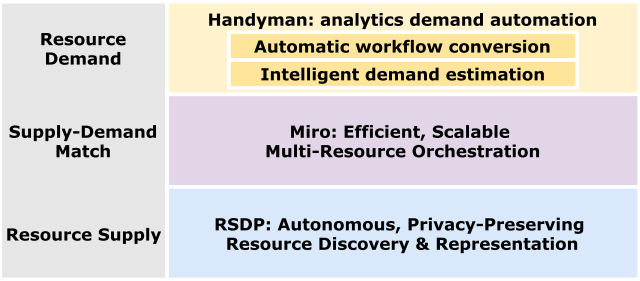


Fig. 2. The Architecture of Unicorn framework.

RSDP: an automatic, privacy-preserving resource discovery and representation system. RSDP provides an accurate view on resource supply dynamic and is the foundation of the Unicorn framework. This is achieved through a novel abstraction called *resource vector abstraction*. Given a set of tasks T, a set of resources R and a set of resource attributes P, resource vector abstraction uses a set of linear constraints to represent the feasibility and constraints of resource availability and sharing. When a set of original linear constraints C is obtained, RSDP does not directly return the whole set as the resource availability information. Instead, it uses a lightweight, optimal algorithm to compress C into a minimal, equivalent set of constraints C' where the feasible regions represented by C and C' are the same. Through this compression, RSDP (1) avoids the high communication overhead of transmitting resource availability information between the orchestrator and sites, and (2) minimizes the risk of each site exposing private information about its computing facilities, *e.g.*, topology, policies, etc.

Handyman: an analytics demand automation system. Handyman automatically (1) converts high-level analytics workflows into low-level task workflows, *i.e.*, a set of tasks with precedence encoded in a directed acyclic graph (DAG), and (2) finds the optimal configuration for each task. This component saves users from the trouble of manually specifying low-level task workflows. During the conversion, a critical challenge must be addressed is that high-level analytics workflows are sometimes stateful while low-level workflows are not. To guarantee that the workflows are equivalent before and after conversion, the current Handyman design performs conversion against the state, and schedules automatic conversion re-execution when state changes happen. Another solution under investigation involves changing the programming models of low-level task workflows by allowing them to be stateful.

After the conversion, Handyman automatically estimates the optimal configuration (resource demand, *e.g.*, the number of CPUs, the size of memory and disk, I/O bandwidth, etc.) for each task. It leverages the fact that low-level tasks are typically repetitive with strong similarities and applies reinforcement learning to estimate optimal configurations for similar tasks. In articular, it records the utilization and configurations of each task executed and predicts the resource utilization for different configurations. When a task comes, Handyman chooses an unrecorded configuration with the best prediction, and records the resource utilization of the actual execution for next round of reinforcement learning.

Miro: an efficient, scalable multi-resource orchestrator. Miro makes resource allocation decisions based on resource demand and supply dynamic. It receives the resource demand information, *i.e.*, a set of low-level task workflows and their configurations, from Handyman. Then it sends queries to the RSDP system at each site with a *what-if question*: what resource would be provided if the requested tasks were to be executed here? After receiving the responses encoded in resource vector abstraction, Miro looks up different resource provisions and makes task placement decisions to guarantee that the available resources are efficiently matched with the resource demand of each task. These decisions are then sent to the task execution agents at each site, who execute the tasks, monitor the progress and resource usage and report back to Miro. Miro supports different scheduling modes, including FIFO, global mixed binary programming, least-demand-first (LDF), etc. Comparing and understanding the performance of different scheduling modes is the next step of Miro.

The most fundamental design challenge of Unicorn is how to accurately discover and represent the resource availability over a large set of distributively owned heterogeneous resources. In the next two sections, we focus on addressing this challenge and present our solution RSDP, the foundation of Unicorn. Details of Handyman and Miro are omitted for the interests of brevity.

III. RSDP: Autonomous, Privacy-Preserving Resource Discovery and Representation

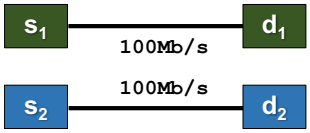
In this section, we first review the limitation of resource availability graph, a common resource discovery design adopted in most current resource management systems (Section III-A). We then propose RSDP, an autonomous, privacy-preserving resource discovery and representation framework to accurately represent available resources in collaborative computing systems, in which the resource availability is encoded in a novel abstraction called *resource vector abstraction* as a set of linear constraints (Section III-B). Next, we discuss the challenges of generalizing the resource vector abstraction design and some practical concerns for production deployment (Section III-C).

A. Resource Availability Graph: Incomplete State-of-the-Art

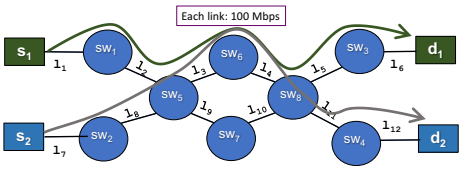
Basic idea. Computer systems typically consist of three types of resources: computation, storage, and networking. A typical design of resource discovery adopted by most current systems is a *resource availability graph*, which is built on top of the physical topology. Each node in the graph is a physical node, with computation/storage resources annotated, and each edge between a pair of nodes is annotated with available networking resources.

Drawbacks. The main assumption of a resource availability graph is that node and link attributes are described as independent variables. However, it has several drawbacks when used for resource discovery of a large set of distributively owned resources. First, the multi-domain nature of collaborative computing has determined that race conditions could happen frequently using resource availability graphs. Suppose a resource owner announces the availability of a set of resources, and two different tasks both want to use these resources. Without careful synchronizations, both tasks may try to use this set of resources, leading to conflicts.

Secondly, current systems mainly focus on discovering computation and storage resources (*e.g.*, CPU, memory and disk) and assume networking resource is not a bottleneck. However, recently during a production trace of cluster computing, it was shown that all three types of resources have approximately the same probability of becoming a bottleneck in affecting the performance of data-intensive analytics [12]. With this fact, the graph representation could lead to inefficient use of allocated resources. Consider an example where nodes A, B and edge $\{A, B\}$ are allocated to a task. Nodes A and B both have a local I/O bandwidth of $1Gb/s$ while edge $\{A, B\}$ is annotated with a bandwidth of $10Gb/s$. We see that the computation and storage resources become the bottleneck of the task (*i.e.*, $1Gb/s$), and the communication bandwidth (*i.e.*, $10Gb/s$) will be at most utilized by $1/10 = 10\%$.



(a) The resource graph representation shows that $\{bw(s_1, d_1) \leq 100Mb/s, bw(s_2, d_2) \leq 100Mb/s\}$.



(b) The physical topology shows that the paths of (s_1, d_1) and (s_2, d_2) share bottleneck links, *i.e.*, $bw(s_1, d_1) + bw(s_2, d_2) \leq 100Mb/s$.

Fig. 3. An example to demonstrate the inefficiency of resource graph representation.

Thirdly, the graph representation abstracts each resource into a single node or edge regardless of the shared resource between nodes or edges in the physical topology. This would lead to over-provisioning of resource and jeopardize the performance of analytic workflows. Consider the example in Figure 3. The resource graph in Figure 3a) contains two sets of resources: $\{s_1, (s_1, d_1), d_1\}$ and $\{s_2, (s_2, d_2), d_2\}$. The available bandwidth on edges (s_1, d_1) and (s_2, d_2) are both $100Mb/s$ and for simplicity, we assume the I/O bandwidths of all end hosts are larger than $100Mb/s$. However, in the physical topology shown in Figure 3b), the communication between two sets of nodes shares resources on links l_3 and l_4. Hence, the actual networking resource for the two sets of physical nodes must satisfy the constraint $bw(s_1, d_1) + bw(s_2, d_2) \leq 100Mb/s$. This key constraint cannot be expressed using the variable annotation in a resource availability graph.

The fundamental cause to these drawbacks is that resource availability graph lacks the ability to represent the feasibility and constraints related to resource sharing. To address this problem, we propose a new resource discovery and representation system called RSDP.

B. RSDP: Autonomous, Privacy-Preserving Resource Discovery and Representation

To address the aforementioned limitations of resource availability graph, we design a new system called RSDP for resource discovery and representation over a large set of

heterogeneous distributively owned resources. In particular, we provide a complete view of resource availability at each site using a novel abstraction called resource vector abstraction.

Basic idea. The design principle of resource vector abstraction is simple yet powerful: instead of abstracting resources into a single node or edge out of the physical topology, we *keep the physical topology and use a set of linear constraints to represent the feasibility and constraints of resource availability and sharing*.

Resource vector abstraction. Given a set of data-intensive analytic tasks T, a set of physical resources R (*i.e.*, computation, storage and networking) and a series of attributes $r.P$ for each resource $r \in R$, we use $C(r,p)$ to denote the capacity of resource r in providing attribute p and use $c(t,r,p)$ to denote the usage of the attribute $p \in r.P$ of resource $r \in R$ by task $t \in T$, the resource availability of this set of physical resources R over the set of tasks T can be expressed as:

$$\begin{array}{ll} \sum_{t\in T} c(t,r,p) \le C(r,p), & \forall r \in R, p \in r.P. \\ c(t,r_i,p) = c(t,r_j,p) & \forall r_i, r_j \in R, \text{ given } (t,p) \end{array} \quad (1)$$

In addition to the generic representation, another benefit of the resource vector abstraction is that the site-dependent policies can be naturally mapped into additional linear constraints. This is because a site-dependent policy p_s is typically represented as a set of tasks/flows that (1) can or cannot use a certain resource; (2) cannot exceed an upper bound of a certain resource; or (3) will be guaranteed a lower bound of a certain resource. All such policies can be expressed in the form of linear constraints.

Example. To illustrate how resource vector abstraction represents the resource availability, we revisit the physical topology in Figure 3b). Assume there are 2 tasks t_1, t_2 and we focus on the bandwidth attribute of networking resource in the set of links L, consisting of l_1 to l_{12}. We first follow the definition in Equation (1) and represent the resource availability of link bandwidth as:

$$\begin{array}{ll} c(t_1,l_i,bw) + c((t_2,l_i,bw) \le 100Mb/s, & \forall l_i \in L. \\ c(t_1,l_i,bw) = c(t_1,l_j,bw), & \forall l_i, l_j \in L. \\ c(t_2,l_i,bw) = c(t_2,l_j,bw), & \forall l_i, l_j \in L. \end{array} \quad (2)$$

Assume the policies in this site enforces that (1) t_1 must use computation/storage resources on s_1 and d_1; (2) t_2 must use computation/storage resources on s_2 and d_2; and (3) the traffic of (s_1,d_1) and (s_2,d_2) must follow the predefined routes. After combining these policies with the constraints in Equation (2), we get:

$$\begin{array}{ll} c(t_1,bw) \le 100Mb/s & \forall l_i \in \{l_1,l_2,l_5,l_6\}, \\ c(t_2,bw) \le 100Mb/s & \forall l_i \in \{l_7,l_8,l_{11},l_{12}\}, \\ c(t_1,bw) + c(t_2,bw) \le 100Mb/s & \forall l_i \in \{l_3,l_4\}, \\ c(t_1,bw) + c(t_2,bw) = 0 & \forall l_i \in \{l_9,l_{10}\}, \end{array} \quad (3)$$

which is a set of linear constraints that provides accurate, complete information about resource availability in this site.

Computing minimal, equivalent resource state abstraction. The representation of resource availability specified in Equation (1) and site policies is accurate and complete, but may result in a large set of linear constraints with redundant information. Directly sending them back to the querying party would introduce a large communication overhead and expose private information about each site, *e.g.*, site policies and topology. To address the efficiency, privacy and security concerns, we develop a lightweight, optimal algorithm in RSDP to compress the original large set of linear constraints into a minimal, equivalent set of linear constraints, which has the same feasible region as the original set but with a much smaller number of constraints. The basis of this compression algorithm is simple: given an original set of linear constraints $C : \mathbf{Ax} \le \mathbf{b}$, we iteratively select one constraint $c \in C : \mathbf{a}^T\mathbf{x} \le b$ and calculate the optimal solution of problem $y \leftarrow \max \mathbf{a}^T\mathbf{x}$, subject to, $C - \{c\}$. If b is smaller than the resulting y, c is an indispensable constraint in determining the feasible region and will be put into the minimal, equivalent constraint set C'. Otherwise, c is a redundant constraint. We prove the optimality of this algorithm via contradiction.

Applying the algorithm above on the original set of linear constraints in Equation (3) with 12 constraints, RSDP can compute and get the minimal, equivalent set of constraints C' with only 1 constraint: $\{(t1,bw) + bw(t2,bw) \le 100Mb/s\}$, achieving a compression ratio of $\frac{1}{12}$.

Schedulability. RSDP provides an accurate view of resource availability while allowing resource owners to make and practice their own policies with minimal exposure of private information. One important remaining question is whether RSDP provides full a schedulability of resources for a logically centralized orchestrator. We answer this question with the following theorem.

Theorem 1: When the resources represented by the resource vector abstraction satisfies one of the following conditions:

1) resources represented in the original set of constraints C can be fully controlled on the edge, *i.e.*, all the attributes of each resource can be controlled at end host;
2) all the attributes computed in C', the minimal, equivalent resource vector abstraction, can be fully controlled on the edge;

RSDP provides a full schedulability of resources to a logically centralized resource orchestrator.

Proof: The proof of this theorem is straightforward. Condition 1 requires that all the resources and their attributes can be controlled on the edge for orchestration purposes. Because resource vector abstraction encodes all resource attributes in the original set of constraints C, it provides a complete view of resource availability so that the orchestrator can control them on the edge. For instance, if the sending rate of each end host can be controlled by end host rate limiting, the bandwidth usage of each end host, therefore, can be controlled by the orchestrator to achieve efficient bandwidth utilization. On the contrary, if TCP is used to perform window-based congestion control, the control functionality of bandwidth allocation is given to TCP and the orchestrator cannot allocate bandwidth via the representation provided by resource vector abstraction. Condition 2 relaxes condition 1 by only requiring the attributes left in the minimal, equivalent resource vector abstraction C' to be controllable. Because of the equivalence between C and C', satisfying condition 2 ensures that the orchestrator can achieve a full schedulability of resources through the resource vector abstraction representation. ■

C. Generalization of Resource Vector Abstraction

resource vector abstraction is a powerful abstraction for resource availability. However, several issues must be addressed to apply it in the general case. We describe these issues,

propose potential solutions, and discuss practical concerns for production deployment of RSDP.

Inter-attribute Correlation. The first limitation of resource vector abstraction is that the availability of different resources is only coupled through common resource attributes, *e.g.*, bandwidth of storage and networking resources. But in practice, different resource attributes can have impacts on others too. For example, the block size of storage resources will affect the data transferring time between computation resources and storage resources. Such correlation between different resource attributes is not encoded in the current design of resource vector abstraction.

To address this issue, we can add the following constraints in the definition in Equation 1.

$$f(T, R, P) \leq 0, \tag{4}$$

which are a set of functions modeling the impact between different resource attributes. There are two challenging problems: (1) formulations of f depend on specific resource attributes and are often unknown; (2) whether it is possible to eliminate the redundancy between different f to get a minimal, equivalent resource vector abstraction depends on certain properties of f (*e.g.*, convex, linear or concave). Learning techniques can be applied to cope with these challenges and is part of the ongoing efforts in the Unicorn framework.

Coexistence of unschedulable resources. As Theorem 1 states, the orchestrator on top of RSDP cannot achieve full schedulability of resources via the current resource vector abstraction design, when some resources are not controlled at the edge. For example, if the site uses TCP for congestion control instead of end-host rate limiting, bandwidth of each flow is decided by TCP via congestion signaling, *e.g.*, packet loss. The resulted packet-level rate fluctuation would prevent an accurate prediction on the per-flow achievable bandwidth in a network containing many TCP flows.

We use a black-box approach to predict the converged resource availability of unschedulable resources. For instance, by applying the Newton-Exact-Diagonal (NED) method in network utilization maximization [13], it is possible to quickly compute the converged rate of flows. This approach adds the following constraint to resource vector abstraction:

$$g(t, R, P) \leq a, \tag{5}$$

where the resource availability of unschedulable resources is predicted as a scalar a. Again, it also requires certain learning techniques to accurately predict the performance of unschedulable resources and is part of the ongoing efforts of the Unicorn framework.

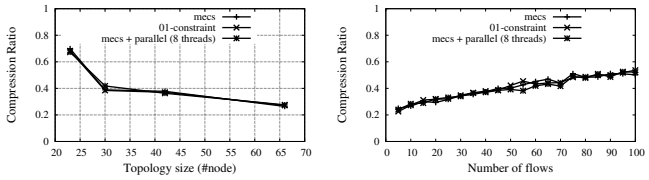
In addition, the generalization of resource vector abstraction also requires consideration on the correlation between task workflows and resource availability, the trade-off between maximizing short-term resource utilization and the long-term stability of resource availability, etc. We leave such topics as future work.

Practical issues for production deployment of RSDP. Other issues also arise when RSDP is deployed in production. First, the current design of RSDP requires a logically centralized controller to query resource availability at each site for a given set of analytics tasks. This design may not scale if the whole computing system involves many globally distributed sites and have heavy analytics workflows. Potential solutions to improve the scalability of RSDP include (1) using hierarchical organized controllers; (2) leveraging the repetitive property of analytics jobs to selectively query resource availability for only a small set of tasks instead of all of them; and (3) applying machine learning techniques to predict resource availability.

The second issue is whether site/resource owners should have total autonomy. This is related to the specific form of contracts between sites participating in collaborative computing. If all sites agree on partial autonomy, a resource enforcer module needs to be deployed together with RSDP to ensure that each site provides the amount of resources specified in contracts. The third issue is how to enforce global scheduling policies, *e.g.*, user/group/virtual organization priorities, etc., when querying for resource availability information. Solutions to this issue are overlapping with those of the previous two issues and we leave them as future work.

IV. Performance Evaluation

A prototype of RSDP has been implemented and we present key evaluation results to demonstrate its efficiency and efficacy in providing an autonomous, privacy-preserving resource representation. Without loss of generality, we focus on the bandwidth attribute of networking resources in our evaluation. We select 10 physical topologies from the topology zoo [14] with the number of nodes ranging from 13 to 117. We vary the number of tasks in the evaluation to range from 5 to 100.



(a) Compression ratio under different sizes of physical topology.

(b) Compression ratio under different numbers of tasks.

Fig. 4. Constraint compression ratio of RSDP.

We run different versions of the constraint compression algorithm proposed in Section III-B, including the regular version, a parallelized version and a modified version leveraging the fact that $c(t, r, p)$ always has a coefficient of 0 or 1 to speed up the compression. We present the compression ratio of RSDP in Figure 4. It can be observed that all three versions of the constraint compression algorithm produce the same compression ratio under all settings. Figure 4a) shows that the compression ratio of RSDP decreases as the topology size grows. This is because with more networking resource entities (*i.e.*, links) available, the chance for different tasks to share the same resource entities decreases. Figure 4b) shows that when the topology size is fixed, the compression ratio grows as the number of tasks grows. Even so, we observe that RSDP can still compress more than 40% of the original resource state, achieving a compression ratio less than 0.6. These results demonstrate the efficiency and scalability of RSDP in providing autonomous, privacy-preserving resource representation. We also measure the computation overhead of RSDP and observe that RSDP using a parallel compression algorithm has a very low computation delay, *e.g.*, $<$100ms, even for a combination of a large topology and a large set of tasks. We omit other evaluation results due to the page limit.

V. Related Work

The fundamental challenge of resource management for multi-organizational, geo-distributed, data-intensive collaborative computing is to accurately discover and represent resource availability in a large set of distributively owned heterogeneous resources. There exists a rich literature in the field of resource management of cluster computing [2]–[11]. Most of these studies focus on managing resources of a single cluster/data center. YARN [4] is the core resource management framework of Hadoop. Mesos [3] is a platform designed to share resources among multiple cluster computing frameworks, *e.g.*, MapReduce [15], Spark [16], MPI and etc. Google designs a system called Borg [5] to orchestrate the cluster resources for its proprietary data analytics frameworks. Microsoft (*i.e.*, Apollo [6]) and Facebook (*i.e.*, Corona [7]) also develop similar systems tailored to their data analytics needs. HTCondor [2] proposes a ClassAds programming model, which allows different resource owners to advertise their resource supply and the job owners to advertise the resource demand. The CMS [1] experiment at CERN uses HTCondor and glideinWMS [8] to manage a set of distributively owned computing resources in a globally distributed system.

The common settings of these systems, *i.e.*single-domain (except for HTCondor) and no networking bottleneck, usually lead to easier designs for resource discovery and representation. In particular, a graph representation is adopted by current systems to represent available resources. However, in multi-organizational, data-intensive collaborative computing, this design suffers from race conditions between resource suppliers consumers. What is worse, with the recent observation that computation, storage and networking resources have approximately the same probability to become the bottleneck affecting the performance of data-intensive analytics jobs [12], such a graph representation would lead to inefficient use of resources and resource over-provision. On the contrary, the RSDP system in Unicorn addresses these drawbacks by using a set of linear constraints to represent the feasibility and constraints of resource availability and sharing.

Another line of work called geo-distributed data analytics is also related to Unicorn. Solutions in this field include (1) moving the input dataset to a single data center before the computation [17], [18] and (2) placing different amounts of tasks at different sites depending on dataset availability to achieve a better parallelization and hence a lower latency [9]–[11]. The main focus of these solutions is to optimize the usage of a set of dedicated networking resources. This simplified setting is different from that of Unicorn, where different types of resources owned by different owners need to be orchestrated for data-intensive collaborative computing.

VI. Conclusion and Future Work

Multi-organizational, geo-distributed, data-intensive collaborative computing calls for a framework to manage a large set of distributively owned heterogeneous resources, with the fundamental objective of efficient resource utilization, following the autonomy and privacy of resource owners. In this paper, we propose Unicorn, the first unified framework that accomplishes this goal. The foundation of Unicorn is RSDP, an automatic, privacy-preserving resource discovery and representation system which describes the resource availability using a set of linear constraints. In addition, Unicorn also provides an analytics demand automation system, *i.e.*, Handyman, and an efficient, scalable multi-resource orchestrator, *i.e.*, Miro. We have implemented a prototype of Unicorn and performed a preliminary evaluation. For future work, we plan to evaluate the performance and scalability of Unicorn more extensively before moving to production deployment.

Acknowledgement

We thank Justas Balcas, Shiwei Chen, Lili Liu, Maria Spiropulu and Jean-Roch Vlimant for helpful discussion during the work. The Yale team was supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; International Technology Alliance Agreement No W911NF-16-3-0002; Google Research Award, SDN Programming Using Just Minimal Abstractions; NSFC #61672385, FAST Magellan. The Tongji team was supported by China Postdoctoral Science Foundation #2017M611618. The Caltech team was supported in part by DOE/ASCR project #000219898, SDN NGenIA; DOE award #DE-AC02-07CH11359, SENSE, FNAL PO #626507; NSF award #1246133, ANES; NSF award #1341024, CHOPIN.

References

[1] T. C. Collaboration, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, 2008.

[2] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency and computation: practice and experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *NSDI*, 2011.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *SoCC*. ACM, 2013, p. 5.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *EuroSys*. ACM, 2015, p. 18.

[6] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *OSDI*, 2014, pp. 285–300.

[7] "Under the hood: Scheduling MapReduce jobs more efficiently with Corona," http://on.fb.me/TxUsYN, [Online; accessed: 09-May-2017].

[8] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, "The pilot way to grid resources using glideinWMS," in *CSIE*. IEEE, 2009, pp. 428–432.

[9] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "WANalytics: Analytics for a geo-distributed data-intensive world," in *CIDR*, 2015.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-distributed Data Analytics," in *SIGCOMM*. ACM, 2015, pp. 421–434.

[11] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *SoCC*. ACM, 2015, pp. 111–124.

[12] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI, "Making sense of performance in data analytics frameworks," in *NSDI*, 2015, pp. 293–307.

[13] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet control for datacenter networks," in *NSDI*. USENIX Association, 2017.

[14] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," vol. 29, no. 9, pp. 1765–1775, 00249.

[15] D. Jeffrey and G. Sanjay, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *HotCloud'10*.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *SIGCOMM'13*.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *SIGCOMM*. ACM, 2013.

2018 IEEE 38th International Conference on Distributed Computing Systems

DDP: Distributed Network Updates in SDN

Geng Li[†*], Yichen Qian[†], Chenxingyu Zhao[‡], Y.Richard Yang[*], Tong Yang[‡]
[†]Tongji University, China, [*]Yale University, USA, [‡]Peking University, China

***Abstract*—How to quickly and consistently update a network is among the most fundamental and common challenges in software defined networking (SDN) systems. Current approaches heavily rely on the (logically) centralized controller to initiate and orchestrate the network updates, resulting in long latency of update completion. In this paper, we present DDP, a system for fast, distributed network updates while preserving various consistency properties. The key technique in DDP is a novel primitive named datapath operation container (DOC), where each DOC is encoded with an individual operation and its dependency logic. DDP adopts the simple, but powerful DOCs to configure the network, so that network updates can be triggered and executed at the data plane in a distributed and local manner. Novel algorithms are designed to compute and optimize the DOCs for consistent updates. We implement DDP to evaluate its performance in various update scenarios. Experimental results show that DDP significantly improves network update speed by up to 52.1% for the real-time updates initiated by the controller, and further improves the speed by 55.6-61.4% for the updates directly triggered at the data plane, such as failure recovery.**

I. Introduction

Software-defined networking (SDN) is considered as a major recent advance in networking [1], [2]. It significantly simplifies network management and provides real-time network programmability by decoupling the network control plane from the data plane. Network updates are among the most common data plane operations, either periodically or triggered by local events such as failures. However, quickly and consistently updating the distributed data plane poses a major and common challenge in SDN systems [3], [4]. Specifically, due to asynchronous communication channels, control messages are often received and executed by switches in an order different from the order sent by the controller. An inappropriate control order may violate the consistency properties on the datapath, resulting in network anomalies, such as blackholes, traffic loops and congestion [5]–[8].

The consistent network update problem has been widely studied in the literature [3]–[7]. However, all of the work is based on centralized initiation and orchestration to perform the updates. An update can be launched only by the control plane, where the controller decides an order in which operations must be applied. The coordination of the distributed data plane requires frequent communication between the controller and switches, which slows down the update completion time, and increases the controller's processing load. In addition, centralized updates rely on the control plane too heavily. When the controller becomes a bottleneck, the network may suffer from substantial performance and reliability degradation.

In this paper, we present Distributed Datapath (DDP), a system for fast, distributed network updates in SDN, while maintaining various consistency properties. DDP still benefits from centralized intelligence at the control plane, but develops distributed coordination abilities at the data plane. The key technique in DDP is a simple, but powerful primitive named datapath operation container (DOC), where each DOC is encoded with an individual operation and its dependency logic. For real-time updates initiated by the controller, the involved DOCs are sent to the data plane in one shot, and the switches can consistently execute them in a distributed manner. For updates directly triggered by local events, the controller prestores the DOCs at the data plane, and when corresponding events happen, the updates can be locally triggered and executed. We further design novel algorithms to compute and optimize the primitive DOCs for consistent updates.

We fully implement the DDP system to evaluate its performance in various update scenarios. The results demonstrate that compared to state-of-the-art centralized approaches (*e.g.*, Dionysus [5]), DDP improves real-time network update speed by 31.4-52.1%. Furthermore, we show that DDP can locally initiate updates triggered by link failures, and is up to 61.4% faster than centralized approaches to recovering routing.

II. Network Update and Motivation

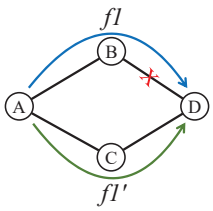
We start by formalizing the network update problem and then give an example to show the limitations of centralized approaches.

A. Network Update Problem

Our focus is on flow-based traffic management applications [2], [5], where each flow is an aggregate of packets between ingress and egress switches. We let C denote a network configuration state, which is a collection of exact match rules determining each flow's datapath. A network update is defined as a transition of configuration state from C to C'. We denote the update process as $C' = update(C, O, e)$, where $O = \{o\}$ is a set of datapath operations that implement the update. Each operation o is a modification on the data plan state, *e.g.*, to insert/delete/modify a flow rule at a particular switch. e is a local event at the data plane that triggers the update, *e.g.*, a link/switch failure and link congestion. Note that e is just used for identifying the update origin. Sometimes the update is triggered by operators or applications, and then $e = \varnothing$.

A network update can involve multiple unsynchronized devices at the data plane, so achieving the consistency is challenging during the updates. The consistency usually implies three properties: 1) *blackhole-*, 2) *loop-* and 3) *congestion-freedom*, and the detailed definitions can be found in [5], [8]. To prevent a violation of the consistency properties in any intermediate states from C to C', the datapath operations with various dependencies are constrained in a correct processing order. Assume an update $C' = update(C, O, e)$ is given, our

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00150



e_1: BD link down

o_c: insert $f1'$ at C
o_a: modify $f1$ at A
o_b: delete $f1$ at B

Fig. 1. A consistent network update example that incurs 3 rounds of controller-to-switch communication to orderly apply the operations.

work is to quickly apply the operations O after e happens, while in a correct order.

B. Motivation Example

Consider an example shown in Fig. 1. There is a flow $f1$ in the network with path ABD. Assume that the link BD happens to be down, triggering an update from $f1$ to $f1'$ with 3 operations shown in the figure. So the update can be expressed as $C' = update(C, \{o_c, o_a, o_b\}, e_1)$. To ensure consistency, the 3 operations have to be processed orderly. o_c has to be applied before o_a; otherwise the flow $f1'$ will encounter a blackhole at C where no matched rule exists. Similarly, o_b has to be applied after o_a to avoid the blackhole at B. The ordered processing can be solely coordinated by the controller in centralized update approaches: the controller sends o_c to C, waits for its processing confirmation, and then sends o_a and o_b by the same token. As a result, the simple network update incurs at least 3 rounds of communication between the controller and switches, leading to substantial extra delays.

An insight we can extract from the example is that the 3 datapath operations will be applied at adjacent switches, and if the switches themselves can coordinate with each other to orderly apply the operations, the update time as well as the controller's processing load will be greatly reduced. Further more, if we can pre-store the operations at the data plane, and the link-down event can directly trigger them to apply, then the network update will be executed in a fully local manner. In this way, the update speed will be further improved.

III. DDP Design

DDP is proposed as a system to achieve fast, distributed, consistent updates in SDN. We first explore the dependencies among datapath operations and local events to ensure the consistency properties, and such dependency information is then encapsulated in a novel primitive DOC for each operation. DDP configures the network by the simple, but powerful primitive, so that network updates can be triggered and executed at the data plane in a distributed and local manner.

A. Operation Dependency Graph (ODG)

We first introduce the concept of an Operation Dependency Graph (ODG) that captures the data plane dependencies. An ODG is a directed acyclic graph (DAG) where the nodes are the operations O and the trigger event e for an update $C' = update(C, O, e)$. The edge in an ODG reflects a timed order in a broad sense: upstream nodes have to happen before

(a) Type 1 connection **(b) Type 2 connection**

Fig. 2. Two types of connection in an ODG. (a) Type 1 connection: operation o_2 depends on the completion of operation o_1. (b) Type 2 connection: event e can trigger operation o.

downstream nodes. There are two types of connection in an ODG. The first type as in Fig. 2(a) denotes that o_2 depends on the completion of another operation o_1, while the second type as in Fig. 2(b) denotes that event e can trigger o to handle this event. Note that there is no incoming edge connected to an event e, and Type 2 connection is dispensable since e can be $\varnothing$. An ODG well describes a network update, where Type 1 connection implies the correct order in which the operations are applied to ensure consistency, and Type 2 connection identifies the update trigger.

Properties of the ODG. The ODG hold several nice properties. First, the dependency is unidirectional, resulting in no cycles in the graph. Second, the dependency relations are transitive, *e.g.*, if an event e can trigger both o_1 and o_2, and o_2 depends on o_1, then e will be connected with only o_1 whose child node is o_2. Therefore, the ODG expresses an optimized result of the whole dependency relations. Third, connectivity is dispensable in the ODG. For the operations without dependencies, they will form into isolated pieces with no connections. Lastly, the ODGs are composable. Multiple ODGs for different update events can be composed together, so that the data plane can locally handle more events in DDP. The ODG composition algorithm will be introduced in Sec. IV-B

B. Datapath Operation Container (DOC) Specification

```
    structure DOC
    {
1:      operation o
2:      boolean expression gate
3:      operation set release
    }
```

Fig. 3. Illustration of the primitive DOC.

To enable distributed and local network updates in DDP, we propose a novel primitive named DOC. The DOC is a structure as shown in Fig. 3, including 3 members as follows.

- ***o*** is an ordinary datapath operation.
- `gate` is the condition to apply o, represented by a Boolean expression of o's parent nodes in the ODG.
- `release` is a set of operations that depend on o, *i.e.*, the set of o's child nodes in the ODG.

Semantics. The semantics of DOC execution is simple. For each DOC d, 1) the inside operation $d.o$ is not applied until the gate logic d.`gate` is fully satisfied; 2) After $d.o$ is applied, all operations in d.`release` will be notified. The Boolean expression in `gate` consists of either a single operation or several operations joined by the Boolean operators AND (&) and OR (||). For example, if the execution of operation o_1 depends on

the completion of both o_2 and o_3, then d_1.gate $= o_2 \& o_3$, and d_2.release $= d_3$.release $= o_1$. The content in gate and release of each DOC is computed by the algorithm in Sec. IV-A.

In the DDP system, the SDN controller adopts DOCs to configure the data plane, rather than directly sending operations as in traditional SDN. The switches then coordinate with each other to execute the update at right time.

C. Execution Behaviors

In DDP, we define two types of execution behaviors at the data plane for every DOC: Push and Pull.

- **Push:** Upon a DOC is executed at the data plane (*i.e.*, the inside o is applied in the switch), it will send Push messages to the DOCs of all release operations. Hence the direction of Push is downward along with the ODG.
- **Pull:** Upon a DOC is received at the data plane, a Pull message will be sent to the DOC of every operation in its gate. So the Pull direction is upward. In addition, a DOC also has responsibility to Push back after receiving a Pull.

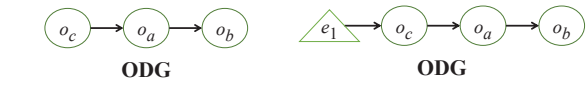
Correctness. The two behaviors guarantee the safety and liveness of distributed execution in DDP. The correctness intuition is that DOCs will be eventually executed as planned in the ODG regardless of arriving order. Suppose there are two operations with a dependency $o_1 \rightarrow o_2$, which means o_1 should be applied before o_2. The SDN controller sends out d_1 and d_2 at the same time. Case 1: d_2 arrives at the data plane first. According to the semantics, o_2 will not be applied until d_1 arrives at the data plane and sends d_2 a Push after execution. Case 2: d_1 arrives first. Then it is executed immediately and sends d_2 a Push which is useless because d_2 has yet to arrive. When d_2 arrives, it will Pull d_1 to Push back again, so that o_2 can be applied. In summary, Push and Pull are complementary to each other, and with the two behaviors, all operations will be consistently applied in a correct order. The detailed proof of the correctness can be found in our technical report [9].

D. Examples

Now we use the example in Fig. 1 to see how the network update is executed in DDP.

Real-time update. Assume after detecting the link-down event e_1, the SDN controller decides to update $f1$ to $f1'$. In this case, $e = \varnothing$ because the update is initiated by the controller. Fig. 4(a) illustrates the ODG and all the DOCs involved in this real-time update. The 3 DOCs are sent to the data plane in one shot. First, d_c is executed upon arriving at C owing to the empty gate which is satisfied naturally. Then, a Push message is sent to d_a to apply o_a at A. Lastly, B can apply o_b after receiving the Push message from d_a. Compared with the centralized update which incurs 3 round-trip delays between the remote controller and the switches, DDP needs only 2 one-way delays between adjacent switches (for coordination), therefore reduces the update completion time.

Local update. DDP can perform the update even better by pre-sending the DOCs d_a, d_b and d_c as shown in Fig. 4(b) to A, B and C respectively. Because of their non-empty gates, the three operations will not take effect on the datapath at first. But when link BD is down, and C detects this event e_1 (we assume



	operation	gate	release
d_c	o_c: insert $f1'$		o_a
d_a	o_a: modify $f1$	o_c	o_b
d_b	o_b: delete $f1$	o_a	

(a) Real-time update

	operation	gate	release
d_c	o_c: insert $f1'$	e_1	o_a
d_a	o_a: modify $f1$	o_c	o_b
d_b	o_b: delete $f1$	o_a	

(b) Local update

Fig. 4. Illustration of the ODGs and DOCs for the example in Fig. 1. (a) Real-time update which is initiated by the controller. (b) Local update which is directly triggered at the data plane.

B will flood the link-down event), d_c.gate will become true and o_c is applied accordingly. After that, d_a and d_b will also be executed sequentially. As a result, with the powerful DOCs in DDP, the network update can be both triggered and executed in a fully local manner, further improving the update speed while maintaining the consistency.

IV. Algorithms

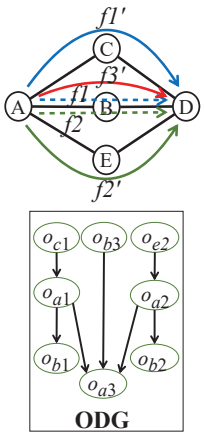
We design two algorithms in DDP: 1) computing the basic DOCs for individual updates, and 2) an optimization for multiple local updates by ODG composition.

A. Computing DOC

This algorithm computes the DOC of each datapath operation to ensure the consistency during an update. It takes the set of operations O and the event e as the input and outputs the corresponding DOCs. The algorithm consists of two steps: (1) ODG construction, and (2) Computing gate and release.

Example. To illustrate the algorithm, we provide a real-time update example in Fig. 5. Each link has a capacity of 10 units and each flow has a size of 5. The old configuration includes two flows $f1$ and $f2$ with same path ABD (labeled by dashed lines), while the updated one includes two modified flows $f1'$ with path ACD, $f2'$ with path AED and a new flow $f3'$ with path ACD (solid lines). This update is initiated by the controller, so $e = \varnothing$. We can use the 2-step algorithm in our paper to compute the corresponding DOCs.

Step 1: ODG construction. To construct an ODG, we use similar ideas of existing work on consistent updates [5], [8]. To ensure *blackhole- and loop-freedom*, 'insert' and 'modify' operations on the source switch depend on the successors on the flow route. For example, o_{a1} depends on o_{c1}, and o_{a2} depends on o_{e2} in Fig. 5. A 'delete' operation on the last switch depends on the predecessors on the route, *e.g.*, $o_{a1} \rightarrow o_{b1}$ and $o_{a2} \rightarrow o_{b2}$. To ensure *Congestion-freedom*, current remaining resources should be enough for a resource-consuming operation. Otherwise this operation will depend on resource-freeing operations on the same link to get enough resources, *e.g.*, o_{a3} (resource-consuming operation) depends on both o_{a1} and o_{a2} (resource-freeing operations). We use the same priority-criteria in [5] to schedule resource-consuming operations to avoid deadlocks. Lastly, if the update is triggered by a local event e, directional edges will be placed from e to the root operations in the ODG.



Step 1: ODG construction

	operation at A	gate	release
d_{a1}	o_{a1}: modify $f1$	o_{c1}	o_{a3}, o_{b1}
d_{a2}	o_{a2}: modify $f2$	o_{e2}	o_{a3}, o_{b2}
d_{a3}	o_{a3}: insert $f3'$	$(o_{a1}\|\|o_{a2})$ &o_{b3}	

	operation at B	gate	release
d_{b1}	o_{b1}: delete $f1$	o_{a1}	
d_{b2}	o_{b2}: delete $f2$	o_{a2}	
d_{b3}	o_{b3}: insert $f3'$		o_{a3}

	operation at C	gate	release
d_{c1}	o_{c1}: insert $f1'$		o_{a1}

	operation at D	gate	release
d_{e2}	o_{e2}: insert $f2'$		o_{a2}

Step 2: Computing gate and release

Fig. 5. An example of the algorithm computing the basic DOCs.

Step 2: Computing gate and release. Algorithm 1 is a function to compute the boolean logic in `gate` and the operation set in `release` of each operation o_i. The intuition is that by constructing the ODG in Step 1, the elements in each DOC's `gate` and `release` are determined, while in Step 2, logic operators are further inserted to obtain the final Boolean expressions in `gate`. There is no logic operator in `release`, so d_i.`release` is the operation set of o_i's children. For `gate`, the parent operations not located at the same switch as o_i are responsible for *blackhole-* and *loop-freedom*. Hence these parent operations are joined by & operators, *e.g.*, o_{b3} in d_{a3}.`gate`. Otherwise, we use another function *FindFeasibleScheduling* to compute the logic.

Specifically, *FindFeasibleScheduling* is a function to find all possible resource-freeing conditions that can make o_i scheduled. We let $o_i.flow^-$ denote the set of resource-freeing operations for o_i, which are identified as the operations on the same switch as o_i. For example, o_{a1} and o_{a2} are in $o_{a3}.flow^-$. If a resource-freeing condition is enough to schedule o_i, the combination becomes a feasible plan (f). Different feasible plans are separated by OR operators. For the example in Fig. 5, o_{a1} and o_{a2} are two feasible scheduling planes for operation o_{a3}. So d_{a3}.`gate` includes $o_{a1}\|o_{a2}$. The details of this function can be found in the technical report [9].

B. ODG Composition

As discussed earlier, an ODG corresponds to only one update. If we want the data plane to locally handle one of multiple updates $C_1 = update(C, O_1, e_1), C_2 = update(C, O_2, e_2), ...$, we need to prepare multiple ODGs. Here we assume one update is executed at a time, because all of the updates are based on the current configuration C. For the local update example in Fig. 1, if we hope the data plane can locally handle another event $e_2 = AB$ link down, then we need another ODG with a new set of DOCs. Different updates may share common operations, so an ODG composition is required to rewrite the DOCs. We let G_i denote an ODG and $\oplus$ denote the composing operator. Since the composing operator $\oplus$ is associative, *i.e.*, $(G_1 \oplus G_2) \oplus G_3 = G_1 \oplus (G_2 \oplus G_3)$, therefore the composition of arbitrary ODGs can be derived by the algorithm for composing two ODGs in Algorithm 2.

Algorithm 1 ComputeGate&Release(o_i)

```
1: for each o_j ∈ o_i.children do
2:     d_i.release ← d_i.release ∪ o_j
3: end for
4: o_i.flow⁻ ← ∅
5: for each o_k ∈ o_i.parents do
6:     if o_k and o_i at same switch then
7:         o_i.flow⁻ ← o_i.flow⁻ ∪ o_k
8:     else
9:         d_i.gate ← d_i.gate&o_k
10:    end if
11: end for
12: F ← ∅
13: for each f ∈ FindFeasibleScheduling(o_i.flow⁻) do
14:    F ← F||f
15: end for
16: d_i.gate ← d_i.gate&F
```

Algorithm 2 ODGComposition(G_1, G_2)

```
1: for each o_i1 = o_i2, o_i1 ∈ G_1, o_i2 ∈ G_2 do
2:     if d_i1.gate ≠ d_i2.gate then
3:         d_i1,i2.gate ← d_i1.gate||d_i2.gate
       // d_i1,i2 is the DOC after composing
4:     end if
5:     if d_i1.release ≠ d_i2.release then
6:         d_i1,i2.release ← d_i1.release ∪ d_i2.release
7:         P_1 ← o_i1.parents
8:         P_2 ← o_i2.parents
9:         while P_1 ⊆ P_2||P_2 ⊆ P_1 do
       // iteratively find the nearest different ancestors
10:            P_1 ← P_1.parents
11:            P_2 ← P_2.parents
12:        end while
13:        Find t_1|t_1 ∈ P_1&t_1 ∉ P_2
14:        Find t_2|t_2 ∈ P_2&t_2 ∉ P_1
       // t_1, t_2 is either an operation or an event
15:        for each o_j1 ∈ d_i1.release, o_j2 ∈ d_i2.release do
16:            d_j1.gate ← d_j1.gate&t_1
17:            d_j2.gate ← d_j2.gate&t_2
18:        end for
19:    end if
20: end for
```

In the algorithm, we combine `gates` and `releases` for the common operations in both G_1 and G_2. The `gates` are joined by an OR ($\|$) operator to make sure the common operations can be triggered in both updates. In addition, we have to distinguish the child nodes of a common operation in different graphs; otherwise all of them will be released after the common operation. To cope with this, we iteratively find the nearest different ancestors to identify the two ODGs. First, we find P_1 and P_2 as the two non-containment ancestor sets for the common operation o_{i1} (o_{i2}). Then we pick out only one item t_1 (t_2) as a representative for each set P_1 (P_2). Here t_1 and t_2 are any of the elements in the ODG, *i.e.*, either an operation or an event. At last, we add t_1 into every child's `gate` in G_1, and t_2 into every child's `gate` in G_2, with the operator &. The rewrites of `releases` for t_1 and t_2 are omitted for space constraints. As a result, for the example in Fig. 1,

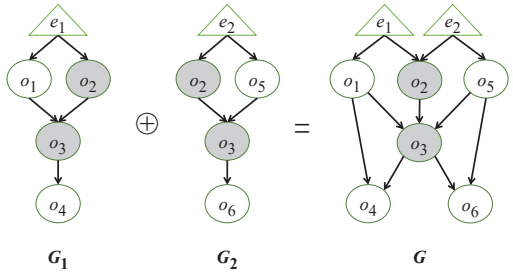


Fig. 6. An example of ODG composition where $G = G_1 \oplus G_2$.

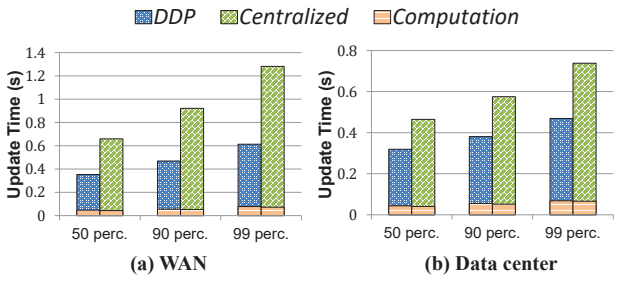


Fig. 7. Update completion time, broken down by the amount of computation and execution.

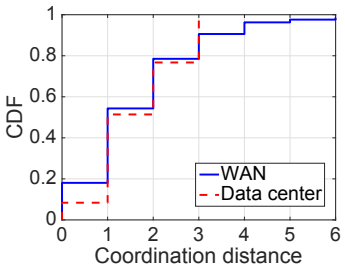


Fig. 8. CDFs of the coordination distance between DOCs, specified by the hop-count of switches between $d.o$ and $d.\texttt{release}$ (if any), *e.g.*, the distance between d_{a1} (d_{a2}) and d_{a3} in Fig. 5 is 0.

$d_c.\texttt{gate} = e_1 \| e_2$ after composition, so that no mater AB or BD down, the network can locally recover routing.

Example. We give another example in Fig. 6, where G_1 and G_2 are the ODGs for two local updates prepared in DDP (e_1 and e_2 have not happened yet). After composing, $d_2.\texttt{gate}$ becomes $e_1 \| e_2$, and $d_2.\texttt{release} = o_3$ keeps unchanged. For o_3, both the `gates` and `releases` are different in the two ODGs, so they are rewritten as $d_3.\texttt{gate} = o_2 \& (o_1 \| o_5)$ and $d_3.\texttt{release} = \{o_4, o_6\}$. In addition, $P_1 = \{o_1, o_2\}$ and $P_2 = \{o_2, o_5\}$ are found as the nearest non-containment ancestor sets, and $t_1 = o_1$ and $t_2 = o_5$ are picked out to represent P_1 and P_2 respectively. In the end, o_4 and o_6 are distinguished by $d_4.\texttt{gate} = o_1 \& o_3$ and $d_6.\texttt{gate} = o_3 \& o_5$.

V. Performance Evaluation

We fully implement the DDP system with 3000+ lines of Python code to evaluate its performance in various update scenarios. Experimental results show that DDP can significantly speed up network updates.

A. Experimental Methodology

We conduct all experiments on real topologies consisting of Open vSwitches in both WAN and data center scenarios. For WAN, we choose 5 topologies from the Topology Zoo [10] that interconnect $O(30)$ sites with link capacities between 10 and 100 $Gbps$. For data center, we emulate a 3-tier datacenter network topology with $O(60)$ switches, where each edge link is of $10Gbps$ capacity, and aggregated link is of $100Gbps$ capacity. A custom software agent is running on each switch to coordinate with each other and create execution logs. The communication protocol between the controller and the agents is implemented by an extension version of OpenFlow, and the communication between the agents is via UDP messages. In our experiments, we compare DDP against a Centralized update system based on Dionysus [5].

B. Experimental Results

Real-time updates. We measure the update completion time of 30 real-time updates in both WAN and data center scenarios. We break down the overall time by the amount of computation at the controller and the execution at the data plane. First, Fig. 7 shows that computing the updates is not a bottleneck in both schemes, and the DDP system requires a little longer computation time than Centralized because the scheduling plans are pre-computed and encoded in the DOCs.

From Fig. 7, we can observe that DDP achieves a much lower execution time than Centralized. This major gain is from the distributed manner of update execution in DDP. Centralized approach relies on the controller to orchestrating the updates, so the coordinating time is a sum of many-round of communication between the controller and switches. However in DDP, switches directly coordinate with each other at the data plane, therefore reducing the total execution time. From the CDFs shown in Fig. 8, we can see most of the coordination in DDP is within the same switch (up to 18.4%) or between adjacent switches (up to 43.0%), so it's more efficient for switches to coordinate with each other rather than communicating with the remote controller.

Overall, as shown in Fig. 7, DDP outperforms Centralized in real-time updates under both WAN and data center settings. For WAN, DDP is 46.4%, 49.1%, and 52.1% faster than Centralized in the 50^{th}, 90^{th}, and 99^{th} percentile, respectively. For data center, the corresponding numbers are 31.4%, 33.7%, and 36.4% faster than Centralized,. The WAN topologies are more complex than the data center ones, resulting in longer completion time.

Local updates. We conduct another experiment to show the benefit of DDP in local updates. We choose one configuration in each WAN topology, and pre-compute the DOCs for any link-down events. For simplicity, we consider only 1-failure case. The ODG Composition algorithm in Sec. IV-B is used to compute the final DOCs, and the DOC number is reduced by 60.3% after composition. In the experiment, we randomly fail one link and measure the recovery time for all re-routable flows in both approaches. In Centralized approach, the link-

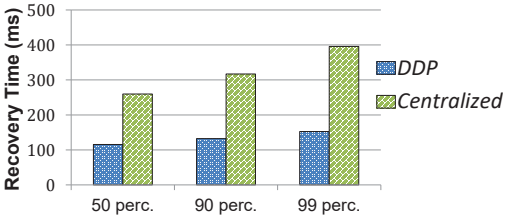


Fig. 9. Recovery time from a random link failure.

down event has to be reported to the controller, who will then compute new routes to update the network. But in DDP, when the link-down event happens, pre-stored DOCs are locally trigged at the data plane and directly take effect to recover the routing. Note that the consistency properties are preserved during all updates. As shown in Fig. 9, DDP is 55.6%, 58.2%, and 61.4% faster than Centralized in the 50^{th}, 90^{th}, and 99^{th} percentile, respectively. By locally initiating the updates at the data plane, DDP avoids reporting time and real-time computation time at the controller, therefore speeds up the updates further more.

VI. Related work

De-centralized update. ez-Segway is proposed to address the network update problem in a de-centralized manner, where switches receive partial knowledge of the network from the controller and conduct distributed computing to execute the update [8]. DDP's improvement over it is a much more powerful primitive DOC, leading to much lower overhead and computation complexity at switches, while enabling local updates. A timed update is another decentralized approach that uses synchronized clocks to coordinate the update [11]–[13]. However, due to imprecise clock synchronization and time prediction, the consistency and efficiency of network updates are not guaranteed as in DDP. Our primitive DOC can have a wide range of extension, *e.g.*, the Boolean expression in `gate` may support time variant in the future work.

Local Recovery. Prior art on failure recovery in SDN relies on Openflow local fast failover mechanisms [14]–[16]. The controller pre-installs backup rules (tunnels) in switches' group tables, so that the backups can be immediately activated upon a link failure. DDP transforms the backup rules into compact pending DOCs, therefore saves the scarce table resources without performance loss. In addition, this line of work deals with only link-down events, whereas DDP is capable of handling any happenings that can be detected by switches, such as link congestion or unbalanced load. Our novelty lies in allowing local events to directly trigger the network-wide update, which to our knowledge has not been done before.

VII. Conclusion

This paper presents DDP, a system for fast, distributed, consistent network updates in SDN. The configuration in DDP is based on a novel primitive named DOC. The DOCs can be executed by the switches following the dependencies among different datapath operations, while achieving data plane consistency. We also design two algorithms to compute and optimize the primitive DOCs respectively. Evaluation results show that DDP significantly improves network update speed in various update scenarios. Developing some high-level APIs in DDP to automatically generate the DOCs will be the next step of this research.

Acknowledgment

This research was supported in part by NSFC #61701347, NSFC #61702373, NSFC #61672385 and NSFC #61672061; NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Tong Yang is the corresponding author of this paper.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[3] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, p. 20.

[4] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 323–334, 2012.

[5] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM CCR*, vol. 44, no. 4, 2014, pp. 539–550.

[6] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," in *ACM SIGCOMM CCR*, vol. 43, no. 4, 2013, pp. 411–422.

[7] K.-T. Förster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *IFIP Networking Conference*, 2016, pp. 1–9.

[8] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in sdn," in *Proceedings of the SOSR*, 2017, pp. 21–33.

[9] "Anoymous Technical Report," https://github.com/technical-report-2018/ICDCS2018.

[10] "The Internet Topology Zoo," http://www.topology-zoo.org.

[11] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *Proceedings of the 1st ACM SIGCOMM SOSR*, 2015, p. 21.

[12] J. Zheng, G. Chen, S. Schmid, H. Dai, J. Wu, and Q. Ni, "Scheduling congestion-and loop-free network update in timed sdns," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, 2017.

[13] J. Zheng, G. Chen, S. Schmid, H. Dai, and J. Wu, "Chronus: Consistent data plane updates in timed sdns," in *Distributed Computing Systems (ICDCS), IEEE 37th International Conference on*, 2017, pp. 319–327.

[14] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, "We've got you covered: Failure recovery with backup tunnels in traffic engineering," in *Network Protocols (ICNP), International Conference on*, 2016, pp. 1–10.

[15] M. Borokhovich and S. Schmid, "How (not) to shoot in your foot with sdn local fast failover," in *International Conference On Principles Of Distributed Systems*, 2013, pp. 68–82.

[16] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms," in *Proceedings of HotSDN*, 2014, pp. 121–126.

Unicorn: Unified Resource Orchestration for Multi-Domain, Geo-Distributed Data Analytics

Qiao Xiang[+‡], X. Tony Wang[+‡], J. Jensen Zhang[+],
Harvey Newman[⋄], Y. Richard Yang[+‡], Y. Jace Liu[+],
[+]Tongji University, [‡]Yale University, [⋄]California Institute of Technology,
{qiao.xiang, xin.wang, yang.r.yang}@yale.edu, jensen@jensen-zhang.site,
newman@hep.caltech.edu, yang.jace.liu@linux.com

Abstract

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi organizational, geographically-distributed, collaborative computing, where different organizations contribute various yet scarce resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment, one of the largest scientific experiments in the world, and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of efficient resource utilization, following the autonomy and privacy of different domains, and that the fundamental challenge for designing such a framework is: *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?* Existing resource management systems are designed for single-domain clusters and cannot address this challenge. In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. In Unicorn, we encode the resource availability for each domain into resource state abstraction, a variant of the network view abstraction extended to accurately represent the availability of multiple resources with minimal information exposure using a set of linear inequalities. We then design a novel, efficient cross-domain query algorithm and a privacy-preserving resource information integration protocol to discover and integrate the accurate, minimal resource availability information for a set of data analytics jobs across different domains. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs. We discuss the implementation of Unicorn and present preliminary evaluation results to demonstrate the efficiency and efficacy of Unicorn. We will also give a full demonstration of the Unicorn system at SuperComputing 2017.

1. Introduction

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi-organizational, geographically-distributed, collaborative computing, where different organizations (also called domains) contribute various yet disaggregated resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. One important example of this paradigm is the Compact Muon Solenoid (CMS) experiment at CERN [1], one of the largest scientific experiments in the world. The CMS data analytics system is composed of over 150 participating organizations, including national laboratories, universities and other research institutes. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment over a 7-day period and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of *efficient resource utilization, following the autonomy and privacy of different domains.*

In particular, our trace analysis shows that (1) over 35% of data analytics jobs are *remote jobs*, *i.e.*, jobs that require different types of resources from different domains for execution; (2) the 90% quantile of the job execution time of remote jobs is approximately 38.9% longer than that of local jobs, *i.e.*, jobs that only require resources from a single domain for execution; and (3) the data transfer traffic is saturating the CMS network, leaving limited networking resources (*i.e.*, less than 15%) for data analytics traffic. These observations show that resources in multi-domain, geo-distributed analytics are highly disaggregated, *i.e.*, unbalanced distributed across domains. Although there is much related work on resource management for clusters and data centers, such as [2–12], they are mostly designed for managing resources in single-domain clusters, and cannot accomplish the aforementioned goal

for multi-domain, geo-distributed data analytics. In particular, these systems typically adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for two reasons. First, it *compromises the privacy of different domains* by revealing all the details of resources in each domain. Secondly, *the overhead to keep the resource availability graph up to date* is too expensive due to the heterogeneity and dynamicity of resources from different domains. Some systems such as HTCondor [2] adopts a simpler abstraction that only represents computation and storage resources in multi-domain clusters. This approach, however, leaves the orchestration of networking resources completely to the transmission control protocol (TCP), which has long been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, and hence is inefficient. Through trace analysis and related work study, we identify the fundamental design challenge for designing an orchestration framework for multi-domain, geo-distributed data analytics is: *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?*

In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo distributed data analytics. In Unicorn, the resource availability of each domain is abstracted into resource state abstraction, a variant of the network view abstraction [13] extended to accurately represent the availability of multiple resources with minimal information exposure using *a set of linear inequalities*. With this intra-domain abstraction, Unicorn uses a novel, efficient cross-domain resource discovery component to find the accurate resource availability information for a set of data analytics jobs across different domains with minimal information exposure, while allowing each domain to make and practice their own resource management strategies. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs.

This paper makes the following **main contribution**:

- we study the novel problem of resource orchestration for multi-domain, geo-distributed data analytics and identify the *cross-domain resource discovery challenge* as the fundamental design challenge for this problem through systematic trace-analysis and vigorously related work investigation;
- we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. Unicorn provides the resource state abstraction for each domain to accurately represent its resource availability with minimal information exposure in the form of a set of linear equalities, a novel, efficient cross-domain resource discovery component to provide the accurate, minimal resource availability information across different domains, and a global resource orchestrator to compute optimal resource allocations for data analytics jobs;
- we discuss the implementation details of Unicorn and perform preliminary evaluations to demonstrate the efficiency and efficacy of Unicorn. We will also present a full demonstration of Unicorn at SuperComputing 2017.

The rest of the paper is organized as follows. We analyze the data analytics trace of the CMS experiment, discuss the inadequacy of existing resource management systems and identify the key design challenge for multi-domain, geo-distributed data analytics systems in Section 2. We introduce the system setting and give an overview of the Unicorn framework in Section 3. We then present the details of two key components of Unicorn, cross-domain resource discovery and representation and global resource orchestration, in Section 4 and 5, respectively. We discuss the implementation details in Section 6 and evaluate the performance of Unicorn in Section 7. We conclude the paper and discuss the next steps of Unicorn in Section 8.

2. Motivation and Challenge

Analytics trace from the CMS experiment. We collect the trace of approximately 479 thousand data analytics jobs from the CMS experiment, one of the largest scientific experiments in the world, over a period of 7 days. From this trace, we find that over 35% of jobs consumes resources across different domains, *i.e.*, these jobs use the computation node and the storage node located at different domains which are connected by networking resources across multiple domains. We call these jobs *remote jobs*, compared with *local jobs* which only use resources within one single domain. This result indicates the *resource disaggregation* in the CMS network, *i.e.*, the unbalanced distribution of storage and computation resources. We also plot the cumulative distribution function of job execution time for this set of traces as shown in Figure 1. We observe that the 90% quantile of job execution time for remote jobs has an extra 38.9% higher latency than local jobs. In addition, we observe that the cross-domain networking resources available for data analytics are very limited because the CMS data transfer traffic is saturating the limited networking resources, *e.g.*, the cross-domain data transfer network traffic of the same 7-day period has a total amount of 8785 terabytes while the cross-domain data analytics traffic is only 1404 terabytes. This observation indicates the scarcity of networking resources available for data analytics in the CMS network. All these results demonstrate that in order to support low-latency, multi-domain, geo-distributed data analytics, it is not only necessary, but crucial to design a multi-domain resource orchestration system.

Related work. There exists a rich literature in the field of resource management of clusters [2–12]. YARN [4] is the core resource management framework of Hadoop. Mesos [3] is a platform designed to share resources among multiple cluster computing frameworks, *e.g.*, MapReduce [14], Spark [15], MPI and etc. Google designs a system called Borg [5] to orchestrate the cluster resources for its proprietary data analytics frameworks. Microsoft (*i.e.*, Apollo [6]) and Facebook (*i.e.*, Corona [7]) also develop similar systems tailored to their data analytics needs. These systems are all designed for managing resources in single-domain

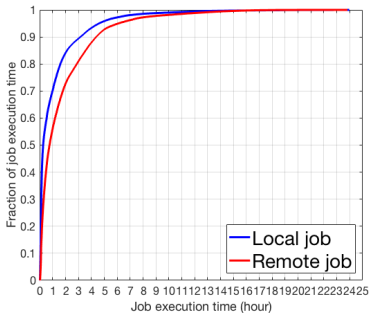


Figure 1: The CDF of job latency local and remote jobs.

clusters and adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for because (1) it compromises the privacy of different domains by revealing all the details of resources in each domain; and (2)the overhead to keep the resource availability graph up to date is too expensive due to the heterogeneity and dynamicity of resources from different domains.

There are also some efforts towards resource management for multi-domain clusters. HTCondor [2] proposes a ClassAds programming model, which allows different resource owners to advertise their resource supply and the job owners to advertise the resource demand. The CMS [1] experiment currently uses HTCondor and glideinWMS [8] to manage a set of distributively owned computing resources in a globally distributed system. These systems only focus on managing storage and computing resources in clusters, while the recent study shows that computation, storage and networking resources have approximately the same probability to become the bottleneck affecting the performance of data-intensive analytics jobs [16]. By leaving the orchestration of networking resources completely to TCP, which has been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, the abstraction adopted by these systems is also inefficient.

Another line of work called geo-distributed data analytics is also related. Solutions in this field include (1) moving the input dataset to a single data center before the computation [17, 18] and (2) placing different amounts of tasks at different sites depending on dataset availability to achieve a better parallelization and hence a lower latency [9–12]. The main focus of these solutions is to optimize the usage of a set of dedicated networking resources. The design of these systems cannot be applied to multi-domain, geo-distributed data analytics where different types of resources owned by different owners need to be orchestrated.

Design challenge. The discussion above shows the urgent need for an efficient resource orchestration framework to support multi-domain, geo-distributed data analytics systems such as CMS. And by investigating the limitations of existing resource management systems, we identify the key design challenge for such a framework is *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?* To this end, we design the Unicorn framework to manage a large set of distributively-owned, heterogeneous resources for multi-domain, geo-distributed data analytics systems. Unicorn achieves efficient resource utilization while allowing the autonomy and privacy of different domains through a novel resource state abstraction, an efficient cross-domain discovery and representation component and a global resource orchestration component, which will be discussed in the next few sections.

3. Overview

In this section, we introduce the system setting for multi-domain, geo-distributed data analytics and give an overview of the Unicorn framework and its workflow.

System settings. We consider a data analytics system composed of multiple organizations (domains). Each domain contributes a certain amount of computation, storage and networking resources for all the users in the system to store, transfer and analyze large-volume datasets. The storage and computation resources are typically physical servers, virtual machines or containers. The networking resources are typically switches and links. Domains that only contribute networking resources are called *transmission domains* and domains that also contribute computation and storage resources are called *leaf domains*. Figure 2 gives an example of such a system. In this example, domain A, B, E and F are all leaf domains while domain C and D are transmission domains.

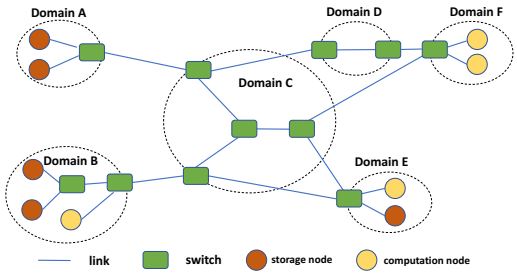


Figure 2: An example of multi-domain, geo-distributed data analytics system. Domains A, B, E and F are **leaf domains**. Domains C and D are **transmission domains.**

A data analytics task is typically decomposed into a set of jobs J whose precedence relation is specified by a directed acyclic graph (DAG). A task is finished if and only if the last job in the decomposed DAG is finished. Each job j has requirements on storage and computation resources, *e.g.*, number of CPUs, size of memory, input dataset and etc. We use $(stg, comp)$ to denote a pair of candidate storage and computation resources satisfying the requirement of j. The orchestration system is in charge of selecting one $(stg, comp)$ pair for each job j and allocating the selected storage and computation resources and the networking resources connecting them for executing j.

Unicorn architecture. We present the architecture of Unicorn in Figure 3. On top of all the domains, Unicorn

provides a logically centralized controller to orchestrate resources for data analytics jobs. This controller includes a cross-domain resource representation and discovery component and a global resource orchestration component. Residing in each domain are a domain resource manager and a set of job execution agents.

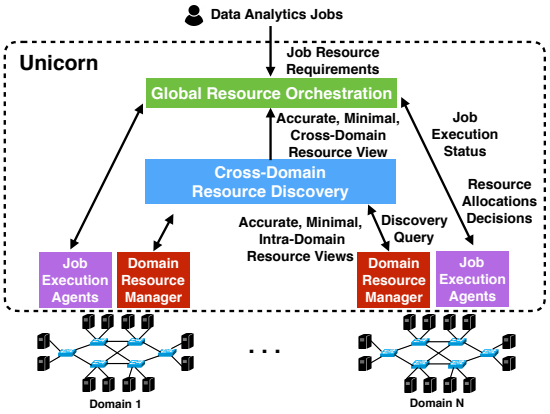


Figure 3: The architecture of Unicorn.

Unicorn provides a novel abstraction called resource state abstraction, a variant of network view abstraction [13]. This abstraction uses a set of linear inequalities to accurately represent the availability of different resources in each domain with minimal information exposure. When a set of data analytics jobs J are submitted to the Unicorn controller, the cross-domain resource discovery and integration component issues discovery queries, *i.e.*, path queries and resource queries, to the domain resource manager at each domain to retrieve the intra-domain resource view of each domain encoded in the resource state abstraction. It then assembles and compresses the responses into an accurate, minimal cross-domain resource view. This view, together with the resource requirements of j, is then used by the global orchestration component to make global, optimal resource allocation decisions and send to the job execution agents at corresponding domains. The execution agents enforce the received decisions, *e.g.*, starting the corresponding program, rate limiting the data accessing bandwidth and etc., and send the job execution status back to the Unicorn controller as feedback. In the next few sections, we present the design details of key components of Unicorn.

4. Cross-Domain Resource Discovery and Representation

In this section, we present our design to address the fundamental challenge of accurately discovering and representing a large set of distributively-owned, heterogeneously resources with minimal information exposure of resource owners. In particular, we introduce a novel abstraction to represent intra-domain resource availability and design an efficient discovery mechanism to discovery resource availability across different domains.

4.1. Intra-Domain Resource State Abstraction

Basic idea. Unicorn framework provides an abstraction called resource state abstraction to accurately represent the availability of multiple resources for a set of data analytics jobs using *a set of linear inequalities*. This is a variant of the network view abstraction [13]. In particular, we consider a set of data analytic jobs J that wants to consume a set of physical resources R (*i.e.*, computation, storage and networking) based on a set of pre-defined policies P. If a resource attribute $attr$ is *capacity-bounded*, *i.e.*, a resource r can only provide this attribute with a certain capacity (denoted as $C^{r,attr}$) and each job j consuming r can only get a portion of this attribute (denoted as $c_j^{r,attr}$), the resource availability of R for J on this attribute can be expressed as:

$$\sum_{j \in J(P,r)} c_j^{r,attr} \leq C^{r,attr}, \forall r \in R, \tag{1a}$$

$$c_j^{R,attr} = f(P, attr, c_j^{r,attr}), \forall (j, r \in R), \tag{1b}$$

$$c_j^{r,attr} = g(P, attr, c_j^{r',attr}), \forall (j, r \in R, r' \in R \ \{r\}). \tag{1c}$$

In this representation. Equation (1a) indicates that the total amount of $attr$ of resource r consumed by all the jobs cannot exceed the supply capacity of r on $attr$, where $J(P, r)$ is the set of jobs that are allowed to consume j based on the policy set P. Equation (1b) represents the total capacity of $attr$ that j can get from the whole set of resources R (denoted as $c_j^{R,attr}$) by a pre-defined linear function of $c_j^{r,attr}$, whose form depends on $attr$ and P. Equation (1c) represents the relation between the amount of $attr$ a job j can get from two resources r and r' by a pre-defined linear function, whose form depends on $attr$ and P. One of the most common capacity-bounded resource attributes is bandwidth.

If a resource attribute $attr$ is *capacity-free*, *i.e.*, each j consuming r who provides this attributes can get the same capacity $C^{r,attr}$ at the same time, the resource availability of R for J on this attribute can be expressed as:

$$c_j^{R,attr} = h(P, R, attr, j), \forall j \in J, \tag{2}$$

where the value of $c_j^{R,attr}$ is computed by a pre-defined function $h(P, R, attr, j)$ whose form depends on $attr$ and P. Note that this function does not need to be linear because the value of the right-hand side can be directly computed in this availability representation. Examples of such capacity-free resource attributes include propagation delay, hop-count, and etc.

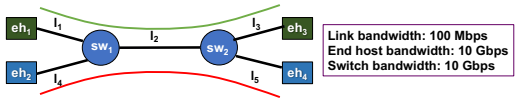


Figure 4: An example to illustrate the resource state abstraction.

Example. We use the physical topology in Figure 4 to illustrate how resource state abstraction works. Suppose two jobs j_1 and j_2 need to read data from storage node eh_1 to computation node eh_3 and from eh_2 to eh_4, respectively. The routing policy for the data flow of each job is also shown in the figure. For simplicity, we only focus on the bandwidth attribute for each resource, *i.e.*, end host,

switch and link. Following the definition in Equation (1), the resource availability of this topology for j_1 and j_2 can be expressed as:

$$\begin{array}{rl}
c_{j_1}^{l_i} \leq 100Mbps, & i = 1,3, \\
c_{j_2}^{l_i} \leq 100Mbps, & i = 4,5, \\
c_{j_1}^{l_i} + c_{j_2}^{l_i} \leq 100Mbps, & i = 2 \\
c_{j_1}^{sw_k} + c_{j_2}^{sw_k} \leq 10Gbps, & k = 1,2 \\
c_{j_1}^{eh_m} \leq 10Gbps, & m = 1,3, \\
c_{j_2}^{eh_m} \leq 10Gbps, & m = 2,4, \\
c_{j_1}^{R} = c_{j_1}^{l_i} = c_{j_1}^{sw_k} = c_{j_1}^{eh_m}, & i = \{1,2,3\}, \forall j, m = \{1,3\}, \\
c_{j_2}^{R} = c_{j_2}^{l_i} = c_{j_2}^{sw_k} = c_{j_2}^{eh_m}, & i = \{2,4,5\}, \forall j, m = \{2,4\}, \\
c_{j_1}^{l_i} = c_{j_1}^{eh_m} = 0, & i = \{4,5\}, m = \{2,4\}, \\
c_{j_2}^{l_i} = c_{j_2}^{eh_m} = 0, & i = \{1,3\}, m = \{1,3\},
\end{array} \tag{3}$$

Computing minimal, equivalent resource state abstraction. The representation of resource availability defined in Equations (1)(2) is accurate and complete, but may result in a large set of linear inequalities with redundant information. In a simple topology in our illustration example, there are already over 20 inequalities. Directly sharing them with a centralized controller or other domains would introduce a large communication overhead and expose unnecessary private information about each domain, *e.g.*, domain topology and policies. To minimize the resource information exposure of a domain, the domain resource manager of Unicorn adopts a lightweight, optimal algorithm to compress the original set of linear inequalities into a minimal, equivalent set of linear inequalities, which has the same feasible region as the original set but with a much smaller number of constraints. The basis of this compression algorithm is simple: given an original set of linear inequalities $C : \mathbf{Ax} \leq \mathbf{b}$, we iteratively select one constraint $c \in C : \mathbf{a}^T\mathbf{x} \leq b$ and calculate the optimal solution of problem $y \leftarrow \max \mathbf{a}^T\mathbf{x}$, subject to, $C - \{c\}$. If b is smaller than the resulting y, c is an indispensable constraint in determining the feasible region and will be put into the minimal, equivalent constraint set C'. Otherwise, c is a redundant constraint. The optimality of this algorithm can be proved via contradiction. Applying this algorithm to the example above, we may find that the minimal, equivalent set of linear inequalities has only one inequality: $c_{j_1}^R + c_{j_2}^R \leq 100Mbps$. This reduction from over 20 inequalities to only one shows the power of our optimal compression algorithm.

4.2. Cross-Domain Resource Discovery

The resource state abstraction allows each domain to represent the accurate resource availability for a set of data analytics jobs using a set of linear inequalities with minimal information exposure, but it still requires the knowledge of all available computation, storage and networking resources, *i.e.*, the domain topology, and the domain policy to construct the original abstraction. As a result, it is non-trivial to extend it for resource discovery cross-domains, when a job needs to consume resources located in different domains, *e.g.*, the storage node and computation node assigned to the same job may be located in two different domains and are connected by network links across multiple domains. This is because information such as domain topology and policy is usually private to each domain itself and is not allowed to be passed around different domains. In this subsection, we present the details of our design to tackle this challenge and extend resource state abstraction for cross-domain resource discovery.

Basic idea. The key insight of our design is simple yet powerful: if we can "chop" the networking resources connecting a $(str, comp)$ candidate pair for job j based on the domains they belong to, as shown in Figure 5, we can then ask the domain resource manager of each domain to compute and represent the resource availability for j in each domain independently.

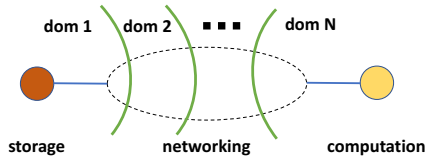


Figure 5: Chop the networking resources by domain.

With this insight, we design the cross-domain resource discovery process of Unicorn whose workflow is shown in Figure 6. In particular, Unicorn performs cross-domain resource discovery for a set of candidate $(stg, comp)$ pairs for a set of job J in four key steps. The first step is the path query process, in which the Unicorn controller issues path queries to the domain resource manager to recursively get a *domain path* in the form of

$$\begin{aligned}(dom_1, srcIP, egress) \rightarrow (dom_2, ingress, egress) \\ \rightarrow \ldots, (dom_N, ingress, dstIP),\end{aligned} \tag{4}$$

for each candidate $(storage, computation)$ node pair. The path query can be executed either recursively or iteratively. The second step is the "chopping" process, which transforms the domain paths for all the $(stg, comp)$ candidate pairs, into a set of segments, *i.e.*, the chopping results, with the form of

$$(dom_i, F_i, F_i.ingress, F_i.egress), \tag{5}$$

for each domain, where F_i denotes the set of all $(stg, comp)$ candidate pairs whose connection use the network resource in domain i. Thirdly, the Unicorn controller sends each chopped segment to the corresponding domain resource manager to issue one resource query for each segment, which asks each domain to compute the minimal, equivalent single-domain resource state abstraction. Fourthly, a privacy-preserving resource information integration protocol will be executed between all the domains to compute the accurate, minimal cross-domain resource view representing the cross-domain resource availability for a set of candidate $(stg, comp)$ pairs for a set of job J.

Path query. We present the pseudocode of the path query process in Algorithm 1. The path query is a recursive query process. In particular, the path query algorithm requires the input of *domain*, which domain the query should be sent to, F, a set of $(stg, comp)$ candidate pairs whose connection use the network resource in *domain*, and $Ingress$, the set of ingress points each candidate pair is entering *domain* from. It starts from the Unicorn controller group the whole set of F into multiple disjoint subsets based on where the storage resources for this subset of pairs are located, and send one path query for each subset to each corresponding domain. When a domain resource manager receives such a query, it first computes the egress point, the next domain, and the ingress

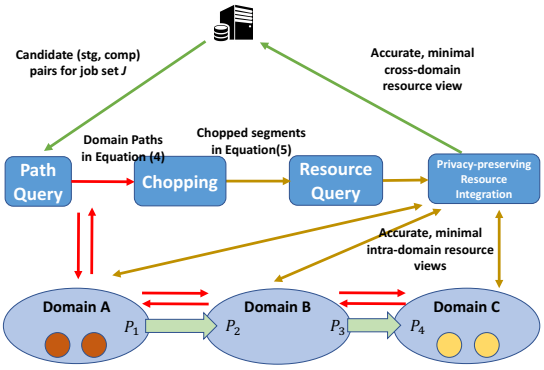


Figure 6: Workflow of cross-domain resource discovery.

point of next domain for each candidate pair f (Line 3-4). Then the set F is grouped into several disjoint subsets based on the next domain of each pair f (Line 5). For each subset F_i whose next domain is not null, the current resource manager adds the current domain into the domain path for F_i and issues another path query to the domain resource manager at $F_i.nextDom$ to get the remaining part of the whole domain path (Line 8-12). If the next domain of F_i is null, it means that the computation resources of these $(stg, comp)$ pairs are in the current domain, *i.e.*, the domain path reaches the destination, and the domain manager simply returns such information to the querying party. During the path query process, each domain only provides the egress points, the next domains and the ingress points for $(stg, comp)$ candidate pairs without revealing any topology or policy information.

Algorithm 1: The algorithm of path query.

```
1  Function domPathQuery(domain, F, Ingress)
2      domPathResponse ← ∅;
3      foreach f ∈ F do
4          (f.egress, f.nextDom, f.nextDomIngress) ←
           getNextDomain(f);
5      {F_1, F_2, ...,} ← F.groupBy(f.nextDom);
6      foreach F_i do
7          if F_i.nextDom! = null then
8              domPathResponse ←
9              domPathResponse∪
10             (domain, F_i.egress)⊕
11             {domPathQuery(F_i.nextDom, F_i,
12             F_i.nextDomIngress)};
13         else
14             domPathResponse ←
15             domPathResponse ∪ {(F_i, null)};
16     return domPathResponse;
```

Resource query. For the sake of integrity, we present the pseudocode of chopping and resource query together in Algorithm 2. In particular, when the Unicorn controller receives the domain path for each $(stg, comp)$ candidate pair, it can use this information to chop each path by domains and get the chopping results in Equation (5) (Line 5-12). Then the Unicorn controller can perform efficient resource queries to ask each domain to compute the intra-domain resource view (Line 13-14).

Algorithm 2: The algorithm of chopping and resource query and.

```
1  Function resourceQuery(F, F.domainPath)
2      resourceView ← ∅;
3      foreach domain do
4          domain.F ← ∅;
5      foreach f ∈ F do
6          hIdx ← 0;
7          dom ← getDom(f.domainPath, hIdx);
8          do
9              dom.F ← dom.F ∪ {f};
10             hIdx ← hIdx + 1;
11             dom ← getDom(f.domainPath, hIdx);
12         while dom ≠ null;
13     foreach domain do
14         resourceQueryByDomain(domain, F)
```

This resource query process is efficient due to the following lemma:

Lemma 1. *Given a set of candidate $(storage, computation)$ node pairs for a job set of J, Unicorn achieves the minimal number of resource queries at each domain.*

Proof 1. *With the domain path for each $(str, comp)$ candidate pair, the chopping process yields a set of segments defined in Equation (5), one segment for each domain. Hence the Unicorn controller only needs to generate one resource query for each domain if the corresponding F_i is not empty, which completes our proof.*

Privacy-preserving resource information integration. During the resource query phase, each domain d computes the equivalent resource state abstraction that is only minimal to d itself. When the controller collects the resource state abstraction from every domain, a linear inequality that was from domain d_1 may be a redundant one due to the existence of another linear inequality from domain d_2. For instance, d_1 may return $f_1 + f_2 \leq 10$ to the controller while d_2 may return $f_1 + f_2 \leq 5$. It is easy to see that the cross-domain minimal, equivalent resource state abstraction would only contain $f_1 + f_2 \leq 5$, not $f_1 + f_2 \leq 10$. A strawman approach to compute the cross-domain minimal, equivalent resource state abstraction is to have the controller run the MECS algorithm with all the resource state abstraction from every domain as input. This approach, however, would force each domain to expose unnecessary resource information, *i.e.*, the redundant linear inequality, to the controller, leading to unnecessary privacy leaks.

In Unicorn, we design a privacy-preserving resource information integration protocol that allows every domain to discover linear inequalities in its own domain that are redundant to the minimal cross-domain resource state abstraction. This protocol involves two steps. In the first step, each domain d uses the classic pivoting algorithm [19] to compute all the vertices of the convex polyhedron defined by all the linear inequalities of its own single-domain resource state abstraction. In the second step, each domain d peers with every other domain $d' \in D$, and uses a

customized secure two-party computational geometry protocol to decide if all the vertices computed by d are on the same halfspace defined by a given linear inequality c in the resource state abstraction of d'. If this is true, then c is a redundant inequality in the final cross-domain resource state abstraction, hence will not be sent from domain d' to the controller. After executing this protocol for every inequality in d', domain d' will know which linear inequalities of its own single-domain resource state abstraction are redundant to the single-domain resource state abstraction of d without knowing what the resource state abstraction of d has. We left the details of this privacy-preserving resource information integration protocol in our technical report [20] due to the space limit.

Schedulability. The cross-domain resource discovery process in Unicorn provides an accurate view of resource availability across domains with minimal exposure of private information. One important question left, however, is whether this view provides full a schedulability of resources for a logically centralized orchestrator. We answer this question with the following theorem.

Theorem 1. *When all the resources represented in the final resource state abstraction queried from the cross-domain discovery process in Unicorn can be fully controlled on the edge,* i.e.*, all the attributes of each resource can be controlled by end host, the resource view provided by RSDP provides a full schedulability of resources to a centralized resource orchestrator.*

We omit the proof of this theorem due to the space limit.

5. Global Resource Orchestration

With the accurate, minimal cross-domain resource view, Unicorn performs global resource orchestration to compute optimal resource allocation decisions for a given set of jobs J. The modular design of Unicorn allows different allocation algorithms to be deployed. For simplicity, we consider a set of jobs J with no precedence from the same task, *i.e.*, all the jobs can be executed in parallel. We leave a more generic problem formulation as future work. We assume that each computation resource has infinite computation power, *i.e.*, the data accessing delay reading data from storage resources over networking resources to computation resources is the only bottleneck determining the delay for each workflow. For each job $j \in J$, let Stg_j denote the set of storage resources storing a copy of the input dataset of j, $Comp_j$ denote the set of computation resources that can execute j, v_j denote the volume of input dataset of j, and t_j denote the data accessing delay of j. We also use b_j^{mn} to denote the data access bandwidth for job j from storage resource m to computation resource n, and a binary variable I_j^{mn} to denote if j is assigned storage resource m and computation resource n simultaneously or not. Note that the global resource orchestration component relies heavily on the cross-domain resource discovery component in Section 4. To illustrate this argument, we first give a formulation of the global optimal resource allocation problem *without cross-domain resource discovery* as follows:

$$\text{minimize} \quad max_{j \in J}\{t_j\} \tag{6}$$

subject to

$$\sum_{\{j \in J | n \in Comp_j\}} \sum_{m \in Stg_j} I_j^{mn} \leq 1, \qquad \forall n \in N, \tag{7a}$$

$$\sum_{m \in Stg_j} \sum_{n \in Comp_j} I_j^{mn} = 1, \qquad \forall j \in J, \tag{7b}$$

$$\frac{v_j}{\sum_{m \in Stg_j} \sum_{n \in Comp_j} b_j^{mn} I_j^{mn}} = t_j, \qquad \forall j \in J, \tag{7c}$$

$$A_1(BI) \leq C_1. \tag{7d}$$

$$A_2(BI) \leq C_2. \tag{7e}$$

$$\ldots \tag{7f}$$

$$A_K(BI) \leq C_K. \tag{7g}$$

In this formulation, Equation (6) indicates that the global resource allocation problem aims to minimize the data accessing delay for the whole set of jobs F. Equation (7a) ensures that for each computation resource, at most one job can be assigned. Equation (7b) ensures that only one computation resource and one storage resource are assigned for each job j. Equation (7c) calculates the data accessing delay for each job j. These constraints, *i.e.*, Equations (7a)(7b)(7c) are job-specific, *i.e.*, they express the requirements of data analytics jobs and can be changed accordingly based on different job requirements. The constraints in Equations (7d)(7e)(7f)(7g) are resource-specific, which depends not only on jobs' resource requirements, but also on the attributes provided by resources from each domain.

Though this formulation is accurate itself, its key limitation is that without a cross-domain resource discovery process, it is infeasible to find the resource-specific constraints in Equations (7d)(7e)(7f)(7g). On the contrary, the cross-domain resource discovery in Unicorn copes with this issue by providing the following constraint to accurately represent the resource availability for a given set of jobs with minimal information exposure.

$$A(BI) \leq C. \tag{8}$$

With this formulation, the global optimal resource allocation problem *with cross-domain resource discovery* can be precisely defined as:

$$\text{minimize} \quad max_{j \in J}\{t_j\} \tag{9}$$

subject to

$$\text{Equations (7a)(7b)(7c)(8).} \tag{10a}$$

Solution. The multi-domain resource allocation problem defined above is complex in that it involves binary decisions, non-linear constraints and a complex objective function. To solve this problem, we first linearize the binary decision variables, then use a standard optimization solver to find the solution to the relaxed non-linear optimization problem, and then round-up the linearized decision variables back to the $\{0, 1\}$ feasible space to get the final resource allocation decisions. Because the cross-domain resource discovery process in Unicorn provides the resource view across domains with a minimal set of linear inequalities, the time overhead to solve the relaxed non-linear optimization problem is typically reasonable. We leave the task of finding a more efficient algorithm for this problem as future work.

6. Implementation

In this section, we discuss the implementation details of the Unicorn framework. The system implementation includes the following components:

Resource discovery protocol. We design and develop a query-based resource discovery protocol by extending the Application-Layer Traffic Optimization (ALTO) protocol [21], to deliver the resource state abstraction from each domain to the Unicorn controller. The protocol provides two major services: *path query service* and *resource query service*. The former is used for delivering next hop information to from domain resource managers the Unicorn controller. The latter is used for executing intra-domain resource queries. Table 1 summarizes the basic view of the two services.

Domain resource manager. We build the prototype implementation of the domain resource manager on top of the OpenDaylight SDN controller [22]. From the view of the Unicorn controller, the domain resource manager works as a web service which provides the resource discovery protocol. From the view of the OpenDaylight controller, the resource manager is a consumer to re-process the topology, the traffic statistics, the intra-domain resource information and the inter-domain routing information.

The implementation includes two sub components: An OpenDaylight application running in the Karaf container; and a Python-based web service to provide the resource discovery protocol. The OpenDaylight application uses the API provided by Model-Driven SAL framework to read the real-time network information from the OpenDaylight DataStore. The two sub components communicate via RPC with each other. So the web service component is decoupled with the OpenDaylight and can be adapted to any other network management platform.

To implement the resource query service, we use the Python web service to look up the raw resource state for the given flow set from the OpenDaylight back end. Our native OpenDaylight application collects the topology and forwarding rules from the `network-topology` and `opendaylight-inventory` model of the DataStore, and computes the intra-domain resource state from these information. In our Python web service, we use GLPK as the underlying LP solver to calculate the minimal equivalent resource state abstraction described in Section 4.1. The solver API is wrapped by PuLP so that we could switch to other LP solvers like CPLEX and Gurobi without many modifications.

We implement the path query service as a BGP compatible service. The domain resource manager reads the inter-domain routing information from the OpenDaylight DataStore and converts it to the *BGP RIB* (Routing Information Base) format to respond the path query. The native OpenDaylight could support multiple inter-domain routing protocols by implementing their adapters. In this prototype, we only implement the BGP adapter which feeds the next-hop information of the inter-domain routing from the `bgp-rib` model.

Cross-domain resource discovery. The cross-domain resource discovery implements the two algorithms, path query (Algorithm 1) and resource query (Algorithm 2) and aggregate resource state abstraction from multiple domains to provide a aggregated resource state abstraction to the Global Resource Orchestration. It provides a high-level API `getGlobalResourceView` which accepts a set of node pairs $(srcIP, dstIP)$ as the queried flow set, and returns a set of linear inequalities as the global resource view. In addition, it also provides some lowel-level APIs including: `getDomainPath` that implements the Algorithm 1 and returns the domain path; and `getDomainResource` that retrieves the intra-domain resource view from a domain via resource discovery protocol.

Global resource orchestration. We implement the global resource orchestrator to subscribe to the analytics job management database. Once new jobs are inserted into the database, the orchestrator fetches them, performs cross-domain resource discovery and then make resource allocation decisions. It provides numerous Python APIs for developing new resource allocation algorithms. Therefore it is flexible for administrators to update the resource allocation policy. Our current orchestrator makes resource allocation decisions by solving the optimization problem defined in Section 5.

7. Performance Evaluation

We evaluate the performance of Unicorn through trace-based simulations. In particular, we focus on the efficiency of Unicorn in (1) discovering and represent a cross-domain resource view with minimal information exposure; and (2) performing global resource allocation decisions for data analytics jobs. All the simulations are conducted on a laptop with two 1.6GHz Intel i5 Cores and a 4GB memory.

7.1. Methodology

We emulate three multi-domain data analytics networks with different number of domains and topologies. For each setting, we first randomly select one topology from Topology Zoo [23] and let that topology be the domain-level topology with each node represent a single domain. And we also generate the intra-domain topology, *i.e.*, switches and the intra-domain links, for each domain. The emulated multi-domain topologies are labeled as Arpanet (composed of 4 domains), Aarnet (composed of 19 domains) and Chinanet (composed of 42 domains). We set the available link bandwidth within each domain to be 0.2-1Gbps and the available link bandwidth between domains to be 2-4Gbps. And we assume the I/O bandwidth of storage and computation resources are way larger than the bandwidths of links. We assume each domain's intra-domain and inter-domain routing policies both use the typical routing policies, *i.e.*, the shortest path routing, except that the former is on the router level and the latter is on the domain level. We vary the number of data analytics jobs J from the same task to be from 5 to 30, each of which requires reading 1000 gigabytes of data.

7.2. Results

Cross-domain resource discovery and representation. We first present the compression ratio of the Unicorn in discovering and representing the accurate, minimal intra- / cross- domain resource views. This is computed as by dividing the number of linear inequalities in the accurate, minimal intra- / cross- domain over the number of original, linear inequalities used to represent the resource

Service	Path Query	Resource Query
HTTP Method	POST	POST
Media Type	`application`	`application`
Accept Subtype	`alto-flowfilter+json`	`alto-flowfilter+json`
Content Subtype	`alto-nextas+json`	`alto-pathvector+json`
Function	Implement $getNextDomain()$ in Algorithm 1.	Implement $resourceQueryByDomain()$ in Algorithm 2.

Table 1: Unicorn Resource Discovery Protocol

availability across domains. Figure 7 shows this compression ratio in a 19-domain data analytics network derived from the Aarnet topology [23] with different number of data analytics jobs, and Figure 8 shows this ratio under different number of domains when fixing the number of jobs to be 20. From these results we observe that the average compression ratio of intra-domain resource view is only around 60-70% while that of the cross-domain resource view is around 25-45%. These show that Unicorn provides a highly compact view of cross-domain resource availability for data analytics jobs. The higher compression ratio in the cross-domain view is because a multi-domain data analytics network provides more resources for data analytics jobs, *i.e.*, there are fewer jobs sharing the same set of resources. On the other hand, the fact that the highest cross-domain compression ratio is still 45% shows that even with more resources, jobs sharing the same set of resources is still a common situation, indicating the necessity and importance for discovering the accurate, minimal resource availability across domains.

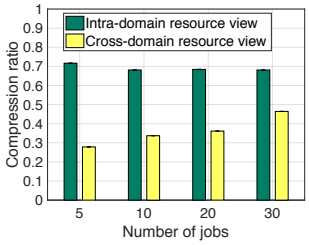


Figure 7: Compression ratio of intra-domain resource view and cross-domain resource view with varying numbers of jobs.

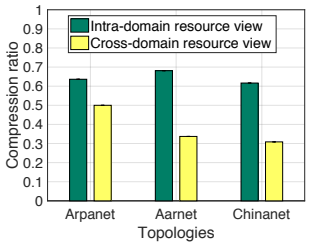


Figure 8: Compression ratio of intra-domain resource view and cross-domain resource view with different topologies.

We also plot the number of linear inequalities in the intra- /cross- domain view discovered by Unicorn in Figure 9 and Figure 10. We see that as the number of domains and the number of jobs grow, the number of linear inequalities in the accurate, minimal resource view computed by Unicorn increases at a very slow rate, which demonstrates the scalability of Unicorn.

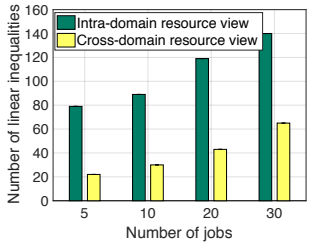


Figure 9: Number of linear inequalities in intra-domain resource view and cross-domain resource view with varying numbers of jobs.

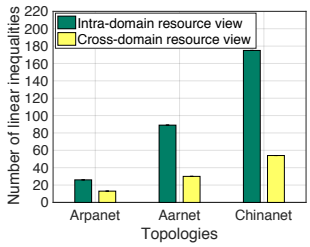


Figure 10: Number of linear inequalities in intra-domain resource view and cross-domain resource view with different topologies.

Global resource orchestration. We next demonstrate the efficiency of Unicorn in performing global resource orchestration for data analytics jobs. In particular, we focus on the latency of a task composed of a job set J, which is computed as the longest execution time of all jobs. In our evaluation, we assume all the computation nodes have the same computation power, hence we only need to focus on minimizing the maximal data accessing delay among all jobs, as defined in Equation (6). We compare the task latency provided by Unicorn with that provided by a domain-path based resource allocation scheme, which allocates computation and storage resources for a job based on the shortest AS path and use the classic max-in fairness mechanism to allocate bandwidth among data accessing flows of analytics jobs. We summarize the results under the combinations of different multi-domain topologies and different numbers of jobs in Table 2. We see that Unicorn provides an up to 65% task latency reduction in all cases. This shows that Unicorn provides a significant latency re-

duction for multi-domain data analytics.

Topology \ #Jobs	5	10	20	30
Arpanet	31%	24%	27%	65%
Aarnet	27%	46%	55%	10%

Table 2: The reduction of task latency of Unicorn over the domain-path allocation scheme with max-min fairness.

8. Conclusion and Future Work

Summary. In this paper, we identify the objective and the fundamental challenge for designing a resource orchestration system for multi-domain, geo-distributed data analytics system through analyzing the data analytics trace from one of the largest scientific experiments in the world and examining the design of existing resource management systems for single-domain clusters. We design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics systems. Unicorn realizes the accurate, cross-domain resource availability discovery with minimal information exposure of each domain through the RSDP and a novel, efficient cross-domain resource availability query algorithm. Unicorn also provides a global resource orchestrator to compute optimal resource allocation decisions for data analytics tasks. We present the implementation details and the preliminary evaluation results of Unicorn.

Prototype and full demonstration at SuperComputing 2017. The source code and more comprehensive evaluation results of Unicorn will be open-sourced at [24]. A full demonstration of the Unicorn prototype will be given at SuperComputing 2017. In this demonstration, we will demonstrate the efficiency and efficacy of Unicorn on cross-domain resource discovery and global resource allocation in a multi-domain, geo-distributed data analytics system involving the Caltech booth, the USC booth and the UNESP booth at the conference exhibition, the SCinet network, and the Caltech testbed at Pasadena.

Acknowledgement

We thank Shenshen Chen, Shiwei Chen and Kai Gao for helpful discussion during the work. The Yale team was supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions; NSFC #61672385, FAST Magellan. The Yale team is also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The Tongji team was supported by China Postdoctoral Science Foundation #2017-M611618; NSFC #61702373. The Caltech team was supported in part by DOE/ASCR project #000219898, SDN NGenIA; DOE award #DE-AC02-07CH11359, SENSE, FNAL PO #626507; NSF award #1246133, ANSE; NSF award #1341024, CHOPIN.

Referencese

[1] T. C. Collaboration, The CMS experiment at the CERN LHC, Journal of Instrumentation 3 (08). doi:10.1088/1748-0221/3/08/S08004.

[2] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurrency and computation: practice and experience 17 (2-4) (2005) 323–356.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in: NSDI, 2011.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache Hadoop YARN: Yet another resource negotiator, in: SoCC, ACM, 2013, p. 5.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: EuroSys, ACM, 2015, p. 18.

[6] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, L. Zhou, Apollo: Scalable and coordinated scheduling for cloud-scale computing, in: OSDI, 2014, pp. 285–300.

[7] Under the hood: Scheduling MapReduce jobs more efficiently with Corona, http://on.fb.me/TxUsYN, [Online; accessed: 09-May-2017].

[8] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, F. Wurthwein, The pilot way to grid resources using glideinWMS, in: CSIE, IEEE, 2009, pp. 428–432.

[9] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, G. Varghese, WANalytics: Analytics for a geo-distributed data-intensive world, in: CIDR, 2015.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, Low Latency Geo-distributed Data Analytics, in: SIGCOMM, ACM, 2015, pp. 421–434. doi:10.475/123.

[11] C.-C. Hung, L. Golubchik, M. Yu, Scheduling jobs across geo-distributed datacenters, in: SoCC, ACM, 2015, pp. 111–124.

[12] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, S. Wang, Rapier: Integrating routing and scheduling for coflow-aware data center networks, in: INFOCOM, 2015.

[13] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, J. Bi, Nova: Towards on-demand equivalent network view abstraction for network optimization, in: IWQoS 2017, 2017.

[14] D. Jeffrey, G. Sanjay, MapReduce: simplified data processing on large clusters, Communications of the ACM.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: HotCloud'10.

[16] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, V. ICSI, Making sense of performance in data analytics frameworks, in: NSDI, 2015, pp. 293–307.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined WAN, in: SIGCOMM'13.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: SIGCOMM, ACM, 2013.

[19] D. Avis, K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete & Computational Geometry 8 (3) (1992) 295–313.

[20] Secure cross-domain resource discovery, technical report, http://www.cs.yale.edu/homes/qiaoxiang/publications.

[21] R. Alimi, Y. Yang, R. Penno, RFC 7285, Application-layer traffic optimization (ALTO) protocol (2014).

[22] J. Medved, R. Varga, A. Tkacik, K. Gray, Opendaylight: Towards a model-driven SDN controller architecture, in: IEEE WoWMoM, 2014.

[23] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo 29 (9) 1765–1775, 00249.
[24] Public repository of unicorn, `https://github.com/snlab/Unicorn`.

Wireless Pers Commun (2017) 95:2691–2708
DOI 10.1007/s11277-017-3950-8

Game-Theoretic User Association in Ultra-dense Networks with Device-to-Device Relays

Geng Li[1,2] · Yuping Zhao[3] · Dou Li[3]

Published online: 18 January 2017

Abstract Device-to-device communication can assist cellular networks by making certain users equipment (UEs) work as relays between the base station (BS) and other users. In this paper, we present the ultra-dense network (UDN) with D2D relays instead of small cells, where UEs can form into clusters according to the traffic demand in hot-spot areas. Each UE requires to decide whether to connect to the BS, or to get associated with one of the D2D relays, a.k.a. cluster heads (CHs). To optimize the downlink system performance, we propose a game-theoretic user association scheme in the UDN with D2D relays, specifically focused on load balancing among the BS and CHs. The dynamic user association is formulated as a hedonic coalition game where we adopt a simplified but efficient measurement of the utility and select the effective game players in a smaller number. In the game, we estimate the number of users associated with each CH at the Nash-stable state which can indicate the overall expected load condition, and an admission control mechanism is finally employed on the basis of these values. Simulation results show that the UDN adopting the D2D relay technology can achieve a higher system rate than the traditional cellular network, and the proposed user association scheme outperforms the existing schemes while having a small computational complexity.

Keywords D2D relay · Ultra-dense network · User association · Load balancing · Hedonic coalition game

✉ Geng Li
geng.li@yale.edu

Yuping Zhao
yuping.zhao@pku.edu.cn

Dou Li
lidou@pku.edu.cn

[1] Department of Computer Science, Tongji University, Shanghai, China

[2] Department of Computer Science, Yale University, New Haven, CT, USA

[3] School of Electronics Engineering and Computer Science, Peking University, Beijing, China

1 Introduction

In the vision of 5G, the link capacity and the connection density are two of the vital requirements in the future wireless networks, as METIS project has identified the "Amazingly fast" with "instantaneous connectivity" and the "Great service in a crowd" scenarios [1]. To cope with these, the Ultra-Dense Networks (UDN) deployment is deemed to be one of the most promising solutions [2]. The traditional UDN that incorporates dense small cells in areas with an expected high traffic demand can improve the network capacity and coverage. Nevertheless, as the traffic demand in networks is time-varying, which leads to the locations of hot-spots getting dynamically changed, the small cells with fixed locations hence not only fail to meet all the demand, but also incur extra energy consumption and interference [3, 4].

Fortunately, 5G brings the Device-to-device (D2D) communication as an underlay to cellular networks, which makes direct communication between devices feasible [5]. The D2D relay technology has been recently proposed and studied in [6–9], where a D2D-enabled user equipment (UE) can assist cellular transmission by acting as a relay between the the base station (BS) and some other UEs within a cluster. Due to the absence of the closed loop physical layer feedback link supported in 3GPP specifications before Release 12, most of the academic studies on D2D relays mainly focus on addressing broadcast or multicast services [10]. However, the 3GPP specifications groups have agreed to enhance unicast services for D2D relays in Release 13, which makes the common data transmission practicable [11].

To cope with the problems in traditional UDN with fixed small cells and motivated by the the merit of D2D relays, we present the ultra-dense network with D2D relays in this paper. As shown in Fig. 1, certain UEs with better link conditions to the BS serve as D2D relays, a.k.a. cluster heads (CHs), while other UEs are allowed to get dynamically associated with a CH who can forward all their traffic from the BS, rather than directly connecting to the BS. Compared with the traditional UDN with small cells whose locations are fixed, the UDN with D2D relays can adapt the traffic demand varying with time and locations, and thus is more flexible and demand-driven. In addition, the system capacity, as well as the connections density can be improved.

The dynamic user association for the D2D relay requires more consideration about load balancing than that for the regular and fixed relay, because the topology and the load distribution varies from time to time. For instance, the UE i in Fig. 1 is allowed to get associated with one of the neighboring D2D relays (CH 1 or CH 2), and moreover, it can directly connect to the BS if both the neighboring CHs are overloaded. As the number of UEs increases, the user association problem will become more dynamic and intractable. In conventional user association schemes, each user is associated with the node on the basis of a certain criterion, such as the node with the maximum received power [12] or the maximum effective link SINR (signal to interference and noise ratio) [13]. However, since the load conditions on different nodes are not well considered, it will lead to congestion at certain CHs. In [14, 15], the user association schemes by jointly considering both the radio link and load qualities have been proposed, but the users get associated greedily without admission control to improve the overall system performance.[1]

In this paper, we aim at addressing the user association problem in the UDN with D2D relays. Specifically, the major contributions of this paper are summarized as follows:

[1] The user association scheme in [14] and that in [15] are similar to each other with slight changes, therefore we treat them as one throughout the rest of this paper.

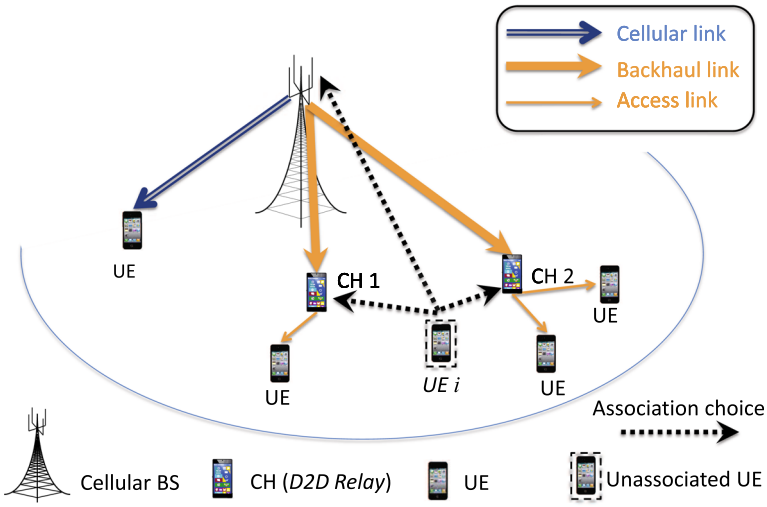


Fig. 1 Illustration of the ultra-dense network with D2D relays, where all the users equipment (UEs) are D2D-enabled. In this UDN, a UE is allowed to get associated with the BS or one of the CHs (D2D relays)

1. We present the ultra-dense network with D2D relays, which enables UEs form into clusters and the CH serves the UEs within a cluster as a D2D relay, such that the network is more flexible and demand-driven than that with small cells.
2. We propose a game-theoretic user association scheme that optimizes the system downlink rate in the UDN with D2D relays. Specifically, a *hedonic coalition game* is formulated and further refined with a simplified measurement of utility and a smaller number of effective players. Then an admission control policy, on the basis of the estimated users numbers at the Nash-stable state, is performed to archive system load balancing.

Simulation results demonstrate that although the D2D relay serves users in a two-hop fashion and consumes two orthogonal resources, the network adopting the D2D relay technology can achieve a higher system rate than the traditional cellular network at the same consumption of resource. The results also validate that our proposed user association scheme for a local optimal solution but almost obtains the same performance as the global optimization, and it provides better performance than the existing schemes such as the conventional ones based on the maximum power [12] and the maximum effective SINR [13], as well as the user association scheme presented in [14, 15]. In addition, the proposed scheme with a low computational complexity can be better applied in practice.

The rest of this paper is organized as follows. Section 2 presents the system model of a ultra-dense network with D2D relays. The user association problem is discussed in Sect. 3, including the problem formulation, the proposed game-theoretic scheme and the time complexity analysis. Section 4 provides the performance evaluation results compared with various existing schemes. Finally, we conclude this paper in Sect. 5.

2 System Model

Without loss of generality, we consider a cellular network with one BS and a number of D2D-enabled UEs which can either work as D2D relays or directly communicate with D2D relays. We only concentrate on the downlink transmission in this paper, and the results can be extended to the uplink case easily.

In the UDN with D2D relays, UEs can form into clusters according to the traffic demand in hot-spot areas, with the definitions below.

Definition 1 (*Nodes and links*) A *cluster* is a temporary transmission set, including a bunch of UEs. One certain UE working as a D2D relay is defined as the *CH (cluster head)*, while the other UEs defined as *cluster members* are served by the CH in a two-hop fashion to communicate with the BS. The number of cluster members in a cluster is defined as the *cluster size*. The radio link between the BS and a CH is referred to as the *backhaul link*. The radio link between a CH and a cluster member is referred to as the *access link*, and that between the BS and an ordinary UE out of any clusters is referred to as the *cellular link*, as shown in Fig. 1.

In our model, we consider that there are N UEs and C clusters whose CHs are assigned in advance. Each UE i ($i \in \{1, 2, \ldots, N\}$) can get associated with the BS or one of the CHs (CH j, $j \in \{1, 2, \ldots, C\}$), corresponding to two cases when conducting the downlink data transmission as shown in Fig. 2, namely the cellular transmission (Case A), and the D2D-relay transmission (Case B). Once UE i is associated with CH j, we think UE i joins the cluster and becomes one of the cluster members of cluster j.

We assume that the CH (a D2D relay) employs the half-duplex relay technology, where two orthogonal resources (e.g., two frequency bands or equivalently two time slots) are needed for the respective reception and transmission. Specifically, the first resource consumed in backhaul links takes the proportion of $1 - \eta$, while the second orthogonal resource consumed in access links takes the rest proportion of η. In this paper, the two hops of transmission operate in different frequency bands, say f_1 and f_2 as shown in Fig. 3. Since f_1 and f_2 are orthogonal, there is no interference to each other. Let W_1 denote the bandwidth of f_1 for the backhaul link (the BS to the CH) and W_2 denote the bandwidth of f_2 for the

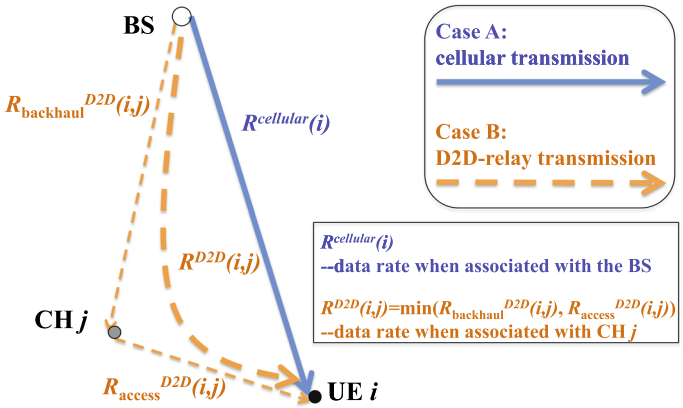


Fig. 2 Two cases for downlink transmission

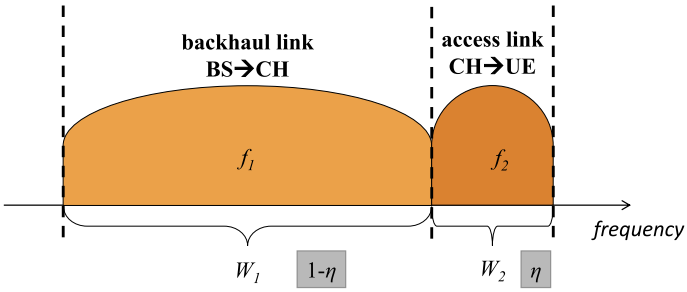


Fig. 3 Illustration of the bandwidth partition for D2D relays in backhaul link and access link, where η $(0<\eta<1)$ is the bandwidth partition ratio

access link (the CH to the cluster member), then $W_2/W_1 = \eta/(1-\eta)$, where η $(0<\eta<1)$ is the bandwidth partition ratio.

Regarding to the resource allocation, we adopt the proportional fair model [16–18], in which the BS or each CH evenly divides its available bandwidth amongst its users, i.e., the traffic is assumed to be homogenous. Note that the cellular link of a UE when associated with the BS also operates in f_1 band, and the bandwidth resource is orthogonally allocated among UEs and CHs. Every UE needs a portion of f_1 spectrum, which is used for either the cellular transmission or the backhaul transmission by the CH. Then the bandwidth resource acquired by a UE when associated with the BS (Case A) is given by $B = W_1/N$, which is an equal share from f_1 band.[2] The CH is treated as a "super UE" with the aggregated data traffic of its cluster members, so the total bandwidth for CH j in the backhaul link is $\sum_{k=1}^{k_j} B$, where k_j denotes the cluster size of cluster j (i.e., the number of UEs associated with CH j). For the access link of the UE when associated with CH j (Case B), the bandwidth can be given by W_2/k_j, which can reflect the load condition in cluster j.

Given the allocated bandwidth in these two cases, we can quantify the downlink rate of a user in consideration.

Case A: Cellular transmission.

For a given UE i, when it is associated with the BS and independent to any cluster, its data rate $R^{cellular}(i)$ can be calculated as the Shannon capacity as follows:

$$R^{cellular}(i) = B * \log_2(1 + \beta * SINR_{BS,i}), \tag{1}$$

where β called the *SINR* gap is set as 1 for simplicity, and $SINR_{BS,i}$ is the *SINR* over the cellular link between the BS and UE i. Then $SINR_{BS,i}$ can be expressed as:

$$SINR_{BS,i} = \frac{|h_{BS,i}|^2 P_{BS}}{\sum_{k=1}^{N_{BS}} |h_{BS_k,i}|^2 P_{BS} + \sigma_w^2}, \tag{2}$$

where P_{BS} denotes the transmission power of the cellular BS, $h_{BS,i}$ denotes the channel gain between the BS and UE i, and σ_w^2 is the variance of the white Gaussian noise. It can be seen that the interference in f_1 frequency band comes from the N_{BS} neighboring cells, and $h_{BS_k,i}$ is the channel gain from the BS k.

[2] Even if the UE gets associated with a CH, it still needs the f_1 resource for its backhaul link, and we assume that CHs have no demands for transmitting their own data, so here the denominator is N.

Case B: D2D-relay transmission.

When UE i is associated with CH j, there is no cellular link between the UE and the BS, and all the data traffic from BS will be relayed by CH j. In order not to affect other UEs or CHs, UE i will devote its belonging resource B to the CH for its data transmission in the backhaul link. Note that this portion of bandwidth in the backhaul link can be only used for transmitting downlink data of UE i. Thus, the effective rate of the data dedicated to UE i in the backhaul link can be expressed as:

$$R^{D2D}_{backhaul}(i,j) = B * \log_2(1 + SINR_{BS,CH_j}), \tag{3}$$

where $SINR_{BS,CH_j}$ is the received $SINR$ of CH j from the BS, and its value can be calculated in the same fashion as (2).

The rate in access link between CH j and UE i is given as:

$$R^{D2D}_{access}(i,j) = \frac{W_2}{k_j} * \log_2(1 + SINR_{CH_j,i}). \tag{4}$$

Note that we only consider the interference from other CHs in the same cell in f_2 frequency band, and the interference from neighboring cells is not taken into account. Therefore the received $SINR$ in access link is determined as:

$$SINR_{CH_j,i} = \frac{\left|h_{CH_j,i}\right|^2 P_{CH}}{\sum_{j'=1,j'\neq j}^{C} \left|h_{CH_{j'},i}\right|^2 P_{CH} + \sigma_w^2}, \tag{5}$$

where P_{CH} denotes the transmission power of the CH, and $h_{CH_j,i}$ is the channel gain between CH j and UE i.

In respect that the backhaul and access links work in different frequency bands, so the CH as a D2D relay can transmit and receive data at the same time. Based on the max-flow min-cut theorem for the link capacity [19], the end-to-end rate experienced by UE i when associated with CH j is the minimum of the two values, expressed as follows:

$$R^{D2D}(i,j) = \min\left(R^{D2D}_{backhaul}(i,j), R^{D2D}_{access}(i,j)\right). \tag{6}$$

With the data rates in two cases given above, we can address the dynamic user association in the next section.

3 User Association in Ultra-dense Networks with D2D Relays

In the network, a UE requires to decide to (1) communicate with the BS directly, or (2) select one of the CHs to relay its traffic. Our goal is to find the user association in the whole network that maximizes the system downlink rate.

3.1 Problem Formulation

Definition 2 (*Decision variables*) There are two permissible decision sets $\mathcal{X} \subseteq \{0,1\}^{N\times C}$ and $\mathcal{Y} \subseteq \{0,1\}^N$, where N and C are the numbers of allocable users and clusters. The binary *decision variables* $x_{j,i} \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ $(i \in \{1,2,\ldots,N\},\ j \in \{1,2,\ldots,C\})$ are defined as:

$$x_{j,i} = \begin{cases} 1, & \text{if UE } i \text{ is associated with CH } j, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

$$y_i = \begin{cases} 1, & \text{if UE } i \text{ is associated with the BS,} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Then the cluster size of cluster j can be expressed as $k_j = \sum_{i=1}^{N} x_{j,i}$.

With the help of the decision variables, we formulate an integer programming problem as below:

$$max\left(\sum_{i=1}^{N} y_i \times R^{cellular}(i) + \sum_{j=1}^{C}\sum_{i=1}^{N} x_{j,i} \times R^{D2D}(i,j)\right), \tag{9}$$

$$s.t. \sum_{j=1}^{C}(x_{j,i} + y_i) = 1, \forall i \in \{1, 2, \ldots, N\}. \tag{10}$$

The objective function in (9) denotes the overall system downlink rate, where the first part is the sum rate of UEs associated with the BS while the second part is that of UEs associated with the C CHs. Since each UE can only get associated with one node for data transmission, therefore the sum of decision variables of a UE equals 1, as the constraint shown in (10).

In such problem, every UE has $C + 1$ options for association. However, the payoff in C of them is dynamically changing before the association process is completed. As a result, the dynamic association problem can be proved to as NP-hard as in [14, 20], and the global optimal solution can be solely obtained by exhaustive search.

3.2 Proposed Game-Theoretic User Association Scheme

As mentioned before, it is hard to solve the integer programming problem directly due to the dynamic change of utilities, and the centralized optimal solution may cost significant computing time. Instead, our idea is to formulate the dynamic association as a *hedonic coalition game* where we adopt a simplified but efficient measurement of the utility (Step 1) and select the effective game players in a smaller number (Step 2). In the game, we estimate the number of UEs associated with each CH at the Nash-stable state which is an indicator of the expected load condition of the overall system (Step 3), and an admission control mechanism is employed on the basis of these values (Step 4). A similar but crude approach has been studied in our previous work [21], and the one proposed in this paper is more sophisticated and advanced. Our game-theoretic, Nash stable-based user association scheme is a local optimal solution, and basically consists of the following four steps.

3.2.1 Step 1: Preparing for the Game

Our aim is to maximize the overall system downlink rate, therefore the utility of each player in the game is the individual downlink rate. However, the data rates are different when the UE is associated with different nodes, and even when associated with the same CH, the UE's data rate varies along with different association results in that cluster. Then,

in order to simplify the measurement of utilities, we calculate the maximum numbers of coexisting peers that can be tolerated in clusters, and build the corresponding matrix, which is defined as below:

Definition 3 (*MNCP*) The *matrix of maximum numbers of coexisting peers (MNCP) that can be tolerated in clusters* is defined as $(\alpha(j,i))^{C\times N}$. The matrix element $\alpha(j,i)$ represents that if UE i wants to get associated with CH j for achieving a higher rate than that in the cellular link, the cluster size of cluster j can not exceed $\alpha(j,i)$, which is given by (11).

$$\alpha(j,i)=\begin{cases}\dfrac{B\cdot\log_2(1+SINR_{BS,i})}{W_2\cdot\log_2(1+SINR_{CH_j,i})} & R^{cellular}(i)<R^{D2D}_{backhaul}(i,j),\\ 0 & R^{cellular}(i)\geq R^{D2D}_{backhaul}(i,j).\end{cases} \quad (11)$$
$$\forall i\in\{1,2,\ldots,N\}\ and\ j\in\{1,2,\ldots,C\}$$

Note that the value of $\alpha(j,i)$ is obtained by solving $R^{cellular}(i)=R^{D2D}(i,j)$. The condition "$R^{cellular}(i)<R^{D2D}_{backhaul}(i,j)$" in (11) implies that UE i will only be associated with the CH whose backhaul rate is higher than the UE's. The rationale is because the effective end-to-end rate $R^{D2D}(i,j)$ by D2D-relay transmission is limited to the CH's rate, and it is no necessary for UE i to get associated with a CH whose rate is even lower than the UE's.

Remark 1 According to the physical meaning of $\alpha(j,i)$, it is noted that

- if $\alpha(j,i)$ is less than 1, it means UE i cannot tolerate any other coexisting peers in cluster j, thus it is not available to be associated with CH j;
- if $\alpha(j,i)$ is more than 1, it means UE i will gain a higher rate when joining cluster j, as long as the number of UEs in cluster j does not exceed $\alpha(j,i)$.

Building the matrix of MNCP is the preparatory work for the following game. In order to make it clear and intuitive, we give an example of our proposed association scheme, and the matrix of MNCP in the example is illustrated at "Step 1" in Fig. 4.

3.2.2 Step 2: Selecting the Game Players in Each Cluster

As the user density in the network becomes higher, the scale of the matrix will grow multiplicatively and the game will be too complex if all of the association strategies are taken into account. However, quite a large proportion of association strategies in the network are redundant and do not need to be considered (e.g., a UE can not possibly be

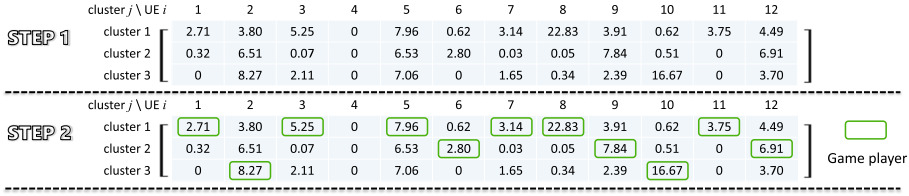


Fig. 4 An example of Step 1 and Step 2 of our proposed association scheme in the case with 3 CHs and 12 UEs. *Step 1* Preparing for the game; *Step 2* Selecting the game players. The *green rectangles* at Step 2 represent the game players. (Color figure online)

associated with a CH far away from it). Therefore, we can refine the game and select the effective players in each cluster.

According to the matrix calculated in Step 1, each UE has a column of MNCP. It picks the maximum value in that column, and the corresponding row specifies the preferred cluster for the UE. A UE choosing the cluster with the maximum MNCP means this UE has the strongest willingness to be associated with that CH rather than others, because it can tolerant the most coexisting peers in that cluster, which is good for this UE as well as the other cluster members. Here we manually remove the case when the maximum value of a column is less than 1 (the case that the UE has no preferred cluster to join), where the UE will stay associated with the BS. Here we use j_i^* to denote the preferred cluster of UE i, then we have

$$j_i^* = \arg\min_j \alpha(j, i). \tag{12}$$

In the meantime, UE i becomes one of the game players in cluster j_i^*. This process allows every cluster to select its effective game players in a smaller number, which are illustrated as "Step 2" in Fig. 4 by green rectangles.

3.2.3 Step 3: Estimating the Cluster Sizes at the Nash-Stable State

Based on the selection process in Step 2, all game players in each cluster are given. However, not all of them can eventually get associated with the CH due to the constraint of available bandwidth and the difference of the utilities, so each player has two strategies: *join* the cluster or *not join* the cluster. According to the players' strategies in a cluster, the user association will result in the formation of two disjoint coalitions, and hence the game in each cluster is classified as a coalition formation game. Coalition formation has been a topic of high interest in game theory [22], and a certain class of coalition formation games known as the *hedonic coalition game* is defined as follows [23].

Definition 4 (*Hedonic Coalition Game*) A coalition formation game is classified as the *hedonic coalition game*, if

1. The payoff of any player depends solely on the members of the coalition which the player belongs to.
2. The coalitions form as a result of the preferences of the players over their possible coalitions' set.

These two conditions characterize the framework of hedonic games. Mainly, the term"hedonic" pertains to the first condition above, whereby the payoff of any player, in a hedonic game, must depend only on the identity of the players in the coalition to which the player belongs, with no dependence on the other players. In the user association problem of this paper, the utility of UE i when associated with CH j_i^* is only related to the cluster members in cluster j_i^*, and independent from other clusters. For the second condition, considering all the players are rational and selfish, each player will choose the preferred coalition where it can achieve a higher utility than that in others. Therefore, according to the the matrix of MNCP, the *switch rule* of each player can be given as below.

Definition 5 (*Switch Rule*) We consider there are two coalitions for each cluster, which represent the users getting associated with it and those not getting associated with it. Then

the *switch rule* of a player is that, UE i decides to join cluster j_i^* if the cluster size is less than its MNCP (i.e., $k_{j_i^*} < \alpha(j_i^*, i)$), and to leave it otherwise (i.e., $k_{j_i^*} \geq \alpha(j_i^*, i)$).

Independent from the preference relations selected, the *switch rule* can be seen as a selfish decision made by a player to move from its current coalition to a new coalition regardless of the effect of this movement on the other players. However, any switching behavior of a player may cause the change of members in the old and new coalitions. Obviously, all the coalitions will change constantly in the formation process, unless all players can reach a Nash-stable state which is defined as a certain state where any movement for a player will lead to a utility decline [24].

To solve this problem, we first sort the players' MNCP in each cluster by the descending arrangement, like $\alpha_1 > \alpha_2 > \alpha_3 \ldots$. Then we estimate the cluster size of each cluster at the Nash-stable state based on Theorem 1, which can be described as the biggest nonnegative integer k_j^{NS} such that there would be at least k_j^{NS} players whose MNCP is greater than k_j^{NS} in cluster j.

Theorem 1 *Starting from at any initial states, all clusters will always end up with a convergence to a final Nash-stable state with the cluster sizes k_j^{NS} obtained below.*

$$k_j^{NS} = \inf_{k_j} k_j < \alpha_k \quad \forall k = 1, 2, \ldots, k_j. \tag{13}$$

Proof Case I: $k_j < k_j^{NS}$. There must exist other players willing to join cluster j for achieving higher rates, because there are more than k_j players whose MNCP $\alpha(j, i) > k_j + 1$.

Case II: $k_j > k_j^{NS}$. The cluster will be too crowded that more than $k_j - k_j^{NS}$ players prefer to leave it, since the number of cluster members exceeds their MNCP that can be tolerated, and the BS is better for them.

Case III: $k_j = k_j^{NS}$. Any other player who tries to join the cluster will have a rate decline, and the players already in the cluster can tolerate with the cluster size hence prefer to remain there. Thus none of the players wants to deviate from this steady state.

Therefore, the final number of UEs associated with CH j $(j \in \{1, 2, \ldots, C\})$ must be k_j^{NS}. □

Remark 2 The estimated cluster sizes of all clusters at the Nash-stable state can reflect the expected load condition of the overall system.

In this step, the number of UEs that associated with each cluster is estimated in a game-theoretic fashion. The estimating process in the example is illustrated at "Step 3" in Fig. 5.

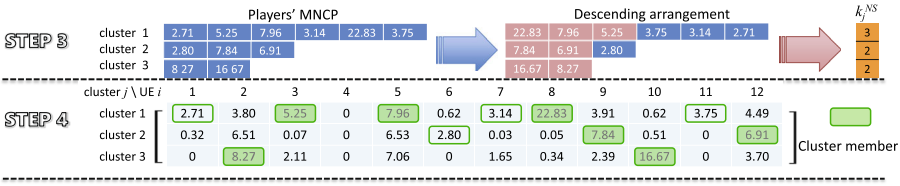


Fig. 5 An example of Step 3 and Step 4 of our proposed association scheme in the case with 3 CHs and 12 UEs. *Step 3* Estimating the cluster sizes at the Nash-stable state; *Step 4* Admission control. The *green boxes* at Step 4 represent the cluster members after association. (Color figure online)

3.2.4 Step 4: Admission Control

Eventually, in order to balance the load among the BS and the CHs, an admission control policy is required to derive the ultimate solution to the dynamic user association problem. As the cluster size of each cluster has been estimated, all we need to do is to keep the first k_j^{NS} players with the highest values of MNCP in each cluster. The rest of UEs will directly connect to the BS and be served as ordinary UEs. Finally if the cluster size in a cluster is zero, it means that no UE will join that cluster.

This admission control process is shown at "Step 4" in Fig. 5, where the cluster members are labeled with solid green boxes. The ultimate solution to the user association problem in the example is:

- BS users' IDs: *1, 4, 6, 7, 11*;
- cluster 1 users' IDs: *3, 5, 8*;
- cluster 2 users' IDs: *9, 12*;
- cluster 3 users' IDs: *2, 10*.

3.3 Complexity Analysis

The normal solution to obtain the optimal user association is the exhaustive search by a centralized controller that picks the association with the maximum overall gain among all permissible association decisions. Since there are $(C+1)^N$ possible results for all the N UEs, thus the complexity of this scheme can be denoted by $O((C+1)^N)$. This makes the exhaustive search computationally impossible in the real-world implementation.

In the conventional association schemes, each UE needs to compare the power strengths of the received signals or the effective link SINR from at most a number of C CHs as well as the BS, therefore the sum complexity is $O(N \cdot (C+1))$.

The time complexity of the scheme in literature [14, 15] is $O((C+1) \cdot N \cdot log_2(C+1))$, whose detailed calculation can be found in [15].

Proposition 1 *The time complexity of the proposed association scheme is* $O(C \times N + N^2)$.

Proof In Step 1, we adopt the measurement of utilities by the matrix of MNCP, whose number of elements is $C \times N$. In Step 2, we select the effective game players by choosing the maximum in each column, thus the time complexity is also $O(C \times N)$. In Step 3, we assume that there are $n_1, n_2, \ldots, n_C$ game players in cluster $1, 2, \ldots, C$, so $\sum_{j=1}^{C} n_j \leq N$. Then in order to estimate the cluster sizes at the Nash-stable state, we need to make comparisons under the condition in (13) for ${n_j}^2$ times in the worst case. So it has to be computed at most $\sum_{j=1}^{C} {n_j}^2$ times in Step 3, which are less than or equal to N^2, because $\sum_{j=1}^{C} {n_j}^2 \leq (\sum_{j=1}^{C} n_j)^2 \leq N^2$. Step 4 simply executes a sorting process, whose time complexity can be proved less than or equal to $O(N^2)$ in the similar way.

Consequently, the sum complexity of the four steps is $O(C \times N + C \times N + N^2 + N^2)$, and the complexity has the upper bound of $O(C \times N + N^2)$, which is comparable to $O(N \cdot (C+1))$ by the conventional schemes, similar to $O((C+1) \cdot N \cdot log_2(C+1))$ by the scheme in [14, 15], but much lower than $O((C+1)^N)$ by the exhaustive search. □

4 Performance Evaluation

4.1 Simulation Setup

We simulate a cellular network with a number of D2D relays (CHs) and D2D-enabled UEs randomly deployed in the cell with a radius of 500m. 6 neighboring BSs are considered for calculating the inter-cell interference. Generally, in order to improve the performance of users in the entire cluster, we prefer to specially select the UEs with good channel conditions in backhaul links as the CHs (e.g., the UE by the windows, the UE with LOS (line-of-sight) link to the BS, etc.). As in the literatures, we assign CHs with a 5dB gain in the backhaul link, which is nearly equal to the building penetration loss [6, 25]. The transmission power of the BS is set as $P_{BS} = 46\,\text{dBm}$, and that of the CH is set as $P_{CH} = 23\,\text{dBm}$, which is the maximum uplink transmission power of a UE specified in the standard. The rest of detailed channel parameters and simulation settings conform to "3GPP TR 36.814", where only the large scale fading channel is considered.

We take "Traditional cellular network", i.e., the system rate achieved in the traditional cellular communication network as the baseline. To conduct a fair comparison between the traditional cellular network and the UDN with D2D relays, the resource consumption in these two scenarios should be equal. As a result, the total bandwidth of the traditional cellular network is set as $W = W_1 + W_2$ in the simulation. We set $W = 10\,\text{MHz}$ as a fixed bandwidth, and the values of W_1 and W_2 depend on the bandwidth partition ratio η, i.e., $W_1 = (1-\eta) \times 10\,\text{MHz}$ and $W_2 = \eta \times 10\,\text{MHz}$.

Multiple association schemes are compared in the simulation, and the schemes along with their time complexity (see Sect. 3.3) are shown in Table 1. Note that "Best-power [12]" and "Best-SINR [13]" represent the conventional user association schemes based on the maximum received power and the maximum effective link SINR, respectively.

4.2 Simulation Results

Figure 6 shows the simulation results of the impact of the bandwidth partition ratio η on the system sum rate. We fix the value of $W = W_1 + W_2$, and change the partition of bandwidth ($W_1 = (1-\eta) \cdot W$, $W_2 = \eta \cdot W$) by adjusting η. The scenario of a high user density ($N = 500$) is considered, and the proposed user association scheme is employed in the UDN with D2D relays. As the results show, with different numbers of CHs ($C = 50, 100$), the UDN with D2D relays always can archive a higher system rate than the traditional cellular network at the same consumption of bandwidth resource. Since the CHs relay the traffic of some UEs from the BS, those UEs' original poor channel conditions of cellular links can be replaced by the better ones provided by the CHs. In particular, the network with D2D relays only outperforms the traditional one when η is small from the

Table 1 Complexity of the schemes compared in the simulation

User association schemes	Time complexity
Exhaustive search	$O((C+1)^N)$
Best-power [12]	$O(N \cdot (C+1))$
Best-SINR [13]	$O(N \cdot (C+1))$
Scheme in [14, 15]	$O((C+1) \cdot N \cdot log_2(C+1))$
Proposed	$O(C \times N + N^2)$

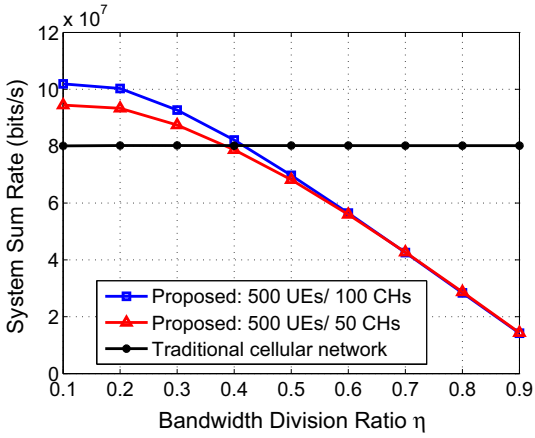


Fig. 6 The comparison of the system sum rate in the network with D2D relays and the traditional one as a function of the bandwidth partition ratio η (system sum rate vs. η)

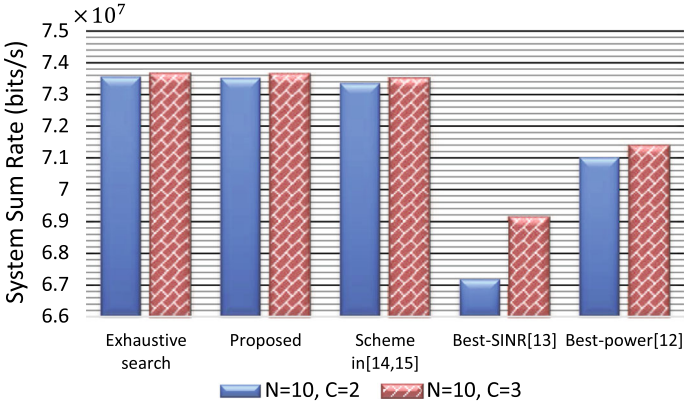


Fig. 7 The comparison of the system sum rate for various user association schemes with different CHs numbers (including the exhaustive search scheme)

results, say $\eta < 0.4$. That is because the backhaul link of the D2D relay is more likely to be a bottleneck than the access link, and W_2, the little portion of total bandwidth, is more efficiently utilized in clusters than in the cellular transmission. Therefore, the bandwidth partition ratio is set as $\eta = 0.1$ in the following simulation.

Figure 7 shows the results with a small number of UEs ($N = 10$), to make the exhaustive search scheme computationally feasible.[3] It is shown that, our proposed user association scheme performs better than the conventional ones and the scheme in the literature. More importantly, the proposed scheme obtains almost the same performance as exhaustive search, which provides an upper bound of the association problem. Nevertheless, our scheme has a much lower complexity according to Table 1.

Figure 8 illustrates the number of relay serving UEs versus the number of CHs for various user association schemes in the scenario of a high user density ($N = 500$), where we remove the comparison with the exhaustive search scheme which is computationally

[3] The time complexity of the exhaustive search scheme is still considerable. Taking the case of $N = 10$ and $C = 3$ as an example, the complexity reaches as high as $(3 + 1)^{10}$.

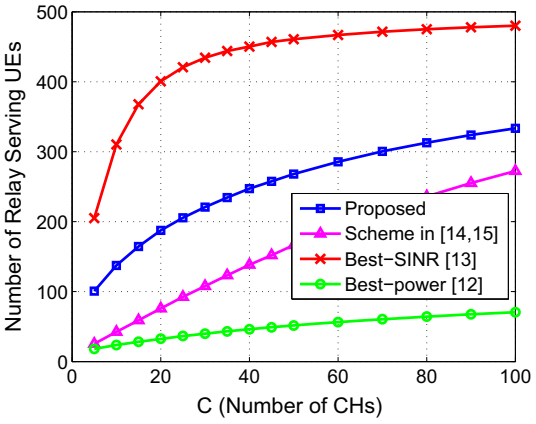


Fig. 8 Number of relay serving UEs versus CHs number

expensive here. In the "Best-power [12]" scheme, due to the imbalance of transmission power between the BS and the CH, only a few of UEs can be associated into clusters; while it is just the opposite by the "Best-SINR [13]" scheme, because the criterion makes the CHs more favoured. For the association scheme in [14, 15], the UEs are associated sequentially until the resource consumption exceeds the constraint, therefore the number of UEs in each cluster can hardly reach the optimal result. For the proposed scheme in this paper, since the association is based on the estimated cluster sizes at the Nash-stable state which can reflect the overall expected load condition, so the number of relay serving UEs is growing more properly and can approximate the optimal.

Figure 9 compares the system sum rate in the scenario of $N = 500$ for various user association schemes and the traditional network. It is shown that, without an efficient user association (e.g., the "Best-power [12]" scheme), the system rate in the network with D2D relays is even lower than the traditional cellular network, because the D2D relay need two orthogonal resources. Referring to the results in Fig. 8, the relay serving UEs are way too many in the "Best-SINR [13]" scheme, so the clusters are overloaded from the beginning (40 cluster members in each cluster on average when the CHs number is 5). As the CHs number increases, the number of overloaded clusters also increases, which leads to performance decline. But when the number of clusters is big enough to relive the congestion at certain CHs, the performance can get improved gradually. Therefore the system rate for the "Best-SINR [13]" scheme as a function of the CHs number presents as a concave curve. The greedy user association scheme proposed in [14, 15] can offload some traffic from the BS to the CHs, and hence also leads to a better performance than the conventional counterparts. However, there is no admission control (as in our scheme) for the users to improve the overall system performance. As a result, by jointly considering the link quality and the load condition, our proposed scheme achieves a much higher rate than the other ones, but has the complexity comparable to them.[4]

In Fig. 10, the comparison of the fairness index in the scenario of $N = 500$ is shown. The fairness index F is defined by $F = (\sum_{i=1}^{N} R(i))^2/(N\sum_{i=1}^{N} R(i)^2)$, where $R(i)$ is the data rate of UE i [26]. It is observed that the fairness for each scheme has the similar trend as

[4] The comparison of complexity between the proposed scheme and scheme in [14, 15] depends on the exact values of N and C. For example, when $N = 500$ and $C = 20$, the complexity of the proposed scheme is higher; but when $N = 500$ and $C = 100$, that of the scheme in [14, 15] becomes higher.

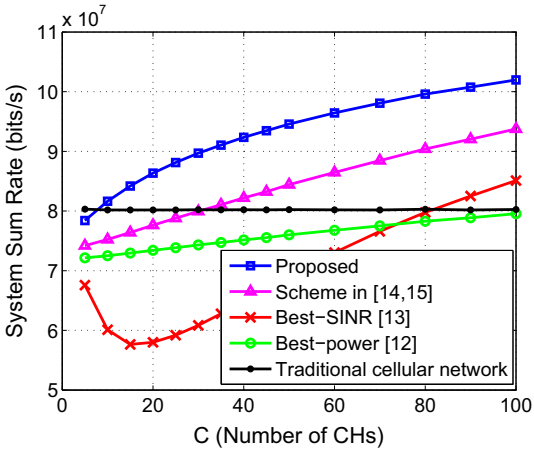


Fig. 9 System sum rate versus CHs number

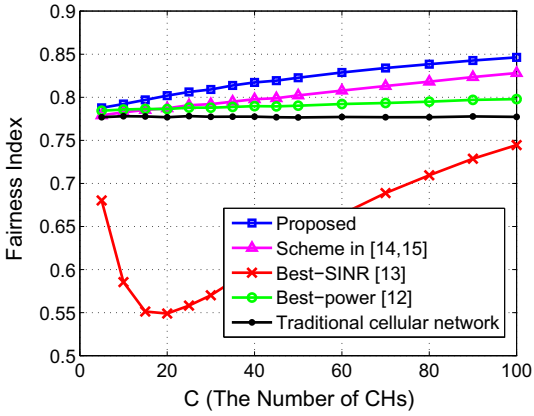


Fig. 10 Fairness index versus CHs number

the system rate results in Fig. 9, because the system rate is partly related to the balance of load distribution in the network. In the meantime, the results can demonstrate that our proposed user association scheme also has the advantage in terms of the system fairness.

5 Conclusions

The ultra-dense network with D2D relays can adapt the traffic demand varying with time and locations. Although the D2D relay serves users in a two-hop fashion and consumes two orthogonal resources, the network adopting the D2D relay technology can achieve a higher system rate than the traditional cellular network at the same consumption of resource, as long as there is effective user association in the network. In this paper, we propose a game-theoretic, Nash stable-based user association scheme to address dynamic load balancing among the BS and the CHs. We compared our proposed scheme with the existing schemes by simulation, including the conventional ones based on the maximum power or SINR, and also the user association scheme presented in the literature. Simulation results demonstrate that our game-theoretic scheme has the advantage in terms of the system throughput, load

balancing and the system fairness. In addition, the proposed scheme leads to limited computation complexity.

Acknowledgements This work was supported in part by Huawei Technologies Company, Ltd., and in part by the National Science and Technology Major Project of China under Grant 2016ZX03001018-005.

References

1. Osseiran, A., Boccardi, F., Braun, V., Kusume, K., Marsch, P., Maternia, M., et al. (2014). Scenarios for 5G mobile and wireless communications: The vision of the metis project. *IEEE Communications Magazine*, *52*(5), 26–35.
2. Baldemair, R., Irnich, T., Balachandran, K., Dahlman, E., Mildh, G., Selén, Y., et al. (2015). Ultra-dense networks in millimeter-wave frequencies. *IEEE Communications Magazine*, *53*(1), 202–208.
3. Valkama, M., & Niemelä, J. (2015). Spectral and energy efficiency of ultra-dense networks under different deployment strategies. *IEEE Communications Magazine*, *53*(1), 90–100.
4. Li, Y.-N. R., Li, J., Wu, H., & Z., Wenfeng. (2014). Energy efficient small cell operation under ultra dense cloud radio access networks. In *Globecom Workshops (GC Wkshps), 2014* (pp. 1120–1125). IEEE.
5. Doppler, K., Rinne, M., Wijting, C., Ribeiro, C. B., & Hugl, K. (2009). Device-to-device communication as an underlay to LTE-advanced networks. *IEEE Communications Magazine*, *47*(12), 42–49.
6. Zhang, G., Yang, K., Shuanshuan, W., Mei, X., & Zhao, Z. (2015). Efficient power control for half-duplex relay based D2D networks under sum power constraints. *Wireless Networks*, *21*(7), 2345–2355.
7. Gong, W., & Wang, X. (2015). Particle swarm optimization based power allocation schemes of device-to-device multicast communication. *Wireless Personal Communications*, *85*(3), 1261–1277.
8. Lin, X., Ratasuk, R., Ghosh, A., & Andrews, J. G. (2014). Modeling, analysis, and optimization of multicast device-to-device transmissions. *IEEE Transactions on Wireless Communications*, *13*(8), 4346–4359.
9. Zhou, B., Honglin, H., Huang, S.-Q., & Chen, H.-H. (2013). Intracluster device-to-device relay algorithm with optimal resource utilization. *IEEE Transactions on Vehicular Technology*, *62*(5), 2315–2326.
10. TR 36.843. (2014). *Study on LTE device to device proximity services; radio aspects (release 12)*. Technical report, 3GPP Technical report.
11. RP-151952. (2015). *Enhanced D2D status report for ran 70*. Technical report, Status Report to TSG.
12. Damnjanovic, A., Montojo, J., Wei, Y., Ji, T., Luo, T., Vajapeyam, M., et al. (2011). A survey on 3GPP heterogeneous networks. *IEEE Wireless Communications*, *18*(3), 10–21.
13. Mezzavilla, M., Somasundaram, K., & Zorzi, M. (2014). Joint user association and resource allocation in UE-relay assisted heterogeneous networks. In *2014 IEEE International Conference on Communications Workshops (ICC)*, pp. 628–634.
14. Yu, Y., Hu, R. Q., Bontu, C. S., & Cai, Z. (2011). Mobile association and load balancing in a cooperative relay cellular network. *IEEE Communications Magazine*, *49*(5), 83–89.
15. Siviloglou, O., Rouskas, A., & Karetsos, G. (2015). Association of mobile stations in cellular networks with relay nodes. In *2015 38th international conference on telecommunications and signal processing (TSP)*, pp. 1–5.
16. Singh, S., Dhillon, H. S., & Andrews, J. G. (2013). Offloading in heterogeneous networks: Modeling, analysis, and design insights. *IEEE Transactions on Wireless Communications*, *12*(5), 2484–2497.
17. Sadr, S., & Adve, R. S. (2014). Tier association probability and spectrum partitioning for maximum rate coverage in multi-tier heterogeneous networks. *IEEE Communications Letters*, *18*(10), 1791–1794.
18. Lin, Y., & Yu, W. (2013). Optimizing user association and frequency reuse for heterogeneous network under stochastic model. In *2013 IEEE global communications conference (GLOBECOM)* (pp. 2045–2050). IEEE.
19. West, D. B., et al. (2001). *Introduction to graph theory* (Vol. 2). Upper Saddle River: Prentice Hall.
20. Li, Q., Hu, R. Q., Wu, G., & Qian, Y. (2012). On the optimal mobile association in heterogeneous wireless relay networks. In *INFOCOM, 2012 Proceedings IEEE*, pp. 1359–1367.
21. Li, G., Zhao, Y., & Bian, K. (2016). Efficient user association in cellular networks with hybrid cognitive radio relays. *IEEE Communications Letters*, *20*(7), 1413–1416.
22. Ray, D. (2007). *A game-theoretic perspective on coalition formation*. Oxford: Oxford University Press.

23. Saad, W., Han, Z., Basar, T., Debbah, M., & Hjorungnes, A. (2011). Hedonic coalition formation for distributed task allocation among wireless agents. *IEEE Transactions on Mobile Computing*, *10*(9), 1327–1344.
24. Jackson, O. (2006). Definitions of equilibrium in network formation games. *International Journal of Game Theory*, *34*(3), 305–318.
25. Sui, Y., Guvenc, I., & Svensson, T. (2015). Interference management for moving networks in ultra-dense urban scenarios. *EURASIP Journal on Wireless Communications and Networking*, *2015*(1), 1–32.
26. Raj, J., Dah-Ming, C., & Hawe, W. R. (1984). *A quantitative measure of fairness and discrimination for resource allocation in shared computer system* (Vol. 38). Hudson, MA: Eastern Research Laboratory, Digital Equipment Corporation.

Geng Li received the B.S. degree in electronics engineering, the B.A. degree in economics (double major), and the Ph.D. degree in wireless communications from Peking University, Beijing, China, in 2011, 2011 and 2016, respectively. He is currently working as a postdoctoral fellow in the Department of Computer Science from Yale University, USA, and the Department of Computer Science from Tongji University, China. His research interests include Software-defined Network, LTE Layer 1&2, Protocol Stack, HetNet, relay, D2D, indoor localization, UMTS, signal processing and network security.

Yuping Zhao received the B.S. and M.S. degrees in electrical engineering from Northern Jiaotong University, Beijing, P. R. China, in 1983 and 1986, respectively. She received the Ph.D. and Doctor of Science degrees in wireless communications from Helsinki University of Technology, Helsinki, Finland, in 1997 and 1999, respectively. She was a system engineer for telecommunication companies in China and Japan. She worked as a research engineer at the Helsinki University of Technology, Helsinki, Finland, and at the Nokia Research Center in the field of radio resource management for wireless mobile communication networks. Currently, she is a Professor in the State Key Laboratory of Advanced Optical Communication Systems & Networks, School of Electronics Engineering and Computer Science, Peking University, Beijing, P. R. China. Her research interests include the areas of wireless communications and corresponding signal processing, especially for OFDM, UWB and MIMO systems, cooperative networks, cognitive radio, and wireless sensor networks.

Dou Li is an associate professor in the School of Electronics Engineering and Computer Science, Peking University. She received B.S., M.S. and Ph.D. degrees from Peking University, China, in 1989, 1992, and 2007, respectively. In 1992, she became an assistant professor in Peking University. Now she is an associate professor of Advanced Institute of Wireless Communication and Signal Processing, Peking University. Her current research interests include signal processing for information system and resource management in wireless communication system.

2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications

Auc2Reserve: A Differentially Private Auction for Electric Vehicle Fast Charging Reservation

Invited Paper

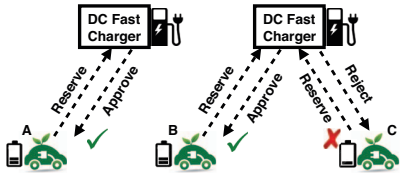
Qiao Xiang [1,5], Linghe Kong[2,3], Xue Liu[2], Jingdong Xu[4], Wei Wang[1]
[1]Department of Computer Science and Technology, Tongji University, China
[2]School of Computer Science, McGill University, Canada
[3]Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[4]Department of Computer Science and Technology, Nankai University, China
[5]Department of Computer Science, Yale University, United States
qiao.xiang@tongji.edu.cn, linghe.kong@sjtu.edu.cn, xueliu@cs.mcgill.ca,
xujd@nankai.edu.cn, wwang@tongji.edu.cn

***Abstract*—The increasing market share of electric vehicles (EVs) makes charging facilities indispensable infrastructure for integrating EVs into the future intelligent transportation systems and smart grid. One promising facility called fast charging reservation(FCR) system was recently proposed. It allows people to reserve fast chargers ahead of time. In this system, fast chargers are the most scarce resource instead of electricity. Thus how to allocate these charging points requires careful designing. A good allocation policy should 1) ensure charging points to be allocated to EV users who really value them, and 2) prevent users' private information, e.g., identity, personal agenda, residing area and etc., from being inferred. A simple combination of classic multi-item auction and user identity anonymization cannot satisfy both criteria simultaneously. To find such an allocation, in this paper we investigate the design of privacy-preserving auctions in FCR systems. Traditional privacy-preserving strategies such as cryptography could incur high computation and communication overhead and hence jeopardize the efficiency of allocation. To this end, we propose *Auc2Reserve*, a differentially private randomized auction. *Auc2Reserve* applies an improved approximate sampler and the belief propagation(BP) technique to accelerate the resource allocation and pricing process. As a result, it is much more computationally efficient than generic exponential differentially private mechanisms and other theoretical approximate implementations. Through theoretical analysis, we show that *Auc2Reserve* is γ-incentive compatible, individual rational and ϵ-differentially private. And it provides a close-form approximation ratio in social welfare of FCR systems. In addition, we also demonstrate the efficacy of *Auc2Reserve* in terms of social welfare and privacy leakage via numerical simulation.**

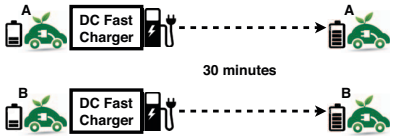
I. Introduction

The electric vehicle (EV) is visioned as a crucial component of intelligent transportation systems (ITS) [5]. Compared with gasoline-powered vehicles, EVs have the potential benefits of a lower carbon emission, a lower powering cost and a higher power efficiency. With these promising benefits, however, they also introduce a high penetration into the power grid by shifting the energy load from gasoline to electricity. As EVs' market share is increasing, the large-scale integration of EVs into the future smart grid has drawn great attention from both academia and industry. And charging facilities have become indispensable infrastructure to support such integration[10][13].

Among various charging facilities have been designed and studied, e.g., home charging point [9] [24], workplace charging facility [10] [23] and etc., one up-and-coming facility called *fast charging reservation* (FCR) system was recently proposed. In an FCR system, EV users can send requests to reserve Direct Current (DC) fast charging points at different locations, which are capable of charging the battery of EV to 80% capacity, i.e., a 0.8 state of charge(SOC), in half an hour. Figure 1 gives an overview of the FCR system. FCR systems facilitate users to charge EVs during a long distance trip without experiencing long-time charging delay. Several FCR systems have been deployed in major automobile markets [4], [1], [2]. For instance, Tesla has deployed over 400 Supercharger stations across the United States [4]. And China has initiated a project to develop a FCR system with over 600 fast chargers along major highways across the country by 2020[2].



(a) EVs reserve fast chargers.



(b) EVs get charged within 30 minutes.

Fig. 1: An overview of fast charging reservation system.

One important observation we can get from these FCR systems is that the main principle when deploying a FCR system is to ensure coverage, i.e. distribute DC fast chargers across a large area. This is because 1) the hardware cost of

Most part of this work is done during the first author's Postdoctoral Fellowship in the Cyber-Physical System Lab (CPSL) at McGill University.

978-1-5090-2479-7/16 $31.00 © 2016 IEEE
DOI 10.1109/RTCSA.2016.19

DC fast charger is high; and 2) the short charging time of fast charger determines that a 1:1 ratio between the number of fast chargers and EV is unnecessary. As a result, in the FCR system, the number of fast charging points is much less than that of EVs. This means when EV users send reservation request to the FCR system, they are competing for not only the electricity, but also the fast charging points. It is shown in recent study [3] [35] that in an FCR system where there are more EVs than fast charging points, fast chargers are *the most scarce resource*, instead of the commonly assumed electricity. Therefore, *how to allocate fast chargers* in FCR systems requires careful designing.

Regardless of the differences on hardware and charging schedule, current FCR systems usually adopt a first-reserve-first-serve approach with fixed pricing policies, e.g., pay-per-use or flat-rate, to allocate fast charging points to EV users. Though this strategy is simple and could help the market expanding of EVs, they are not efficient allocation mechanisms in that 1) fast charging points may not be assigned to EV users who really value them, i.e., EVs with a lower state of charge (SOC); and 2) overpricing and underpricing could happen due to the fluctuation of electricity price, thus impairing the benefit of both EV users and fast charging stations. These deficiencies have also been identified in other charging facilities. Recently researchers propose to tackle these drawbacks using auctions as resource allocation strategy for EV charging. Different auctions have been proposed to compute an efficient allocation of electricity and charging points for EV users so that the system social welfare can be maximized. And social welfare is the monetary sum of the revenue of charging facilities and the utility gained by EV users. Exemplary studies in this area include auctions for residential charging [32], park-and-charge [29] and FCR systems [35].

As a powerful tool for resource allocation in EV charging, auctions in current studies are designed to incentivize EV users to truthfully report their valuation on different sets of resources, i.e., incentive compatible, so that social-welfare-maximization allocation and pricing decisions can be derived. However, forcing EV users to reveal their real valuation profile during the auction put users at the risk of exposing their privacy. These real valuation contains users' preferences on different charging points and different amount of electricity. Adversaries may use these information to infer users' personal information, such as transportation agenda, residing area and etc. These information have high commercial value. More importantly, they are also crucial for EV users' personal safety. And due to the fact that EVs need to be charged every one or two days, users may participate EV charging auction frequently, which makes inferences on these information even easier. Though anonymizing users' valuation profile appears to be an efficient approach for protecting these information and users' identity, recent studies [27], [15] show that adversaries can easily deanonymize users' identity and expose all such private profiles by linking two or more separate sets of users' profiles. Such privacy vulnerabilities is a major barrier preventing the large scale deployment of auction-based EV charging resource allocation. In this paper, we aim to find solutions to overcome this obstacle and hence advocate the further development of electric vehicles. In particular, we take FCR systems as an example and explore the feasibility and benefits of designing *an efficient privacy-preserving auction* to allocate fast chargers, the most scarce resource in FCR systems, between EV users. Designing such a mechanism requires us to address a series of challenges:

Challenge 1. The proposed auction should preserve user privacy while satisfying other requirements of mechanism design, i.e., (approximate) incentive compatibility and individual rationality.

Challenge 2. The proposed auction should provide an explicit guarantee on social welfare of FCR systems.

Challenge 3. The proposed auction should be computationally efficient so that the allocation and pricing decisions can be quickly made in large-scale FCR systems.

Dealing with these challenges is a non-trivial mission. Traditional privacy-preserving mechanisms use cryptosystems to protect users' privacy [26][28]. However, the high computation and communication overhead in such cryptosystems often compromise the performance of corresponding mechanisms such as social welfare and incentive compatibility. McSherry *et al.* [25] proposed to incorporate differential privacy in mechanism design. In a differentially private mechanism, it is hard to infer users' personal information as a single change in users' reported valuation has very limited impact on the outcome of the auction. And it is also proved that differential privacy implies approximate incentive compatibility [25]. Nonetheless, the generic differentially private mechanism has an exponential computational complexity. Though some polynomial-time approximate implementation was proposed afterwards [19], it is only theoretical and impractical in real-world due to its $O(n^{13})$ complexity with a large implied constant.

In this paper we cope with the aforementioned challenges by designing *Auc2Reserve*, a computationally efficient differentially private auction. For an FCR system with M EV users competing for N fast chargers, we developed an improved randomized approximate sampler in *Auc2Reserve* to iteratively allocate fast charging points to EV users. Leveraging the fact that there are usually more EV users than fast chargers in FCR system, *Auc2Reserve* randomly selects an EV user (agent) for receiving a given fast charging point (item) at every iteration. In this way, it reduces the sampling overhead by $\frac{M-N}{M}$ times than that of the original sampler [19]. By integrating this allocation process with an approximate pricing function in generic differentially private mechanisms [19] [25], *Auc2Reserve* successfully address Challenges **1** and **2**. Both the allocation and pricing policies in *Auc2Reserve* involve computing the permanent of non-negative matrix, which is #P-complete. To resolve Challenge **3**, we apply the belief propagation technique [11][12] for permanent approximation in *Auc2Reserve*. As a result, not only does *Auc2Reserve* ensure incentive compatibility, individual rationality and differential privacy, it is also computationally efficient in making social-welfare-guaranteed allocation and pricing decisions for FCR systems.

Our **main contributions** in this paper are as follow:

- We study the novel problem of designing privacy-preserving auction for EV fast charging reservation systems. In particular, We propose *Auc2Reserve*, a differentially private randomized auction. Compared with generic exponential differentially private mechanisms and other approximate implementations, *Auc2Reserve* is much more computationally efficient in making allocation and pricing decisions. In addition, *Auc2Reserve* can also be generalized for different scenarios in FCR systems.

• Through theoretical analysis, we show that *Auc2Reserve* is γ-incentive compatible, individual rational and ϵ-differentially private. It also provides a close-form approximation ratio on social welfare of FCR system. We also demonstrate the efficacy of *Auc2Reserve* under various settings of FCR systems in terms of social welfare and privacy leakage via numerical simulation.

The remaining of this paper is organized as follows. We introduce system settings, related solution concepts and present a formal problem definition in Section II. We give an exponential generic differentially private mechanism in Section III. We develop the *Auc2Reserve* differentially private auction for FCR systems and analyze its performance in Section IV. We demonstrate the efficacy of *Auc2Reserve* via simulation in Section V. We discuss related work on EV charging facilities and auction theory in Section VI, and conclude our paper in Section VII.

II. System Settings and Problem Formulation

In this section, we present the settings of EV fast charging reservation systems, discuss related solution concepts and formally define the auction problem in FCR system.

A. System Description

We consider a fast charging reservation system composed of a set of M EV users, indexed by $i = 1, 2, \ldots, M$, and N DC fast charging points, indexed by $j = 1, 2, \ldots, N$. We assume that $M > N$ as the number of EVs is usually higher than that of DC fast charging points in real world systems. In our model we divide time into slots with a equal length of 30-minute. Thus every fast charging can be finished within one time-slot. To keep a concise presentation, we focus on the auction scenario where all EV users submit bids to reserve fast charging points and electricity for the same future time slot. Our solution to this simpler scenario can be easily generalized to the reservation auction in multiple time slots, as will be discussed in Section IV-E.

In the auction, every EV user i can submit multiple bids to the system central controller to reserve fast charging point. These bids are sent via mobile devices, personal computers or a reliable vehicle-to-infrastructure communication system, e.g., OnStar. Every EV user is unit-demand, i.e., she only needs at most one charging point. Thus for every EV user, no duplicate bids on the same charging point is allowed. We use an M-by-N matrix $\mathbf{B}$ to denoted the bids submitted by EV users. The bids submitted by EV user i are denoted by a row vector $\mathbf{b_i}$. This vector is composed of $b_{ij}, j = 1, 2, \ldots, N$, where $b_{ij} \geq 0$ represents user i's *reported valuation* on getting charging point j in a monetary form. A $b_{ij} > 0$ means that user i submitted a bid of value to reserve the charging point j, and a $b_{ij} = 0$ means that user i does not submit any bid to reserve charging point j in this auction. Other than the reported valuation, every user i also has a *real valuation* v_{ij} on reserving every charging point j. This real valuation is private to user i, which can be affected by many factors, e.g., personal agenda, distance to a certain charging station, risk preference and etc., and is also expressed in monetary form. In addition, given an EV user i, we use $\mathbf{b_{-i}}$ and $\mathbf{v_{-i}}$ to represent the reported valuation and real valuation of all EV users other than i, respectively.

After collecting bids from all EV users, a central controller of the FCR system makes allocation decision on fast charging points, and the corresponding pricing decision. We use a set of binary decision variables $y_{ij} \in \{0, 1\}$ to denote the allocation decision for every bid b_{ij}. A $y_{ij} = 1$ means the bid b_{ij} is a winning bid and user i will get a reservation charging point j. And user i need to pay Γ_i for this reservation. A $y_{ij} = 0$ means user i does not win the bid b_{ij} and will pay nothing for this bid. Because every user is unit-demand, we have $\sum_j^N y_{ij} = 1$ for any EV user i. We define u_i, the utility for user i as follows:

$$u_i = \sum_{j=1}^{N} v_{ij} y_{ij} - \Gamma_i$$

In the auction, every user is selfish and aims to maximize her own utility. We use SW to denote the social welfare of FCR system, which is calculated as the sum of the revenue made by fast charging points and the utility of all users. And we can express it as:

$$SW = \sum_{i}^{M} \Gamma_i + \sum_{i}^{M} u_i = \sum_{i}^{M} \sum_{j}^{N} v_{ij} y_{ij}. \tag{1}$$

We see that the EV fast charging reservation auction falls into the category of multi-item auction[7][8]. We use $r = \{r[1], r[2], \ldots, r[M]\}$ to denote an allocation outcome, where $r[i]$ records which charging point is assigned to user i. If $r[i] \in [1, N]$, we have $y_{ir[i]} = 1$, otherwise $y_{ir[i]} = 0$. We denote the set of all its allocation outcomes as R. And the number of possible allocation outcomes in the auction, i.e., the cardinality of R, can be expressed as $|R| = \frac{M!}{(M-N)!}$.

B. Solution Concepts

To avoid overpricing and underpricing, and to allocate the fast charging point to EV users who really value it, a good auction mechanism for FCR system should possess the following properties:

Incentive Compatibility. An auction achieves *incentive compatibility* if every user i can always maximize her utility by truthfully reporting her *real valuation* as the *reported valuation* no matter what strategies are adopted by other users, i.e., $u_i(\mathbf{v_i}, \mathbf{b_{-i}}) \geq u_i(\mathbf{b_i}, \mathbf{b_{-i}})$ for any $\mathbf{b_i}$. Incentive compatibility saves users the trouble to perform complex strategic calculations[22]. In addition, we also consider an approximate form of incentive compatibility, called γ*-incentive compatibility*. i.e., $u_i(\mathbf{v_i}, \mathbf{b_{-i}}) \geq u_i(\mathbf{b_i}, \mathbf{b_{-i}}) - \gamma$, where $\gamma \geq 0$ is a small constant. This relaxed definition further simplifies the design and analysis of mechanism. When $\gamma = 0$, we see that it reduces to the original definition of incentive compatibility.

Individual Rationality. An auction achieves *individual rationally* if every participating user always gets non-negative utility regardless what strategy is adopted by her and other users, i.e., $u_i(\mathbf{b_i}, \mathbf{b_{-i}}) \geq 0$, for any $\mathbf{b_i}$. This property is also known as the "participation constraint" [22] [19].

Social Welfare Maximization. As shown in Equation (1), the auction maximizes social welfare by maximizing the total *real valuation* of all winning bids. However, the real valuation v of every bid is private information to EV user, and is unknown to the charging reservation system. When the auction is incentive compatible, the expression of social welfare in Equation (1) can be rewritten as $SW = \sum_i^M \sum_j^N b_{ij} y_{ij}$ since the reported valuation for each bid equals to the corresponding real valuation [22].

Other than incentive compatibility, individual rationality and social welfare maximization, the auction for EV FCR systems also needs to ensure the privacy of EV users. This is because users need to charge their EVs every one or two days. As a result, they would participate fast charger reservation auction frequently, which makes the inference on EV users' personal preferences much easier. Not only do such information have high commercial value, they are also crucial in protecting EV users personal safety. Therefore, it is important to design a privacy-preserving auction for FCR systems. Among various privacy standards, in this paper we focus on *differential privacy*, a paradigm for private data analysis that has drawn much attention in the past decade [15] [36].

Differential Privacy. Given a small constant $\epsilon > 0$, an auction *Auc* is ϵ-differentially private if for any two sets of reported valuation $\mathbf{b}$ and $\mathbf{b}'$ that only differ in one reported valuation, and for any set of allocations $S \subset R(Auc)$, we have

$$\mathbf{Pr}[Auc(\mathbf{b}) \in S] \leq \mathbf{Pr}[Auc(\mathbf{b}') \in S] \cdot exp(\epsilon).$$

Differential privacy has many elegant theoretical properties as well as useful applications [16]. And its feasibility and potentials in mechanism design have been studied under different scenarios, such as digital good auction[22] [19], spectrum auction [37] [36] and etc. Having reviewed related concepts in mechanism design and privacy, we are able to formally define the auction design problem for FCR systems.

FCR-Auc Problem: Given the aforementioned settings of FCR system, design an auction that is γ-incentive compatible, individual rational, ϵ-differentially private, and maximizes the social welfare in a computationally efficient way.

III. Exponential Differential Private Mechanism

One general technique in designing differentially private auction is the exponential mechanism [25] [19]. The basic idea of exponential mechanism, denoted as EXP, is to first compute a probability for every feasible allocation outcome $r \in R$ as follows:

$$\mathbf{Pr}[EXP(R, Q, D, \epsilon) = r] = exp(\frac{\epsilon}{2\Delta} Q(D, r)). \quad (2)$$

In Equation (2), Q is the *quality function* in the differential privacy literature [16]. It takes a data set D and a feasible allocation outcome r as the input and compute a real-valued score as the output. When designing differentially private mechanisms, D is usually the set of reported valuation $\mathbf{b}$, and $Q(\mathbf{b}, r) = \sum_i^M \sum_j^N v_{ij} y_{ij}(r)$ is the social welfare of allocation r. In addition, Δ is the Lipschitz constant. Without loss of generality, we set it to be 1.

With the probability for every feasible outcome r, the exponential mechanism randomly selects one outcome based on the computed probability in Equation (2) as the final allocation decision in the auction, and then makes corresponding pricing decisions. We present the exponential differential private mechanism EXP_ϵ^R for the **FCR-Auc** problem in Algorithm 1. Among different pricing policies, we use the well-studied policy proposed in [19] for EXP_ϵ^R.

EXP_ϵ^R has many nice properties, applying the results in [25] we can have the following theorem on the social welfare of EXP_ϵ^R.

Algorithm 1 EXP_ϵ^R: An Exponential Differentially Private Mechanism for **FCR-Auc** Problem

1: INPUT: An M-by-N bidding matrix $\mathbf{B}$
2: **for** every feasible allocation outcome $r \in R$ **do**
3: $\quad \mathbf{Pr}[r] \rightarrow exp(\frac{\epsilon}{2} \sum_i^M b_{ir[i]})$
4: **end for**
5: **for** every feasible allocation outcome $r \in R$ **do**
6: $\quad \mathbf{Pr}[r] \rightarrow \frac{\mathbf{Pr}[r]}{\sum_{r \in R} \mathbf{Pr}[r]}$
7: **end for**
8: Select an allocation outcome r with probability $Pr[r]$
9: **for** $i \rightarrow 1, 2, \ldots, M$ **do**
10: $\quad \Gamma_i = \frac{2}{\epsilon} \ln \Big(\sum_{r \in R} exp\Big(\frac{\epsilon}{2} \sum_{k \neq i} b_{kr[k]}\Big)\Big) - \frac{2}{\epsilon} \cdot S\Big(EXP_\epsilon^R(\mathbf{b_i}, \mathbf{b_{-i}})\Big) - \underset{r \sim EXP_\epsilon^R(\mathbf{b_i}, \mathbf{b_{-i}})}{\mathbf{E}} \Big[\sum_{k \neq i} b_{kr[k]}\Big]$
11: **end for**

Theorem 1: EXP_ϵ^R is ϵ-differential private and ensures that

$$\mathbf{Pr}\Big[SW_{EXP_\epsilon} < SW_{opt} - \ln \frac{|R|}{\epsilon} - \frac{t}{\epsilon}\Big] \leq exp(-t). \quad (3)$$

for any $t > 0$.

Furthermore, in [19] Huang *et al.* proved the following theorem regarding the incentive compatibility and individual rationality of EXP_ϵ^R .

Theorem 2: With the pricing policy in Line 9-11, the exponential mechanism EXP_ϵ^R is incentive compatible and individual rational.

Both theorems have been proved in an elegant way. For instance, the proof of Theorem 2 relies on the connection between exponential mechanism and a well-known probability measure called the Gibbs measure. Interested readers may refer to [25] [19] for more details.

Although with the appealing feature in satisfying all four requirement in **FCR-Auc** problem, EXP_ϵ^R has an important drawback in that it needs to traverse all $\frac{M!}{(M-N)!}$ feasible allocation outcomes and compute a probability to each of them. As a result, this high computational complexity makes EXP_ϵ^R inapplicable in large-scale FCR systems. Therefore, in the next section, we propose $Auc2Reserve$, a computationally efficient differentially private auction as the solution to the **FCR-Auc** problem.

IV. Auc2Reserve: A Differentially Private Auction

In this section, we propose *Auc2Reserve*, a computationally efficient differentially private auction for EV fast charging reservation system. *Auc2Reserve* adopts an approximate random sampler to iteratively allocate fast charing point to EV user. In each iteration, it applies a belief-propagation-based algorithm for matrix permanent approximation. In this way, *Auc2Reserve* is significantly more computationally efficient than exponential differentially private auction EXP_ϵ^R and other theoretical approximate implementations [19]. We show that *Auc2Reserve* is γ-incentive compatible, individual rational, ϵ-differentially private and provides an explicit guarantee

on social welfare of FCR systems. We also discuss the generalization of *Auc2Reserve* in other scenarios of FCR systems.

A. Preliminary

Before we present the design of *Auc2Charge*, we first review an important concept in linear algebra called *permanent*.

Definition 1: Given a K-by-K matrix $\mathbf{W}$, $\mathbf{W} = (W_{ij}|i,j = 1,2,\ldots,K)$, its permanent is defined as

$$perm(\mathbf{W}) = \sum_{\pi \in \Pi(K)} \prod_{i=1}^{K} W_{i\pi[i]}, \tag{4}$$

where $\Pi(K)$ is the set of all permutations of set $\{1,2,\ldots,n\}$. One important property of permanent we leverage in the design of *Auc2Charge* is that $perm(\mathbf{W}) = perm(\mathbf{W}^T)$, where $\mathbf{W}^T$ is the transpose of $\mathbf{W}$. Permanent has many important applications, i.e., the permanent of a 0-1 matrix is equivalent to the number of perfect matchings of a bipartite undirected graph. However, computing the permanent is a complex problem. The fastest known general algorithm in computing the permanent of matrix is the Ryser Algorithm, which requires $O(n2^n)$ operations. Even for the case of non-negative matrix, i.e., $W_{ij} \geq 0, \forall i,j$, finding its permanent is a #P-complete problem. Therefore, people often resort to different approaches to compute the approximate permanent, denoted as $perm_a$, which is a challenging task as well. To keep the intactness of the presentation, we leave the discussion on how to approximate matrix permanent later till Section IV-C.

B. Auc2Reserve in a NutShell

We present the design of *Auc2Resrve* auction in Algorithm 2. In the **FCR-Auc** problem, the bidding matrix $\mathbf{B}$ is of size M-by-N, where $M > N$. Other than N actual fast charging points, we add another $M - N$ dummy charging points into the auction. And the reported valuation from all EV users on these dummy charging points are all zeros. In this way, *Auc2Reserve* creates a M-by-M square bidding matrix $\mathbf{B_{sq}}$ by concatenating a M-by-$(M-N)$ zero matrix to $\mathbf{B}$ (Line 2). After the transformation, we construct an *auxiliary bidding matrix* D, which is the transpose of $\mathbf{B_{sq}}$ (Line 3). For any square matrix $\mathbf{W}$, we define a function G: $G(\mathbf{W}) = \{exp(\frac{\epsilon}{2}w_{ij})\}, \forall i,j$. And we also use $\mathbf{W}_{-i,-j}$ to denote the matrix obtained by removing the ith row and the jth column of suqare matrix $\mathbf{W}$.

In *Auc2Reserve*, we developed an improved approximate sampler that iteratively allocates charging points to EV users, one at a time. Given an actual charging point $j = 1,2,\ldots,N$, we first compute x_i, the probability that EV user i wins the reservation of charging point j, for all the users who have not won any charging point yet (Line 6-15). *Auc2Reserve* then randomly selects an EV user i with the normalized probability x_i as the winner of charging point j (Line 16). Once a charging point j is allocated to a winning user i_c, all bids submitted to reserve j or submitted by user i_c are removed from the auxiliary matrix $\mathbf{D}$ (Line 19). We then repeat the same allocation process using the updated matrix $\mathbf{D}$ until all N actual charging points are allocated.

The allocation process in *Auc2Reserve* is an item-oriented randomized allocation, i.e., randomly select a user (agent) to receive a given charging point (item). It differs from the original sampler in [19], which uses an agent-oriented randomized allocation, i.e., randomly select an item to assign to a given agent. With the property that $perm(\mathbf{W} = perm(\mathbf{W}^T)$, both allocation policies would yield the same output in expectation. However, given the fact that there are more EV users than fast charging points in the FCR system, i.e., $M > N$, the item-oriented allocation used in *Auc2Reserve* is more computationally efficient in that it reduces the computation overhead by $\frac{M-N}{M}$ times than that of agent-oriented allocation. This reduction is significant due to the involvement of matrix permanent computing during the allocation process.

After an actual fast charging point j is allocated to a winning user i_c, *Auc2Reserve* computes the price i_c needs to pay in Line 18 of Algorithm 2. This pricing policy is an approximated price of that in the exponential mechanism EXP^R_ϵ. After all actual charging points have been assigned, users who do not win any charging point do not need to pay anything (Line 22-26).

Observe the structure of Algorithm 2, we see that the key step in determining the computational efficiency of *Auc2Reserve* is to compute the permanent of non-negative matrix. This step is needed in both allocation and pricing process. Due to the #P-completeness of this task[33] , in the following we review possible approaches in approximating matrix permanent, and apply a newly developed belief propagation approximation technique in *Auc2Reserve*.

Algorithm 2 *Auc2Reserve*: A Differentially Private Auction for **FCR-Auc** Problem

1: INPUT: An M-by-N bidding matrix $\mathbf{B}$
2: $\mathbf{B_{sq}} \rightarrow \mathbf{B}||\mathbf{0}_{M\times(M-N)}$
3: $\mathbf{D} \rightarrow \mathbf{B_{sq}}^T$
4: $I \rightarrow \{1,2,\ldots,M\}$
5: **for** $j \rightarrow 1,2,\ldots,N$ **do**
6: **for** $i \rightarrow 1,2,\ldots,M-j+1$ **do**
7: **if** $d_{1i} == 0$ **then**
8: $x_i \rightarrow 0$
9: **else**
10: $x_i \rightarrow perm_a(G(\mathbf{D}_{-1,-i}))$
11: **end if**
12: **end for**
13: **for** $i \rightarrow 1,2,\ldots,M$ **do**
14: $x_i \rightarrow \frac{x_i}{\sum_i x_i}$
15: **end for**
16: Randomly allocate charging point j to user i with probability x_i, denote the assigned user as i_c
17: $y_{I[i_c]j} \rightarrow 1$
18: $\Gamma_{I[i_c]} \rightarrow b_{I[i_c]j} + \frac{2}{\epsilon}\ln\Big(perm_a\big(G(\mathbf{B}(\mathbf{b_i} = \mathbf{0}, \mathbf{b_{-i}}))\big)\Big) - \frac{2}{\epsilon}\ln\Big(perm_a\big(G(\mathbf{B})\big)\Big)$
19: $\mathbf{D} \rightarrow \mathbf{D}_{-1,-i_c}$
20: $I \rightarrow \{1,\ldots,I[i_c]-1, I[i_c]+1,\ldots,M\}$
21: **end for**
22: **for** $i \rightarrow 1,2,\ldots M$ **do**
23: **if** $\sum_j y_{ij} == 0$ **then**
24: $\Gamma_i \rightarrow 0$
25: **end if**
26: **end for**

C. Approximating Matrix Permanent Using Belief Propagation

Researchers have explored different approaches in developing approximate algorithms for matrix permanent. In the

seminal paper [20], Jerrum *et al.* proposed a fully polynomial randomized approximation scheme (FPRAS) for computing the permanent of non-negative matrix using Monte-Carlo sampling. However, its complexity is $O(n^{11})$ in the general case. Even though the complexity of this approach was later improved to $O(n^7)$ in [6], it is still impractical for most realistic applications.

Recently, belief propagation heuristics is applied to computing the matrix permanent and showed surprisingly good performance[11][12]. The basic idea of this approach is that given a non-negative K-by-K matrix $\mathbf{W}$, we can build a graphical model with a bipartite undirected graph $G = (V_1, V_2, E)$. In this graph, V_1 and V_2 are two sets of K vertices. And binary variables $\sigma_{ij} = 0, 1$ are assigned to edges (i, j) with the constraints $\forall i \in V_1, \sum_{j \in V_2} \sigma_{ij} = 1$ and $\forall j \in V_2, \sum_{i \in V_1} \sigma_{ij} = 1$. And W_{ij} is assigned as the weight of edge (i, j) in graph G. In this way, the graphic model can express the permanent of matrix W as:

$$perm(\mathbf{W}) = \sum_{\sigma} \prod_{(i,j) \in E} (W_{ij})^{\sigma_{ij}}. \tag{5}$$

With this graphical model, we define a K-by-K matrix β where β_{ij} is the marginal belief corresponding to finding edge (i, j) in the matching of G. And the following theorem on BP-based permanent approximation was proposed in [11].

Theorem 3: Given a non-negative K-by-K matrix $\mathbf{W}$ with corresponding graph G and marginal belief matrix β. A belief propagation function can be developed as $perm(\mathbf{W})$ as

$$F_{BP}(\beta|\mathbf{W}) = \sum_i \sum_j \Big(\beta_{ij} \log(\frac{\beta_{ij}}{W_{ij}}) - (1-\beta ij) \log(1-\beta_{ij})\Big), \tag{6}$$

and the permanent of $\mathbf{W}$ can be approximated as:

$$perm_{BP}(\mathbf{W}) = exp\Big(- F_{BP}(\beta|\mathbf{W})\Big). \tag{7}$$

To compute the marginal belief matrix β, standard belief propagation technique can be applied, and an iterative algorithm is developed and presented as Algorithm 3. The convergence of this algorithm is proved in [11]. However, its convergence speed highly depends on the initial value of matrix $\beta(0)$. To ensure a fast convergence and hence the computational efficiency of *Auc2Charge*, we use a mean-field heuristic algorithm, presented as Algorithm 4, to precompute a matrix ϕ as the initial value $\beta(0)$. Extensive empirical experiments in [11] [12] show that using ϕ computed by Algorithm 4 as the initial value of marginal belief matrix β ensures the fast convergence of Algorithm 3 with arbitrary input matrix $\mathbf{W}$. In addition, the following Theorem 4 provides a close-form approximation ratio on this BP-based method [12].

Theorem 4: Given a K-by-K matrix $\mathbf{W}$, compute the doubly stochastic matrix β using Algorithm 3, and define

$$\alpha_{\mathbf{W}} = \frac{\prod_{i,j}(1 - \beta_{ij})}{perm\big(\beta \cdot (1 - \beta)\big)}. \tag{8}$$

Then the approximate permanent computed from Theorem 3 satisfies:

$$perm_{BP}(\mathbf{W}) = \alpha_{\mathbf{W}} \cdot perm(\mathbf{W}). \tag{9}$$

To summarize, by transforming a given matrix $\mathbf{W}$ to a graphical model and applying belief propagation technique to computing its marginal belief matrix, $perm(W)$ can be approximated very efficiently with a close-form approximation ratio. We leave the theoretical proof on convergence speed of this BP approach in future work. In this way, *Auc2Reserve* is significantly more computationally efficient than the generic exponential differentially private mechanism EXP^R_ϵ and other theoretical approximate implementations[19].

Algorithm 3 An Iterative Algorithm for Computing Marginal Belief Matrix β

INPUT: a K-by-K non-negative matrix $\mathbf{W}$
$n \to 0$
$u_i(0) = u^j(0) = 1, \forall i, j \to 1, 2, \ldots, K$
Initialize $\beta_{ij}(0)$, $\forall i, j \to 1, 2, \ldots, K$
while 1 **do**
  for $i, j \to 1, 2, \ldots K$ **do**
    $\beta_{ij}(n+1) \to \lambda \beta_{ij}(n) + \frac{(1-\lambda) W_{ij}}{W_{ij} + (\frac{1}{2}\sum_s \beta_{is}(n) + \frac{1}{2}\sum_s \beta_{sj}(n) - \beta_{ij}(n))^2 u_i(n) u^j(n)}$
  end for
  for $i, j \to 1, 2, \ldots K$ **do**
    $\beta_{ij}(n) \to \beta_{ij}(n) / \sum_s \beta_{is}(n)$
  end for
  for $i, j \to 1, 2, \ldots K$ **do**
    $\beta_{ij}(n) \to \beta_{ij}(n) / \sum_s \beta_{sj}(n)$
  end for
  for $i \to 1, 2, \ldots K$ **do**
    $u_i(n+1) \to \frac{\sum_s (W_{is}/u^s(n))}{1 - \sum_j (\beta_{ij}(n))^2}$
  end for
  for $j \to 1, 2, \ldots K$ **do**
    $u^j(n+1) \to \frac{\sum_s (W_{sj}/u_s(n))}{1 - \sum_i (\beta_{ij}(n))^2}$
  end for
  if $max_{i,j}\{|\beta_{ij}(n+1) - \beta_{ij}(n)|\} > \delta$ **then**
    $n \to n + 1$
  else
    for $i, j \to 1, 2, \ldots K$ **do**
      $\beta_{ij} \to \beta_{ij}(n)$
    end for
    break
  end if
end while

D. Properties of Auc2Reserve

We now analyze the performance of *Auc2Reserve* auction in terms of incentive compatibility, individual rationality, differential privacy and social welfare. To this end, we first propose the following theorem.

Theorem 5: Auc2Reserve is an ϵ-differential private and individual rational.

Proof 1: The proof of this theorem follows the sketch in [25] [19]. It relies on the connection between the exponential mechanism and the well-known Gibbs measure, which is also known as the Boltzmann distribution. With the connection established in [19], the correctness of this theorem is a straightforward conclusion.

Next we study the incentive compatibility and social welfare of *Auc2Reserve*. Given a matrix $\mathbf{W}$, we define $\alpha^{max}_{\mathbf{W}} = max_{s(\mathbf{W})}\{\alpha_{s(\mathbf{W})}\}$, where $s(\mathbf{W})$ is any sub-matrix of $\mathbf{W}$. Then we are able to propose the following theorem.

Theorem 6: Given any M-by-N bidding matrix $\mathbf{B}$, the Auc2Reserve mechanism is γ-incentive compatible and the

Algorithm 4 A Mean-Field Algorithm for Initializing Marginal Belief Matrix $\beta(0)$

1: INPUT: a K-by-K non-negative matrix $\mathbf{W}$
2: $n \to 0$
3: $v_i(0) = v^j(0) = 1, \forall i, j \to 1, 2, \ldots, K$
4: $\phi_{ij}(0) = 1, \forall i, j \to 1, 2, \ldots, K$
5: **while** 1 **do**
6: **for** $i, j \to 1, 2, \ldots K$ **do**
7: $\phi_{ij}(n+1) \to \frac{W_{ij}}{W_{ij} + v_i(n)v^j(n)}$
8: **end for**
9: **for** $i \to 1, 2, \ldots K$ **do**
10: $v_i(n+1) \to v_i(n) \sum_j \phi_{ij}(n)$
11: **end for**
12: **for** $j \to 1, 2, \ldots K$ **do**
13: $v^j(n+1) \to v^j(n) \sum_i \phi_{ij}(n)$
14: **end for**
15: **if** $max_{i,j}\{|\phi_{ij}(n+1) - \phi_{ij}(n)|\} > \delta$ **then**
16: $n \to n+1$
17: **else**
18: **for** $i, j \to 1, 2, \ldots K$ **do**
19: $\beta_{ij}(0) \to \phi_{ij}(n)$
20: **end for**
21: **break**
22: **end if**
23: **end while**

social welfare computed by Auc2Reserve satisfies:

$$\mathbf{Pr}\left[SW < SW_{opt} - M\gamma - \frac{M!}{\epsilon(M-N)!} - \frac{t}{\epsilon}\right] \leq exp(-t), \quad (10)$$

where $\gamma = M \ln(\alpha^{max}_{G(\mathbf{B_{sq}})})$ and $\mathbf{B_{sq}}$ is defined in Line 2 of Algorithm 2, for any $t > 0$.

Proof 2: The basic idea in this proof is that according to Bayes' rule and the BP-based permanent approximation ratio in Theorem 4, in every iteration of Algorithm 2, the probability that charging point j is allocated to an user i approximates the correct exponential allocation distribution by a multiplicative factor of $\alpha^{max}_{G(\mathbf{B})}$. So after allocating all M fast charging points, the probability that we sample an allocation outcome $r \in R$ differs from the correct exponential allocation distribution by a multiplicative factor of $(\alpha^{max}_{G(\mathbf{B_{sq}})})^M$. Then the correctness of this theorem can be achieved by directly applying the results in [25] [19]. To begin with, we first prove the following lemma:

Lemma 1: Auc2Reserve yields an outcome $r \in R$ differing from the correct distribution by at most $(\alpha^{max}_{G(\mathbf{B_{sq}})})^M$ factor.

We use $j \to i$ to denote that charing point j is assigned to user i. In the exponential mechanism EXP^R_ϵ, for any allocation outcome $r \in R$, using the Bayes' rule, the probability r is chosen as the final allocation outcome can be expressed as follows:

$$\begin{aligned}\mathbf{Pr}[EXP^R_\epsilon(\mathbf{b}) = r] \quad &= \mathbf{Pr}[\text{point 1 is assigned to } r^{-1}[1]] \\ &\cdot \mathbf{Pr}[\text{point 2 is assigned to } r^{-1}[2] | 1 \to r^{-1} \\ &\ldots \cdot \mathbf{Pr}[\text{point } N \text{ is assigned to } r^{-1}[N] \\ &| 1 \to r^{-1}[1], \ldots, N-1 \to r^{-1}[N-1]]. \end{aligned} \quad (11)$$

Take the first iteration of our *Auc2Reserve* algorithm as start, we use the distribution

$$\mathbf{Pr}[\text{point 1 is assigned to user } i] = x_i \propto perm_a(G(\mathbf{D}_{1,i})).$$

In the exponential mechanism EXP^R_ϵ,

$$\mathbf{Pr}[\text{point 1 is assigned to user } i] \propto exp(\frac{\epsilon}{2} b_{i1}) perm_a(G(\mathbf{D}_{1,i}))$$

Because x_i approximates $perm_a(G(\mathbf{D}_{1,i}))$ by up to an multiplicative factor of $\alpha^{max}_{G(\mathbf{B_{sq}})}$, we know that the probability that user i gets charging point 1 in *Auc2Reserve* approximates the correct marginal in EXP^R_ϵ up to an $\alpha^{max}_{\mathbf{B_{sq}}}$ multiplicative factor. Not only for the first iteration, we can find this $\alpha^{max}_{G(\mathbf{B_{sq}})}$ multiplicative factor holds for the remaining iterations in *Auc2Reserve*. Therefore, the probability that *Auc2Reserve* yields an outcome $r \in R$ differing from the correct distribution by at most $(\alpha^{max}_{G(\mathbf{B_{sq}})})^M$ factor.

Having proved this lemma, the correctness of this theorem can be achieved by directly applying the results in [25] [19].

Theorem 6 provides an upper bound on the social welfare of *Auc2Reserve*. We observe that this bound decreases as the number of EV users increases, and increases as ϵ and N increases. As the differential privacy factor, a smaller ϵ is the indicator of a stronger differential privacy. Therefore, this upper bound indicates that there exists an explicit tradeoff between maximizing social welfare and providing stronger differential privacy for more EV users. Theorems 5 and 6 together show that *Auc2Reserve* satisfies all four requirements in the **FCR-Auc** problem. Compared to the exponential mechanism EXP^R_ϵ, *Auc2Reserve* achieves a much higher computationally efficiency via a tradeoff on an additional γ factor in incentive compatibility and an additional $M\gamma$ factor in social welfare.

E. Generalization of Auc2Reserve

We design *Auc2Reserve* under the scenario where all EV users submit bids to reserve fast charging points for the same future time slot. In fact, *Auc2Reserve* can be applied to other generalized scenarios of FCR systems. One scenario is that future time slots are to be available for reservation one by one. Then the central controller in FCR system simply needs to execute *Auc2Reserve* for every future time slot after it becomes available for reservation. Another scenario is that the administrator opens the reservation for N fast charging points at $T > 1$ different time slots at one time, and make the allocation and pricing decisions all at once. In this case, the central controller can construct a bidding matrix of size M-by-NT and then execute *Auc2Reserve* using this matrix as the input. If $M > NT$, *Auc2Reserve* will use the same item-oriented allocation process. If $M \leq NT$, an agent-oriented allocation process will be applied. In both scenarios, *Auc2Reserve* is able to make fast charging point allocation and pricing decisions in a computationally efficient manner, i.e., providing an explicit guarantee on social welfare of FCR system, and ensure γ-incentive compatibility, individual rationality and ϵ-differential privacy simultaneously.

V. Performance Evaluation

[1]] In this section, we demonstrate the efficiency of our proposed *Auc2Reserve* mechanism for EV fast charging reservation via numerical simulation.

Methodology. In our simulation, we assume a virtual traffic network where M electric vehicles and N DC fast charging

points are randomly distributed. Because EV users usually want to get fast charging at certain locations, they would only submit bids to a limited number of fast charging points. Therefore, in the simulation, we assume that every EV user will submit at most 8 bids in the form of *(valuation, charging point)* to compete for reservation at these fast charging points. We assume that the budget, i.e., the maximal valuation, of each EV user, follows a uniform distribution between 8 and 12 dollars. For every EV user j, the *valuation* and *charging point* in her bids are separately randomly generated. In our simulation, we set the parameters λ and δ in the BP-based permanent approximation algorithm as 0.7 and 0.1, respectively. We perform simulation of *Auc2Reserve* under the settings of $M = 60, 70, 80, 90, 100$ EVs and $N = 10, 20, 30, 40$ fast charging points. And we set the privacy parameter ϵ in *Auc2Reserve* to be 0.1 and 0.5. For each combination of M, N and ϵ, we repeat the simulation for 20 times and compute the average value.

Results. In what follows, we evaluate the performance of *Auc2Charge* on both social welfare and user privacy. Figure 2 shows the social welfare achieved by *Auc2Reserve* under different numbers of EVs when the number of fast charging points is fixed at 40 in the FCR system. We first see that the social welfare of *Auc2Reserve* with $\epsilon = 0.1$ is smaller than that with $\epsilon = 0.5$ at all cases. This is because a smaller ϵ represents a stronger differential privacy of EV users. And as pointed in Theorem 6, this stronger is achieved through a trade of social welfare loss. The smaller ϵ is, the higher this loss becomes. An interesting observation we find is that when ϵ and number of charging points are fixed, the social welfare of *Auc2Reserve* decreases as the number of EVs increases. The reason of this observation is as follows. In our simulation, the reservation bids submitted by EV users are generated following the same distribution. Thus when the number of fast chargers are fixed, the optimal social welfare SW_{opt} should also be a fixed value in expectation, and independent from the number of EV users. However, when there are more EVs participating the auction, *Auc2Reserve* ensures the differential privacy of EV users through a higher tradeoff of social welfare loss. This observation is consistent with the social welfare bound of *Auc2Reserve* in Equation (10) from Theorem 6. From this equation we see that even if SW_{opt} is fixed, both $M\gamma$ and $\frac{M!}{(M-N)!}$ increase as M increases, which leads to the decrease of upper bound on *Auc2Reserve*'s social welfare.

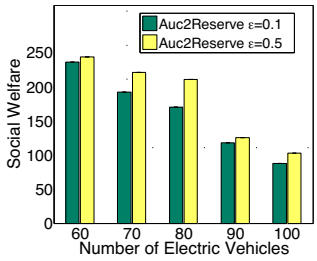


Fig. 2: Social Welfare with $N = 40$ Fast Charging Points

We then plot the social welfare of *Auc2Reserve* under different number of fast chargers when there are 100 EVs in the FCR system in Figure 3. We also observe that the social welfare when $\epsilon = 0.5$ is higher than that of $\epsilon = 0.1$ due to the stronger differential privacy of the latter. And opposite from the trend of varying EV numbers, *Auc2Reserve* yields a higher social welfare when there are more fast charging points in the system. This is because with more resources in the auction, there are more EV users being allocated a fast charger when executing *Auc2Reserve*. And it is also consistent with Equation (10), where the social welfare loss $\frac{M!}{(M-N)!}$ decreases as the number of fast chargers N increases.

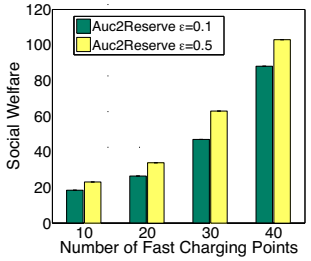


Fig. 3: Social Welfare with $M = 100$ Electric Vehicles

Other than social welfare, we also use the recent proposed notion *privacy leakage* [37] [36] to evaluate the efficacy of *Auc2Charge* in protecting user privacy.

Definition 2: (*Privacy Leakage*[36]) Let $\vec{a}$ and $\vec{a'}$ be probability distributions over a price set P for bidding matrix $\mathbf{B}$ and $\mathbf{B}'$, which only differ in one single bid, respectively. The *privacy leakage* between the two bidding matrices is the maximum of absolute differentces between the logarithmic probabilities of the two distributions, i.e.,

$$max_i |\ln a_i - \ln a'_i|. \tag{12}$$

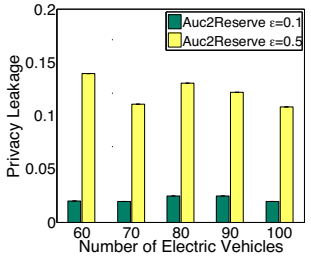


Fig. 4: Privacy Leakage with $N = 40$ Fast Charging Points

Given an auction, a smaller privacy leakage implies that when there is one arbitrary EV user changes one of her bids in the FCR system, the probability distribution over pricing decisions made by this auction would only have a small change as well, which proves the differential privacy of this auction.

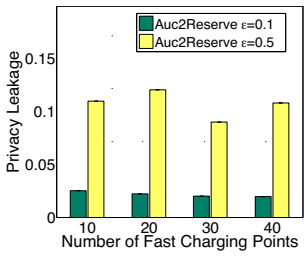
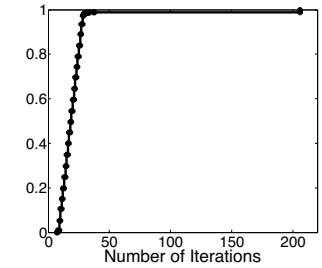


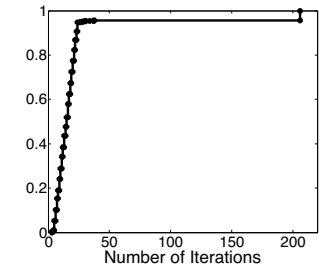
Fig. 5: Privacy Leakage with $M = 100$ Electric Vehicles

Figure 4 plot the privacy leakage of *Auc2Reserve* under different number of EVs with 40 fast chargers in the FCR

system. We observe that when $\epsilon = 0.1$, the privacy leakage of *Auc2Reserve* is less than 0.02. And when $\epsilon = 0.5$, this leakage is less than 0.15. This observation implies that it is almost impossible to for an adversary to infer the personal information of EV users, e.g., charging location preference. When fixing the number of EVs as 100, from Figure 5 we also have a similar observation on the privacy leakage of *Auc2Reserve* under different number of fast chargers. Therefore, *Auc2Reserve* is efficient in ensuring EV users' privacy.



(a) Convergence of Algorithm 3 ($\epsilon = 0.1$)



(b) Convergence of Algorithm 3 ($\epsilon = 0.5$)

Fig. 6: Computational Efficiency of *Auc2Reserve* with $M = 100$ Electric Vehicles

Furthermore, we demonstrate the computational efficiency of *Auc2Reserve* by plotting the cumulative distribution function on number of iterations in Algorithm 3 when it converges. It is shown in Figure 6 that in *Auc2Reserve*, the permanent of bidding matrices can be approximated very fast under different values of ϵ, i.e., converging within 30 iterations in 99% cases. With this fast convergence, *Auc2Reserve* is highly computationally efficient in making fast charger allocation and pricing decisions for FCR systems.

VI. Related Work

EV Charging Facilities. EV charging facilities are indispensable infrastructure of both intelligent transportation systems and smart grid, and have drawn great attention from both academia and industry. There has been a growing literature on various EV charging facilities [23], [10], [5], [17], [9], [4], [1], [2]. Ardakanian *et al.* [5] designed a distributed charging algorithm to adjust EV charging rate for residential chargers. Lopes *et al.* [23] designed a framework to integrate EVs into power system. Chen *et al.*[10] designed a central controller to schedule the EV charging using renewable energy. Chen *et al.* [9] studied a joint optimal power flow and EV charging problem, and built an online controller to enable efficient EV charging. Jin *et al.* [21] built a stochastic optimization framework to minimize the cost of single charging station.

Recently, the fast charging reservation (FCR) system, an innovative charging facility, was developed and has drawn special attention form both academia and industry. In this system, EV users can reserve DC fast chargers at different locations ahead of time. which charge the battery of EV to 80% capacity within 30 minutes. The FCR system facilitate people to charge EVs during a long distance trip with a short time delay, and thus are welcomed by EV users. In major automobile markets, several FCR systems have been developed. Tesla has deployed a Supercharger network with over 400 Supercharger stations across the United State s[4]. BMW and ChargePoint develop the ChargeNow program, in which BMW EV users can reserve public fast chargers via mobile devices [1]. China has initiated a project to develop a fast charging reservation system with over 600 fast chargers along major highways across the country by 2020 [2]. And over 100 fast chargers have been deployed by early 2015. In FCR systems, there are usually more EVs than fast charging points. And recent studies [3] [35] show that fast chargers are in fact the most scarce resource in FCR systems, instead of the commonly assumed electricity. Thus *how to allocate fast chargers* between EV users requires careful study.

Auction Theory. Auction allocates resources to buyers who value them most, reduces the chance of overpricing and underpricing, and thus improves social welfare. It has been widely used in Internet advertisement [14], wholesale electricity market [30] and cloud computing [31]. Recently researchers propose to utilize auction to improve resource allocation efficiency for EV charging, and different auctions are designed for different scenarios [18], [34], [29]. Gerding *et al.* [18] proposed a two-side truthful online auction with advanced reservation, in which EV users and the charging station can exchange their charging preference and cost. Robu *et al.* [29] designed an online mechanism, in which EV users bid for different charging speeds based on their arrival time, and cancel the charging allocation on departure. Xiang *et al.* [34] proposed an online auction framework for EV park-and-charge. However, these auctions achieve social welfare maximization by incentivizing EV users to truthfully report their valuation on different set of resources, e.g., electricity and charging points, putting EV users at the risk of exposing their privacy. Adversaries may use these real valuations to infer EV users' personal information. And this inference becomes even easier when EV users participate charging auctions frequently. Therefore, a privacy-preserving auction is desired to protect the privacy of EV users against such inference.

McSheery *et al.* first proposed to use differentially private mechanisms in auction design in [25]. They showed that differential privacy implies approximate incentive compatibility, and designed exponential mechanism for differentially private digital auctions and attribute auctions. Huang *et al.* [19] instantiate the principle in [25], and develop an approximate implementation for generic differentially private auctions. Though the implementation has a polynomial-time complexity, i.e., $O(n^{13})$. It is impractical in real-world as the implied constant in the O function is very large. Zhu *et al.* designed differentially private mechanisms for spectrum auction in [37] [36]. The proposed algorithms leverage unique characteristics in spectrum auctions and achieves approximate revenue maximization. In our *Auc2Reserve*, we developed an improved approximate sampler for fast charger allocation. To accelerate the allocation, we applies the belief-propagation technique in matrix permanent approximation. Therefore, not only is *Auc2Reserve* γ-incentive compatible, individual rational, ϵ-differentially private, it also makes fast charging

points allocation and pricing decisions with a close-form social welfare guarantee in a computational efficient way. To the best of our knowledge, *Auc2Reserve* is the first differentially private auction for EV fast charging reservation systems.

VII. Conclusion

The FCR system allows EVs to send requests to reserve DC fast chargers ahead of time. In this system, the allocation of fast charging points requires careful design because they are the most scarce resource instead of electricity. Not only charging points should be allocated to EV users who really values them, the allocation and the corresponding pricing policies should also prevent users' private information from being inferred. In this paper, we explore the feasibility and benefits of differentially private auction in FCR systems. We design *Auc2Reserve*, a differentially private randomized auction for FCR system. *Auc2Reserve* applies the belief propagation technique to accelerate the randomized allocation process. Thus it is significantly more computationally efficient than generic differentially private mechanisms. To the best of our knowledge, we are the first to apply belief-propagation in designing computationally efficient differentially private mechanisms. *Auc2Reserve* is γ-incentive compatible, individual rational, ϵ-differentially private and provides an explicit approximation ratio on the social welfare of FCR systems. Using simulation, we further demonstrate the efficiency of *Auc2Reserve* on social welfare and privacy leakage under various settings of FCR systems. In future work, we plan to extend the *Auc2Reserve* mechanism by including other realistic constraints in both the electricity market, e.g., electricity allocation, vehicle-to-grid transmission and ramp-up/ramp-down cost of electricity generation, and intelligent transportation systems, e.g., the uncertainty of EV's mobility. We will also explore the feasibility of computationally efficient differentially private auctions for other charging facilities, e.g., EV park-and-charge lot.

VIII. Acknowledgment

We would like to thank Xi Chen, Fanxin Kong, Yuwei Xu and Rui Zhang for their insightful discussions. The authors' work are supported in part by the NSERC Collaborative Research and Development Grant (CRDJP 418713), CFI Leaders Opportunity Fund 23090, NSFC Grant 61303202 and NSFC Tianjin Research Grant (16JCQNJC00700).

References

[1] The bmw chargenow program. www.chargenow.com.

[2] China plan to develop fast charging network across country. www.ft.com/intl/cms/s/0/1b26d892-9d66-11e4-8946-00144feabdc0.html.

[3] The economics of ev charging stations. https://energyathaas.wordpress.com/2015/03/16/the-economics-of-ev-charging-stations/.

[4] The tesla supercharger. www.teslamotors.com/supercharger.

[5] O. Ardakanian, C. Rosenberg, and S. Keshav. Distributed control of electric vehicle charging. In *ACM e-Energy*, 2013.

[6] I. Bezáková, D. Štefankovič, V. V. Vazirani, and E. Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. In *the 17th ACM-SIAM symposium on Discrete algorithm*.

[7] S. Bhattacharya, G. Goel, S. Gollapudi, and K. Munagala. Budget constrained auctions with heterogeneous items. In *STOC 2010*.

[8] S. Chawla, J. D. Hartline, D. Malec, and B. Sivan. Sequential posted pricing and multi-parameter mechanism design. In *STOC 2010*.

[9] N. Chen, C. W. Tan, and T. Quek. Electric vehicle charging in smart grid: Optimality and valley-filling algorithms. *Technical Report*, 2014.

[10] S. Chen and L. Tong. iems for large scale charging of electric vehicles: Architecture and optimal online scheduling. In *IEEE SmartGridComm*, 2012.

[11] M. Chertkov, L. Kroc, F. Krzakala, M. Vergassola, and L. Zdeborová. Inference in particle tracking experiments by passing messages between images. *Proceedings of the National Academy of Sciences*, 2010.

[12] M. Chertkov and A. B. Yedidia. Approximating the permanent with fractional belief propagation. *J. Mach. Learn. Res.*, 2013.

[13] C. Chung, J. Chynoweth, C. Qiu, C. Chu, and R. Gadh. Design of fair charging algorithm for smart ev charging infrastructure. In *2013 IEEE ICTC*.

[14] N. R. Devanur and T. P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *ACM EC'09*.

[15] C. Dwork. Differential privacy. In *ICALP 2006*.

[16] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(34):211–407, 2013.

[17] L. Gan, U. Topcu, and S. Low. Optimal decentralized protocol for electric vehicle charging. *Power Systems, IEEE Transactions on*, 28(2):940–951, 2013.

[18] E. H. Gerding, S. Stein, V. Robu, D. Zhao, and N. R. Jennings. Two-sided online markets for electric vehicle charging. In *AAMAS'13*.

[19] Z. Huang and S. Kannan. The exponential mechanism for social welfare: Private, truthful, and nearly optimal. FOCS '12.

[20] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM (JACM)*, 51(4):671–697, 2004.

[21] C. Jin, X. Sheng, and P. Ghosh. Energy efficient algorithms for ev charging with intermittent renewable energy sources. In *IEEE PES'13*.

[22] R. J. Lipton, V. V. Vazirani, M. Mihail, C. Tovey, and E. Vigoda. Algorithmic game theory. 2007.

[23] J. A. P. Lopes, F. J. Soares, and P. M. R. Almeida. Integration of electric vehicles in the electric power system. *Proceedings of the IEEE*, 2011.

[24] Z. Ma, D. S. Callaway, and I. A. Hiskens. Decentralized charging control of large populations of plug-in electric vehicles. *Control Systems Technology, IEEE Transactions on*, 2013.

[25] F. McSherry and K. Talwar. Mechanism design via differential privacy. FOCS '07.

[26] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC 2009*.

[27] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Security and Privacy*.

[28] M. Pan, J. Sun, and Y. Fang. Purging the back-room dealing: Secure spectrum auction leveraging paillier cryptosystem. *Selected Areas in Communications, IEEE Journal on*, 29(4):866–876, 2011.

[29] V. Robu, E. H. Gerding, S. Stein, D. C. Parkes, A. Rogers, and N. R. Jennings. An online mechanism for multi-unit demand and its application to plug-in hybrid electric vehicle charging. *JAIR*, 2013.

[30] G. B. Sheblé. *Computational auction mechanisms for restructured power industry operation*. springer, 1999.

[31] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. Lau. An online auction framework for dynamic resource provisioning in cloud computing. In *SIGMETRICS'14*.

[32] S. Stein, E. Gerding, V. Robu, and N. R. Jennings. A model-based online mechanism with pre-commitment and its application to ev charging. In *AAMAS'12*.

[33] L. G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[34] Q. Xiang, F. Kong, X. Chen, L. Kong, X. Liu, and L. Rao. Auc2charge: An online auction framework for electric vehicle park-and-charge. In *ACM e-Energy 2015*.

[35] Q. Xiang, F. Kong, X. Liu, X. Chen, L. Kong, and L. Rao. Bid to charge: Exploring auction design for electric vehicle charging stations. *Technical Report, McGill University*, 2015.

[36] R. Zhu, Z. Li, F. Wu, K. Shin, and G. Chen. Differentially private spectrum auction with approximate revenue maximization. MobiHoc '14.

[37] R. Zhu and K. G. Shin. Differentially private spectrum auction with approximate revenue maximization. In *INFOCOM'15*.

Update Algebra: Toward Continuous, Non-Blocking Composition of Network Updates in SDN

Geng Li*†, Y. Richard Yang*, Franck Le‡, Yeon-sup Lim‡, Junqi Wang§
*Yale University, USA, †Tongji University, China, ‡IBM T.J. Watson Research Center, USA, §Rutgers University, USA

***Abstract*—The ability to support continuous network configuration updates is an important ability for enabling Software Defined Networks (SDN) to handle frequent or bursty changes. Current solutions for updating SDN configurations focus on one single update at a time, leading to slow, sequential (*i.e.*, blocking) update execution. In this paper, we develop update algebra, a novel, systematic, theoretical framework based on abstract algebra, to enable continuous, non-blocking, fast composition of multiple updates. Specifically, by modeling each data-plane operation in the set of data-plane operations to be executed by an update as a *set-theoretical projection*, update algebra defines novel *operation composition* so that the number of projections for the same match remains constant regardless of the number of updates to be composed, leading to substantial performance benefits. Specifying the dependencies of the data-plane operations in updates as a subset of a *free monoid* in the general case and as partial ordering for *basic consistency*, update algebra defines *update composition* that preserves consistency, even under partially-executed updates, to guarantee correctness. We conduct asymptotic analysis, extensive benchmarking using a real controller, and integration with a real application to demonstrate the benefits of update algebra. In particular, our asymptotic analysis demonstrates that in independent-update dominant settings, update completion time of update algebra remains asymptotically constant despite growth of the number of updates to be executed. Our benchmarking shows that update algebra can achieve 16x reduction in update latency even in settings with an update arrival rate of only 1.6/s. Our integration with Hedera, a real SDN traffic engineering application, shows that update algebra can reduce average link bandwidth utilization by 30% compared with sequential updates.**

I. Introduction

The ability to provide continuous, rapid, non-blocking network configuration updates is an essential capability for Software Defined Networking (SDN). First, it provides a foundation for the development of advanced applications with frequent network updates, which are typically prohibited or discouraged in current SDN systems. A recent trend is the application of machine learning to continuously and rapidly adapt routing strategies to minimize maximum link utilization, achieve proportional fairness, or maximize other objectives [1]–[3]. In addition, with studies having revealed that traffic in several settings (*e.g.*, data centers) is highly dynamic, many solutions are advocating switching traffic at a finer granularity than flows, including subflows, and flowlets, to reduce congestion, and optimize path choices more frequently [4]–[6]. These trends and applications emphasize the need for controllers to support continuous, rapid, non-blocking network configuration updates. Second, a network may experience a set of rapid bursts of changes, causing an SDN controller to receive and handle a batch of network configuration updates [7]. For example, such updates may stem from the occurrence of unpredictable events including outages, Denial of Service attacks, BGP re-routes, or flash crowds.

Although a large amount of research effort has recently been devoted to developing efficient algorithms to update forwarding rules in SDN, existing solutions are unsatisfactory, and do not provide the required capability to handle frequent or short sudden groups of network changes. This is because despite having developed algorithms reduce the numbers of changes, or minimize the network update completion times [8], or preserve a range of properties [9]–[12] including loop and blackhole freedom, existing solutions focus on one single network configuration update at a time. In other words, they execute consecutive received network updates individually, and sequentially in a blocking manner [13]. Similar to the development of pipelining executions of instructions that has led to fundamental changes and performance improvements in computer architecture, the ability to execute continuous and non-blocking updates can lead to significant potential improvements in SDN control architecture [14].

Achieving continuous, non-blocking network updates is not straightforward. The first challenge is to guarantee correctness; a naïve execution may lead to unnecessary blocking due to dependencies on updates. To illustrate this, consider two consecutive updates: a first update U_1 sets the route for a flow, and a following update U_2 changes part of the route. Therefore, in a straightforward execution, the execution of U_2 cannot start until U_1 is finished, leading to a sequential (*i.e.*, blocking execution) update model. The second challenge is that a network configuration update often consists of multiple operations that must be executed at different switches in a specific order to guarantee properties, such as blackhole freedom and waypoint routing (*e.g.*, to traverse a sequence of VNFs) [12]. When executing consecutive updates in a non-blocking manner, these properties must also be preserved. The third challenge is that because network configuration updates often consist of multiple operations, updates do not operate atomically, and when a new update arrives, previous updates may be mid-execution. The execution status may even be unknown to the controller due to the fundamentally distributed nature of the SDN system.

In this paper, we develop a novel, systematic, and foundational theoretical framework based on abstract algebra to reason about and support continuous, non-blocking updates. The framework is motivated by the following two insights. First, when handling multiple updates (*i.e.*, multiple batches of operations), operations on the same flow rules from consecutive updates may be replaced by fewer equivalent ones. For example, the creation of a flow rule followed by its modification can be replaced by the creation of an updated rule. To realize this insight, we model each operation as a *set-theoretical projection*, which provides flexible composition

between operations. As such, the first challenge can be adequately addressed. Second, an update can be represented by a set of feasible sequences of operations whose order ensures the desired properties, and composition of multiple updates can be modeled as the application of different mathematical operations on these sequences. In abstract algebra, such a model can be well defined by a *free monoid* (of which a typical example is words with letters). By modeling each update as a subset of a *free monoid* on a set of projections, we can take advantage of the algebraic properties (*i.e.*, associativity, idempotence, selectivity, commutativity) of the structure to identify equivalent operation sequences that preserve correctness and consistency properties, whether the former updates have been completely or partially executed. This insight addresses the second and third challenges.

We conduct asymptotic analysis, extensive benchmarking using a real controller, and integration with a real application to demonstrate the benefits of update algebra. The asymptotic analysis demonstrates that in independent-update dominant settings, the completion time with the existing sequential execution grows linearly, while that of the update algebra remains asymptotically constant. The benchmarking results show that update algebra can achieve 16x reduction in update latency even in settings with an update arrival rate of only 1.6/s Finally, the integration with a real application shows that by applying update algebra, SDN Traffic Engineering applications (*e.g.*, Hedera [2]) can reduce the average link bandwidth utilization by 30% compared to sequential execution.

II. Basic Models

This section introduces the basic models to represent the network data plane configuration and individual network updates. The update algebra framework for the continuous, non-blocking composition of consecutive network updates will be specified in Section III. TABLE I summarizes the main variables and notations used in the following content.

TABLE I: Terminologies

$SW = \{sw\}$	the set of switches (forwarding tables)
$M = \{m\}$	the set of all possible match values
$PR = \{pr\}$	the set of priorities
$key = (key.m, key.pr)$	flow rule key, defined by a match value endowed with a priority
$KEY = \{key\} = M \times PR$	the set of 2-tuple flow rule keys
$AC = \{ac\}$	the set of all possible actions
$AC^+ = AC \cup \{Null\}$	the action set with a null value
$C : SW \times KEY \rightarrow AC^+$	data plane configuration
$o(sw, key, ac) : C \rightarrow C'$	data plane operation
$u = o_1 o_2 ... o_k$	update representative
$U = \{u_1, u_2, ...\} : C \rightarrow C'$	data plane update
$O(U)$	constituent operations of U
$\Omega(U)$	order constraint of update U

A. Data Plane Configurations

An SDN is comprised of a set of switches SW. A data plane configuration C consists of a collection of flow rules that determine the packets' forwarding states in the network. A flow rule defines an action $ac \in AC$ for flows matching a $key \in KEY$ at switch $sw \in SW$. A key therefore has two attributes: (1) the matching criteria ($key.m$), and (2) a priority ($key.pr$). Equation (1) illustrates four flow rule keys.

		$SW = \{A, B, C, E\}$			
		A	**B**	**C**	**E**
$KEY = \{key_1, key_2\}$	key_1	$fwd\_B \rightarrow fwd\_E$	fwd_C	fwd_D	$Null$
	key_2	$Null$	$Null$	$Null$	$Null$

Fig. 1: Illustration of a data plane configuration C with four switches and two keys. A data plane operation $o(A, key_1, fwd\_E) : C \rightarrow C'$ is applied on C to get C' where the changed value is labeled in red.

The matching criteria $key.m$ can have wildcards (*) to match ranges of values, and $key.pr$ is set to a finite integer number where a higher number means a higher priority: if a flow matches the matching criteria from multiple keys, the one with the highest priority is preferred and selected.

$$\begin{aligned} &key_1 = (pr = 1, m = \{src\_ip = 10.0.0.1\}),\\ &key_2 = (pr = 1, m = \{src\_ip = 10.0.0.2\}),\\ &key_3 = (pr = 2, m = \{dst\_ip = 10.0.0.*\}),\\ &key_4 = (pr = 2, m = \{src\_ip = 10.0.1.1, dst\_port = 22\}). \end{aligned} \tag{1}$$

An action ac of a flow rule represents the instruction that is applied to the flows matching it. An action can be forwarding to a specified next-hop, modifying a packet, or pipeline processing. The proposed models support the concept of multi-table pipelines in a switch: each flow table can simply be represented as an individual virtual sw. Formally, we define a data plane configuration as follows.

Definition 1 (Data Plane Configuration)**.** *A **data plane configuration** C of a network is defined as a map from the set of switches and flow rule keys to the set of actions $C : SW \times KEY \rightarrow AC^+$, where $AC^+ = AC \cup \{Null\}$.*

Therefore, a configuration C can be expressed with a 2-dimensional matrix over $SW \times KEY$ where each element $C_{sw,key}$ is an action $ac \in AC^+$ for (sw, key). $C_{sw,key} = Null$ represents the absence of a flow rule with key key at sw. Fig. 1 illustrates an example of C with four switches and two keys.

B. Data Plane Updates

A data plane update U consisting of a set of data plane operations $O(U)$ on multiple switches and flow rules can change one configuration to another. Data plane operations act on keys at a particular switch and fall into one of three categories: addition, modification, or deletion. We denote these operations as $\{add(sw, key, ac), mod(sw, key, ac), del(sw, key)\}$; *e.g.*, $add(sw, key, ac)$ means to add an action ac for key at sw. For example, consider the network topology depicted in Fig. 2. In the first update U_1, flows from source IP address 10.0.0.1 (key_1) are forwarded along the route $A \rightarrow B \rightarrow C \rightarrow D$; in the second update U_2, flows from source IP address 10.0.0.2 (key_2) are forwarded along $B \rightarrow C \rightarrow D$, and the forwarding path for key_1 is changed to $A \rightarrow E \rightarrow C \rightarrow D$ (*e.g.*, to balance the network load). Fig. 2 illustrates the corresponding data plane operations in both U_1 and U_2.

For further derivation, we introduce a more general expression o, parameterized by $SW \times KEY \times AC^+$: add, mod and del are all special cases of o; *e.g.*, $add(sw, key, ac)$ or $mod(sw, key, ac)$ can be expressed as $o(sw, key, ac)$, and $del(sw, key)$ as $o(sw, key, Null)$. An operation o transforms an arbitrary configuration C to another C' as follows:

Definition 2 (Data Plane Operation)**.** *A data plane operation $o(sw, key, ac)$ is defined as a **morphism** between two configurations,* i.e.*, $o : C \rightarrow C'$ or $C \xrightarrow{o} C'$, using:*

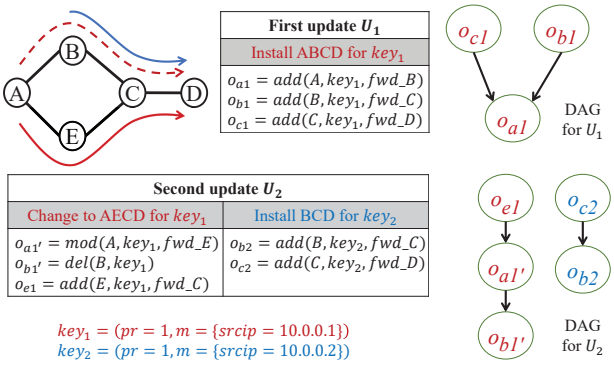


Fig. 2: An example of two consecutive updates involving common flows (match key_1). The order constraint ensuring blackhole freedom for each update is illustrated as a DAG of data plane operations.

$$C'_{sw',key'} = [o(sw, key, ac)(C)]_{sw',key'} \tag{2}$$

$$= \begin{cases} ac & \text{if } sw = sw', key = key', \\ C_{sw',key'} & \text{otherwise.} \end{cases} \tag{3}$$

That is, $o(sw, key, ac)$ maps C to C' by changing $C'_{sw,key} = ac$ but preserving other values of C. Fig. 1 shows an example of applying the operation $o_{a1'} = o(A, key_1, fwd\_E)$ on configuration C to obtain C', where the changed value is labeled in red.

Due to the distributed nature of the data plane, operations in an update can be applied in any order, resulting in different intermediate configuration states. However, some intermediate configurations during an update may violate consistency properties such as blackhole/loop-freedom; an *order constraint* is required to specify the feasible operation orders in an update.

Definition 3 (Order Constraint)**.** *The **order constraint** of update U is defined as $\Omega(U)$, which specifies the feasible operation orders in U to ensure consistency properties.*

A concrete instantiation of $\Omega(U)$ will be given in Section III-C. Fig. 2 presents an example of the order constraint to avoid blackholes where no matched rule exists during the updates; the order constraint is represented in the form of a directed acyclic graph (DAG); *e.g.*, the DAG for U_1 shows that o_{a1} must be applied after o_{b1} and o_{c1}. Note that generating the order constraints for various consistency properties is well studied in literature [9]–[12], and therefore not the focus of our work.

Issues with Multiple Updates. The model for individual updates is simple and well documented, but new issues arise when attempting to compose and execute multiple consecutive updates in a non-blocking manner. First, a naïve parallel execution of consecutive updates could lead to non-deterministic or incorrect outcomes. For example, in the scenario of Fig. 2, the execution of U_1, and U_2, could lead to non-deterministic configurations. This is because the operations o_{a1}, and $o_{a1'}$, apply to the same key_1 at switch A, but differ in actions: $o_{a1}.(sw, key) = o_{a1'}.(sw, key)$, but $o_{a1}.ac \neq o_{a1'}.ac$. Consequently, depending on the order in which they were applied, the two operations would lead to different configurations. Further, the order constraints in different updates may have dependencies that affect the non-blocking composition of consecutive updates and may be difficult to identify. For example, $o_{a1'}$ in U_2 must be applied after o_{c1} in U_1 to guarantee the absence of blackholes. Lastly, when a new update arrives, previous updates may be mid-execution. The execution status may even be unknown to the controller due to the fundamentally distributed nature of the SDN system. For example, if U_1 is partially executed when U_2 arrives, the exact execution status may be unknown to the controller.

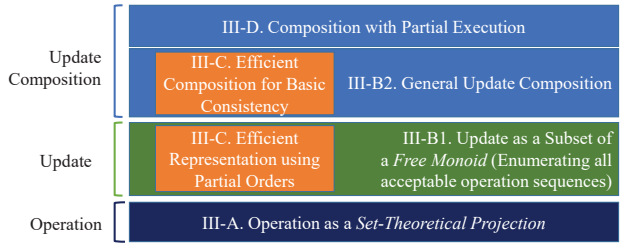


Fig. 3: Roadmap of update algebra.

III. Update Algebra Framework

In this section, we present our update algebra framework based on advanced abstract algebra. Specifically, data plane operations and updates are modeled by the notions of *set-theoretical projection* (Section III-A) and *free monoid* (Section III-B1), which provide the foundation and substantial algebraic properties for further composition. Based on these models, Section III-B2 proposes a general solution to compose consecutive updates. The solution is general as it can preserve any consistency property. Then, Section III-C introduces an efficient representation and composition using partial order to guarantee specific but commonly required properties. Lastly, Section III-D addresses the issue of composition with partially-executed updates to guarantee correctness under uncertainty. The roadmap of update algebra is illustrated in Fig. 3.

A. Operation as a Set-theoretical Projection

As the basic unit of an update, operation and its composition are first introduced in update algebra, providing the foundation and freedom to replace and rearrange the data plane operations within one update or across distinct updates.

Recall Definition 2 where a data plane operation is defined as a morphism from one configuration to another with one value changed. The operation morphism can be viewed as a *set-theoretical projection* as follows:

Definition 4 (Operation Projection)**.** *Considering the set of all possible configurations over a fixed $SW \times KEY$, such a set is the Cartesian product $(AC^+)^{SW \times KEY}$. Then the operation $o(sw, key, ac)$ can be considered a **projection** from $(AC^+)^{SW \times KEY}$ to the subset $\{C | C_{sw,key} = ac\}$.*

Definition 4 helps us to visualize an operation as a morphism of the configuration space, *i.e.*, the Cartesian product $(AC^+)^{SW \times KEY}$ as shown in Fig. 4. For example, assume $|SW \times KEY| = 2$, then the space is two-dimensional. Therefore, $o(sw, key, ac)$ projects any configuration points onto a line with (sw, key)-component $= ac$, and the operation on the other $(sw*, key*)$ is "orthogonal" to $o(sw, key, ac)$.

Definition 5 (Morphism Equality)**.** *Two morphisms π_1 and π_2 are **equivalent** iff for any configuration C, $\pi_1(C) = \pi_2(C)$.*

Since both the domain and codomain of an operation morphism are the configuration space, a set of data plane operations $\{o_1, o_2, ...\}$ can be "composed", *i.e.*, arranged in a sequence to form a new morphism. Formally, we define a binary operation $\circ$, called composition of morphisms, such that for any $o_1 : C_0 \to C_1$ and $o_2 : C_1 \to C_2$, we have $o_2 \circ o_1 : C_0 \to C_2$. By modeling an operation as a *set-theoretical projection* based on Definition 4, the operation composition holds the following properties:

Theorem 1 (Operation Composition Properties)**.** *The universal operation set $\mathcal{O} = \{o_1, o_2, ...\}$ and the binary operator $\circ$ have the following four properties under Definition 5.*

1) **Idempotence** (general): $\forall o_1$
$$o_1 \circ o_1 = o_1. \tag{4}$$
2) **Associativity** (general): $\forall o_1, o_2, o_3$
$$(o_3 \circ o_2) \circ o_1 = o_3 \circ (o_2 \circ o_1). \tag{5}$$
3) **Selectivity** (conditional): if $o_1.(sw, key) = o_2.(sw, key)$,
$$o_2 \circ o_1 = o_2. \tag{6}$$
4) **Commutativity** (conditional): if $o_1.(sw, key) \neq o_2.(sw, key)$,
$$o_2 \circ o_1 = o_1 \circ o_2. \tag{7}$$

Here $o_1.(sw, key) = o_2.(sw, key)$ means both the sw and the key of o_1 and o_2 are equal. Theorem 1 can be directly proven by Definition 2. Intuitively, if each morphism is considered as a projection based on Definition 4, we can visualize the four operation properties in the form of projections in a configuration space as in Fig. 4.

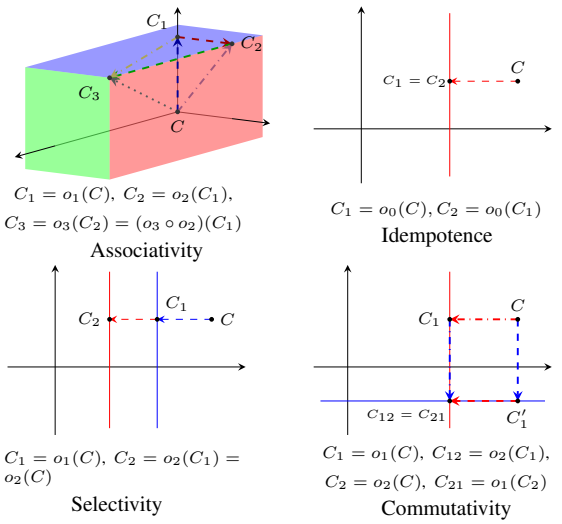


Fig. 4: Visualization of the four properties satisfied by projections.

The properties of real-world SDN implementation (*e.g.*, Flow table modification messages in OpenFlow) align exactly with these properties except for **Selectivity**. TABLE II illustrates a concrete example of **Selectivity**, where $o_1.(sw, key) = o_2.(sw, key)$. Note that if o_2 is a mod, all results after composition become an add. Because in our framework $add(sw, key, ac_2) = mod(sw, key, ac_2) = o(sw, key, ac_2)$, and an add can act as a potent mod (an add operation can override an action even though an action for the identical key already resides in the requested table sw), therefore **Selectivity** property can still hold. Note that $o.key$ is endowed with a determined priority, so operations on overlapping flow rules can be distinguished in composition.

TABLE II: Composition rules for $o_2 \circ o_1$.

$o_2 \setminus o_1$	$\boldsymbol{add(ac_1)}$	$\boldsymbol{mod(ac_1)}$	$\boldsymbol{del()}$
$\boldsymbol{add(ac_2)}$	$add(ac_2)$	$add(ac_2)$	$add(ac_2)$
$\boldsymbol{mod(ac_2)}$	$add(ac_2)$	$add(ac_2)$	$add(ac_2)$
$\boldsymbol{del()}$	$del()$	$del()$	$del()$

B. General Update Representation and Composition

1) Update as a Subset of a Free Monoid: Given the concept of operation composition, an update can be considered as the composition of data plane operations in different sequences, *e.g.*, $o_1 \circ o_2 \circ o_3$ and $o_3 \circ o_1 \circ o_2$. Therefore, we extend the definition of an update using a *free monoid* [15] to address possible operation sequences and capture the order constraint. The formal definitions of the *monoid* and the *free monoid* are given as follows:

Definition 6 (Monoid)**.** *A **monoid** (sometimes called a semigroup with identity element) is a 3-tuple $(S, e, *)$, where S is a set, $e \in S$ is an element, and $*$ is a binary operation $S \times S \to S$ such that for all $x, y, z \in S$, $x*(y*z) = (x*y)*z$ and $e * x = x * e = x$.*

Definition 7 (Free Monoid)**.** *A **free monoid** S^* on a generating set S is a monoid whose elements are all finite sequences (or strings) of zero or more elements from S, with string concatenation as the monoid operation $*$.*

Example: Letter and Words - A typical *free monoid* example is about letters and words. Start with an alphabet S of letters, $S = \{a, b, c, ..., z\}$. A word on the generating set S is a finite sequence of letters, *e.g.*, $infocom$, and $paris$. Thus, S^* is the set of all possible words, the identity element e is an empty word, and the operation $*$ is word-concatenation. In this *free monoid*, any words can be simply composed together to get a new word, *e.g.*, $no * on = noon$.

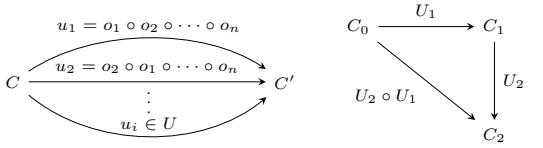
Consider a *free monoid* O^*, in which the generating set is a data plane operation set O and the identity element e is an empty update $\varnothing$ (*i.e.*, applying nothing on a configuration C). Then an update can be modeled as follows:

Definition 8 (Update Representation)**.** *An **update** is represented as a set $U = \{u_1, u_2, ...\}$ where u_i, called a **representative**, is a sequence of elements from $O(U)$, and satisfies the order constraint $\Omega(U)$ and the following conditions:*

- *Constitution:* $\forall o \in O(U),\ o \in u_i$;
- *Distinction:* $\forall o_i, o_j \in u_i,\ o_i.(sw, key) \neq o_j.(sw, key)$.

Remark. U is a subset of the *free monoid* $O(U)^*$ on the generating set $O(U)$. Constitution condition guarantees that all representatives have the constituent operations. Distinction condition avoids configuring a flow rule twice in an update. Each representative u_i representing an order to compose $O(U)$ can transform a C to another as follows:

Definition 9 (Update Representative Morphism)**.** *An update representative $u_i = o_1 o_2 ... o_k$ can be considered as a mor-*



(a) Equivalence of representatives. (b) Update composition.

Fig. 5: Diagrams of the update algebra. (a) Applying any representative of an update achieves the same result. (b) If $U_1 : C_0 \to C_1$ and $U_2 : C_1 \to C_2$, then $U_2 \circ U_1 : C_0 \to C_2$.

phism between two configurations, i.e., $u_i : C \to C'$, *where* $C' = u_i(C) = (o_1 \circ o_2 \circ ... \circ o_k)(C)$.

Theorem 2 (Equivalence of Representatives)**.** *Given an update U, for any configuration C, we have $u_i(C) = u_j(C)$, $\forall u_i, u_j \in U$.*

Theorem 2 can be simply proven with **Commutativity** in Theorem 1 and the conditions in Definition 8. Fig. 5(a) illustrates the diagram of Theorem 2. According to the *monoid* presentation theory [15], an update U is an equivalence class of representatives in a *free monoid*, thus can be further defined as follows:

Definition 10 (Update Morphism)**.** *An update U can be considered a morphism between two configurations,* i.e., *$U : C \to C'$, where $C' = U(C) = u_i(C)$, $\forall u_i \in U$.*

Definitions 8 and 10 illustrate an update in the perspectives of representation and morphism respectively. Updates as morphisms are equipped with a composition operation. The diagram of the update composition is illustrated in Fig. 5(b), and such composition presents the following properties:

Theorem 3 (Update Composition Properties)**.** *The universal update set $\mathcal{U} = \{U_1, U_2, ...\}$ and the binary operator $\circ$ have **Associativity** and **Idempotence** properties under Definition 5;* i.e., *$\forall U_1, U_2, U_3$, we have $(U_3 \circ U_2) \circ U_1 = U_3 \circ (U_2 \circ U_1)$ and $U_1 \circ U_1 = U_1$.*

Associativity in Theorem 3 is inherited from a *monoid* as in Definition 6, and **Idempotence** can be proven by **Idempotence** and **Commutativity** of operation composition in Theorem 1.

2) General Update Composition: Represented by a subset of $O(^*U)$ enumerating all acceptable operation sequences, an update can be determined by its constituent operations $O(U)$ and order constraint $\Omega(U)$. Therefore, the composition $U_k \circ ... \circ U_1$ is determined by $O(U_k \circ ... \circ U_1)$ and $\Omega(U_k \circ ... \circ U_1)$.

Computation of $O(U_k \circ ... \circ U_1)$. The goal of update composition is to obtain the equivalent operation sequences with a lower operation number. Based on Associativity in Theorem 1, multiple updates can be concurrently composed in a flexible way. Given a set of updates $\{U_1, U_2, ..., U_k\}$, we propose a general solution to compute $O(U_k \circ ... \circ U_1)$ with the following steps:

1) Choose an arbitrary representative u_i from each update U_i, $i \in [1, k]$,
2) Concatenate them with $\circ$, *i.e.*, $u_k \circ ... \circ u_1$,
3) Simplify the concatenated sequence of operations using the properties in Theorem 1 until Distinction condition in Definition 8 is satisfied,

Example. Consider the composition of U_1 and U_2 in Fig. 2, *i.e.*, $U_2 \circ U_1$. Randomly choose their representatives as u_1 and u_2 respectively, and then the composition $u_2 \circ u_1$ can be simplified as follows:

$$
\begin{aligned}
u_2 \circ u_1 &= (o_{a1'} o_{b1'} o_{e1} o_{b2} o_{c2}) \circ (o_{a1} o_{b1} o_{c1}) \\
&= (o_{a1'} \circ o_{b1'} \circ o_{e1} \circ o_{b2} \circ o_{c2}) \circ (o_{a1} \circ o_{b1} \circ o_{c1}) \\
&= o_{a1'} \circ o_{a1} \circ o_{b1'} \circ o_{b1} \circ o_{e1} \circ o_{b2} \circ o_{c2} \circ o_{c1} \\
&\quad \text{by Associativity and Commutativity in Theorem 1} \\
&= o_{a1'} \circ o_{b1'} \circ o_{e1} \circ o_{b2} \circ o_{c2} \circ o_{c1} \\
&\quad \text{by Selectivity Theorem 1} \\
&= o_{a1''} \circ o_{b1'} \circ o_{e1} \circ o_{b2} \circ o_{c2} \circ o_{c1} \\
&\quad o_{a1'} \text{ is changed to } o_{a1''} \text{ according to TABLE II}
\end{aligned}
$$

Computation of $\Omega(U_k \circ ... \circ U_1)$. To obtain all acceptable update sequences (representatives) from $O(U_k \circ ... \circ U_1)$, we also need $\Omega(U_k \circ ... \circ U_1)$, so that the consistency properties can be preserved. Given a set of operations, existing work on consistent updates provide a large number of efficient algorithms to generate an order constraint for various consistency properties, including loop-freedom [9], and waypoint routing [11]. A general solution is to take $O(U_k \circ ... \circ U_1)$ as an input and run one of the algorithms to get $\Omega(U_k \circ ... \circ U_1)$. For properties whose order constraint consists of partial orders, Section III-C below presents a solution to compute $\Omega(U_k \circ ... \circ U_1)$ more efficiently.

C. Efficient Representation and Composition for Basic Consistency

The update representation using a subset of a *free monoid* is complete, but the sequence set U can get large, especially when composing updates with long sequences of operations; for example, if U_1 has n_1 sequences and U_2 has n_2, a naïve concatenation could result in $n_1 \times n_2$ possible sequences. This section introduces an efficient and compact way to specify and derive the order constraints consisting of partial orders. This format of order constraint guarantees the basic consistency property defined in Definition 11 below.

Definition 11 (Basic Consistency)**.** *The basic consistency property is defined as blackhole- and loop-freedom.*

Basic consistency is commonly used in networks to avoid packet drops and traffic loops [9], [16]. To ensure it, the update execution can be represented by a DAG as in Fig. 6. Consider each edge in the DAG to be a partial order pair, the $\Omega(U)$ for the basic consistency can be represented by a strict partial order set, whose elements are of the form $o_1 \prec o_2$, denoting that o_1 ought to be applied before o_2. The generation of the DAG and partial order set for the composition can be found in [17]. For example in Fig. 6, $\Omega(U_1) = \{o_{c1} \prec o_{a1}, o_{b1} \prec o_{a1}\}$ and $\Omega(U_2) = \{o_{e1} \prec o_{a1'}, o_{a1'} \prec o_{b1'}, o_{c2} \prec o_{b2}\}$. The strict partial order $\prec$ satisfies the following properties:

1) $o \not\prec o$,
2) if $o_1 \prec o_2$ and $o_2 \prec o_3$, then $o_1 \prec o_3$,
3) if $o_1 \prec o_2$, then $o_2 \not\prec o_1$.

Based on the partial ordering, we propose an efficient solution to achieve update composition, *e.g.*, $U_k \circ ... \circ U_1$. In our solution, the constituent operations $O(U_k \circ ... \circ U_1)$ can be obtained by the same solution as in Section III-B2, while $\Omega(U_k \circ ... \circ U_1)$ is computed as follows:

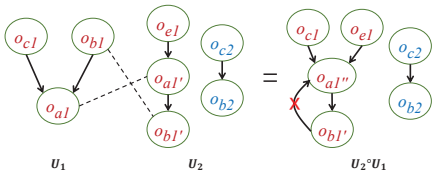


Fig. 6: An example of the efficient composition of U_1 and U_2 in Fig. 2 to preserve the basic consistency, $o_{a1''} = o_{a1'} \circ o_{a1}$ is computed according to TABLE II.

- Step 1: Combine the order constraints together, *i.e.*, $\Omega(U_k \circ ... \circ U_1) = \Omega(U_1) \cup \Omega(U_2) \cup ... \cup \Omega(U_k)$;
- Step 2: Replace operations by composed counterparts in $O(U_k \circ ... \circ U_1)$;
- Step 3: Search cycles in $\Omega(U_k \circ ... \circ U_1)$ based on the properties of $\prec$;
- Step 4.1: Break each cycle by removing the order element inherited from earlier updates, *i.e.*, if $\exists o_1' \prec o_2' \in \Omega(U_i), o_2'' \prec o_1'' \in \Omega(U_j)$, s.t. $o_1.(sw, key) = o_1'.(sw, key) = o_1''.(sw, key), o_2.(sw, key) = o_2'.(sw, key) = o_2''.(sw, key)$ and $i < j$, then remove element $o_1 \prec o_2$;
- Step 4.2: After removing $o_1 \prec o_2$, inherit the implicit dependencies between o_1''s parents and o_2''s children from the previous update U_i, *i.e.*, $\forall m, n$, if $\exists o_m \prec o_1' \in \Omega(U_i)$ and $o_2' \prec o_n \in \Omega(U_i)$, add elements $o_m \prec o_n$ into $\Omega(U_k \circ ... \circ U_1)$;
- Step 5: Simplify $\Omega(U_k \circ ... \circ U_1)$ to a minimal partial order set by removing redundant elements based on the transitivity of $\prec$.

Theorem 4. *If the sequential execution from $\Omega(U_1)$, $\Omega(U_2)$,..., to $\Omega(U_k)$ guarantees the basic consistency, the non-blacking execution of $\Omega(U_k \circ ... \circ U_1)$ in update algebra can guarantee the basic consistency.*

The detailed proof of Theorem 4 is omitted due to space limitation. The intuition here is that the first two steps allow $\Omega(U_k \circ ... \circ U_1)$ to inherit the orders from $\{\Omega(U_1), \Omega(U_2), ..., \Omega(U_k)\}$. In Step 3, any cycle indicates the presence of order conflicts between U_i and U_j for the same flow rules. Since based on **Selectivity** in Theorem 1, the conflicting operations are overwritten by the last one, applying the same rationale into the order constraint, only the order element in U_j, $(i < j)$, is preserved as depicted in Step 4.1. Because of the transitivity property, many of the order dependencies are hidden and implicit in the order constraint. But to break cycles, we remove some elements, which may lead to the loss of these implicit dependencies. For the example in Fig. 6, if we remove $o_{e1} \prec o_{a1'}$ from $\Omega(U_2)$, the implicit dependency $o_{e1} \prec o_{b1'}$ will be lost. Therefore in Step 4.2, to avoid such loss, for each element we remove from $\Omega(U_k \circ ... \circ U_1)$, we inherit its implicit dependencies from the previous corresponding update. The goal of Step 5 is to remove redundant order elements in the constraint, *e.g.*, if Ω contains $o_1 \prec o_2$, $o_2 \prec o_3$ and $o_1 \prec o_3$, the last component $o_1 \prec o_3$ is redundant and should be removed.

Example. Fig. 6 shows the efficient composition of U_1 and U_2 in the example of Fig. 2. We use the format of DAGs for simple illustration. After Steps 1 and 2, there is a cycle between $o_{a1''}$ and $o_{b1'}$, where $o_{a1''} = o_{a1'} \circ o_{a1}$ is computed according to TABLE II. Since $o_{b1'} \prec o_{a1''}$ is inherited from $\Omega(U_1)$ but conflicts with $\Omega(U_2)$, it will be removed as depicted in Step 4.1 of our solution. This example does not involve Steps 4.2 and 5, but more interesting and complicated examples can be found in our technical report [17].

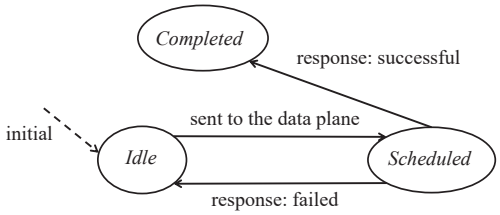


Fig. 7: Finite machine of an operation.

D. Composition with Partial Execution

The update algebra in the previous sections assumes that all operations of updates to be composed are not executed. However in practice, new updates can arrive while previous ones are partially executed. The problem is that such a partial execution results in uncertainty of configuration states. To resolve this problem, we 1) introduce an uncertainty model to reflect partial executions, and 2) provide a solution for continuous and non-blocking composition for updates with partial executions.

1) Uncertainty Model: An SDN controller sends operations of an update to a data plane (in a proper order) to execute the update. Once switches receive the operations, they reply to the controller with the progress of execution. Therefore, from the perspective of the SDN controller, each operation has one of the following states: *Idle*, *Scheduled* or *Completed*. Fig. 7 shows the finite machine of an operation state. Initialized with state *Idle*, once an operation is sent to the data plane for installation, its state becomes *Scheduled*. After installation, switches return a response message to notify whether the operation is applied successfully. A successful response turns the operation state into *Completed* state whereas a failed response turns it back to *Idle*.

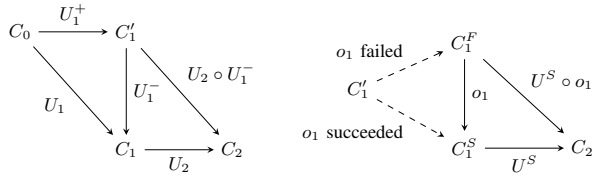
The configurations according to the states of an operation $o : C \rightarrow C'$ are as follows:

- **Invariant 1:** If o is at *Idle*, it is not applied at the data plane; *i.e.*, the configuration is C.
- **Invariant 2:** If o is at *Completed*, it is applied at the data plane; *i.e.*, the configuration is C'.
- **Uncertainty:** If o is at *Scheduled*, it can be applied or not; *i.e.*, the configuration is C or C'.

Partial Execution. Consider an update composition of U_1 and U_2 in which U_1 is partially-executed when U_2 arrives at the controller. Let U_1^+ denote the part of U_1 that has been applied (*Completed*), and U_1^- the remainder, i.e., $U_1 = U_1^- \circ U_1^+$. Based on **Associativity** in Theorem 3, we have:

$$U_2 \circ U_1 = U_2 \circ (U_1^- \circ U_1^+) = (U_2 \circ U_1^-) \circ U_1^+ \quad (8)$$

Fig. 8(a) presents the transitions of configuration states according to executed updates. Note that we compose U_1 and U_2 at the configuration state C_1' with a partial execution U_1^+, which means $U_2 \circ U_1^-$ is our target composition. The problem is that C_1' and U_1^- may be unknown to the controller due to the uncertainty state of operations at *Scheduled* state. A solution is



(a) Uncertainty model. (b) Illustration of Problem 1.

Fig. 8: (a) Uncertainty model where C_1' and U_1^- may be unknown. (b) Illustration of Problem 1 where o_1 is at the *Scheduled* state, so its execution can be either failed or successful.

to prohibit *Scheduled* states during composition; *i.e.*, once U_2 arrives, the controller stops scheduling new operations in U_1 and collects responses for all *Scheduled* ones until their states become steady one such as *Idle* and *Completed*. However, this solution inefficiently blocks updates due to the interruption for collecting the responses.

2) Solution: The basic idea behind our solution is to treat *Scheduled* operations as "not applied" at a data plane during composition. Consider an update composition $U_2 \circ U_1$ in Fig. 8(a). Assume that one operation o_1 in the update U_1 is in *Scheduled* state. Suppose that C_1^F and C_1^S are the configurations after failed and successful responses for o_1, respectively. Based on the two **Invariants**, the current configuration C_1' must be either C_1^F or C_1^S. Then, we aim to solve:

Problem 1. *Given* $C_1' \in \{C_1^F, C_1^S\}$, *find* $U : C_1' \rightarrow C_2$.

Let $U^S : C_1^S \rightarrow C_2$ denote the composed update to achieve U_2. As depicted in Fig. 8(b), we have

$$C_1^S = o_1(C_1^F), \tag{9}$$

$$C_2 = U^S \circ o_1(C_1^F). \tag{10}$$

Recall that our solution treats *Scheduled* operations as "not applied", *i.e.*, $C_1' = C_1^F$. Based on the assumption $C_1' = C_1^F$, the solution is $U = U^S \circ o_1$ from Equation (10). We show that the solution yields $U(C_1') = C_2$ even in the case of $C_1' = C_1^S$ as follows:

$$U^S \circ o_1(C_1') = U^S \circ o_1(C_1^S) \tag{11}$$

$$= U^S \circ o_1(o_1(C_1^F)) \tag{12}$$

$$= (U^S \circ o_1) \circ o_1(C_1^F) \tag{13}$$

$$= U^S \circ (o_1 \circ o_1)(C_1^F) \tag{14}$$

$$= U^S \circ o_1(C_1^F) \tag{15}$$

$$= C_2, \tag{16}$$

where Equation (12), (14), (15) and (16) are deduced from (9), **Associativity**, **Idempotent** in Theorem 1 and Equation (10), respectively.

Therefore, regardless of the uncertainty due to *Scheduled* operations, the update algebra is able to compute an update composition and achieve correctness based on our solution.

IV. Evaluation

This section evaluates the benefits of update algebra through asymptotic analysis (Section IV-A), extensive benchmarking using a real controller (Section IV-B), and integration with a real application (Section IV-C).

A. Asymptotic Analysis

We conduct an asymptotic analysis to compare the correctness, overhead, and completion time of the composition of consecutive updates using sequential, parallel, and update algebra based executions. We denote $p \in [0, 1]$ the probability that two consecutive updates are related, *i.e.*, involve common flows. In other words, $1 - p$ denotes the probability that two consecutive updates are fully independent. Each update is represented as a sequence of operations. For simplicity, if two consecutive updates U_1 and U_2 are related (with probability p), the two sequences are modeled as overlapping, and the common segment is randomly selected using a uniform distribution. The total length of the update after composition is provided by Equation (17), with N_1 and N_2 representing the lengths of U_1 and U_2. The details of the proof are omitted due to space limitation.

$$f(N_1, N_2) = \frac{N_1^2 + N_1 N_2 + N_2^2}{N_1 + N_2} - 1 \tag{17}$$

TABLE III summarizes the results. First, as explained in Section II, parallel execution may violate correctness as it does not respect update dependencies. Second, TABLE III shows that the completion time of a sequential execution increases linearly with the number of updates. In contrast, in independent-update dominant settings, the completion time with update algebra stays asymptotically constant, similar to that of a parallel execution, while guaranteeing correctness.

TABLE III: Asymptotic analysis of the composition of two consecutive updates U_1 and U_2.

	Correctness	Operation Number (N)	Update Completion Time (T)
Sequential	$\surd$	$N_1 + N_2$	$T_1 + T_2$
Parallel	$\times$	$N_1 + N_2$	$max(T_1, T_2)$
Update Algebra	$\surd$	$\approx f(N_1, N_2)p + (N_1 + N_2)(1-p)$	$\approx f(T_1, T_2)p + max(T_1, T_2)(1-p)$

B. Benchmarking

Methodology: We deploy update algebra in a SDN network running a Ryu 4.24 controller, and twenty Open vSwitch [18] 2.5.4, connected in a FatTree topology, which is common in data centers.

We generate updates as follows: each update involves a random number of flows ranging from 17 to 20. Two successive updates include common flows with a probability of p. We vary p, and for each case, we synthesize 300 network update events with different Poisson arrival rates λ.

The controller schedules the arriving updates according to the order constraints (*i.e.*, DAG) derived using the algorithm [9] proposed by Forster *et al.* to ensure a lack of blackholes and loops during the updates. We compare two composition approaches: in the first, the controller keeps track of the state of each operation, and continuously merges new arriving updates with ongoing ones through update algebra; in the second approach, the controller executes the arriving updates in a sequential (blocking) manner, *i.e.*, according to a First-In-First-Out (FIFO) policy.

To compare the performance, we report two metrics: *average operation number*, and *average update completion time*.

The former is the average number of operations applied in the data plane for the 300 updates. The latter is the average duration of an update which begins when it is considered by the controller, and ends when the last operation is completed or merged with new updates.

Results:

• *Control Overhead:* Fig. 9(a) shows the average number of operations, which reflects the control message overhead in the system. The sequential execution yields a constant number of operations for different values of λ as all updates are executed individually and sequentially, independent of their arrival rate. In contrast, the overhead of update algebra based execution varies with λ: the number of operations is similar to that of the sequential execution when λ is low, as each update completes before the next one arrives. However, as λ increases, fewer updates are completed before new ones arrive. Therefore, consecutive updates can be merged, reducing the number of operations. Comparing the results of different p, we observe that when successive updates are more related (*i.e.*, larger p), more operations can be merged through update algebra, resulting in a reduction in numbers of operations. As such, with $\lambda = 2$/s (the network being updated twice a second on average), the non-blocking execution in update algebra reduces the control message overhead by up to 30%.

• *Completion time:* Fig. 9(b) shows the average update completion time. It includes both the results from the experimental evaluation, and the theoretical upper and lower bounds.

For the theoretical bounds, we model the sequential execution as an M/M/1 system where the arrival rate is λ and the service rate μ is 1/(average update execution time); note that the execution time of individual updates is exponentially distributed in our experiments. For the parallel execution, as updates can be executed concurrently, the execution bottleneck comes from the operation queues at the switches. Considering that updates arrive at switches uniformly at random, the parallel execution can be approximated by an M/M/c system, where c is the number of switches in the network. Therefore, the average response time in the M/M/1 system reflects the upper bound of the update completion time, and that the M/M/c system corresponds to the lower bound.

Fig. 9(b) depicts the theoretical bounds, and the measured update completion times of both the sequential execution and update algebra. Both the sequential and update algebra based executions yield similar constant times at low arrival rates as all updates are completed without blocking. However, as λ becomes larger, our approach completes updates faster than the sequential execution. This is because the sequential execution suffers from long waiting times due to the blocking in the queue, while update algebra enables updates to be executed concurrently, and reduces the number of operations, *e.g.*, by merging operations with same keys. When $p = 0.2$ and $\lambda = 1.2$/s, update algebra improves network update speed by up to 8x compared to the sequential execution, and when $p = 0.8$ and $\lambda = 1.6$/s, the gain can reach 16x.

The close-up figure in Fig. 9(b) (right-hand) clearly shows the impact of different update patterns on the update performance. As p increases, update algebra achieves higher gain, and the update completion time with update algebra gets closer to that of the theoretical lower bound. The results demonstrate that by maximizing the execution parallelism while preserving consistency properties, update algebra can handle frequent network changes in a non-blocking, but also efficient way.

C. Performance in a Real Application

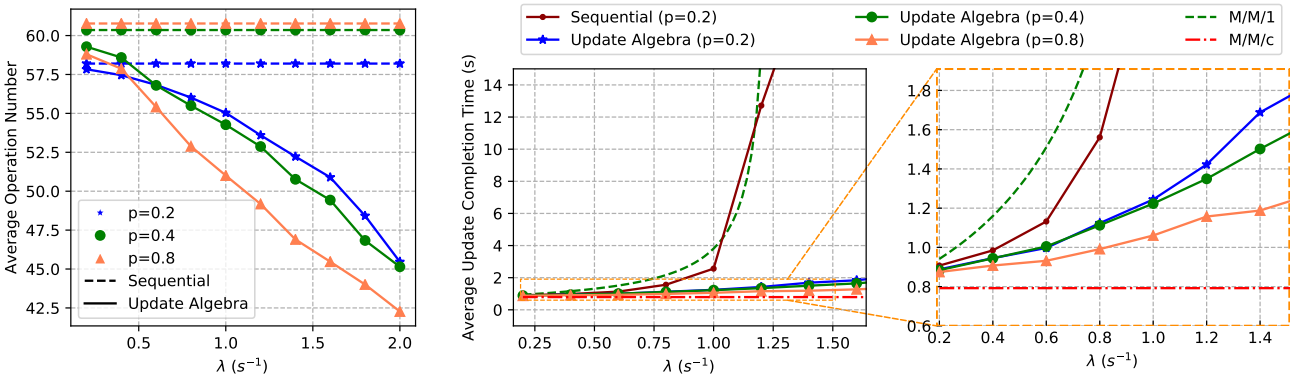
This section evaluates the performance benefits of update algebra integrated with a real application. We deploy Hedera [2] in the SDN controller. Hedera continuously collects network statistics, and updates flow routes to maximize the aggregate network utilization. We use an unbalanced traffic pattern with a large workload, and measure the average link bandwidth utilization with different update frequencies. The update frequency is an adjustable parameter (with a default value 0.2/s) of Hedera specifying how frequently the updates are generated.

Fig. 10 shows that the update algebra based execution outperforms the sequential execution. As the update frequency increases, more updates are generated. The sequential execution cannot complete the updates before the next ones arrive. As a result, the arriving updates accumulate and the controller fails to update the network as directed by Hedera, causing network performance to degrade quickly. This is the reason frequent updates are prohibited in current systems, and the default frequency is set to 0.2/s (an update every five seconds). In contrast, by merging consecutive updates in a non-blocking manner, update algebra allows updates to be quickly enforced, and the network performance keeps increasing with the update frequency. With an update frequency of 1.2/s, update algebra increases the network utilization by 13% compared to the default value 0.2/s, and outperforms the sequential execution by 30%. These results demonstrate the benefits of update algebra with a real application.

V. Related Work

Consistent Updates: A concerted research effort has recently been made to tackle the problem of network updates in SDN for different aims. Xitao *et al.* [8] minimize the number and latency of rule updates for TCAM-based switches by eliminating redundant and unnecessary entry moves. Reitblatt *et al.* [19] introduce the notion of consistent network updates, and propose a two-phase update approach. Solutions supporting a broad range of consistent properties are proposed, including loop freedom [9], congestion-freedom [10], waypoint routing [11] and customizable properties [12]. However, existing work is limited to a single network update at a time, and does not handle consecutive network updates. Peter *et al.* [13] mention the inter-update scheduling problem in their future work, but only provide a strawman solution as an enhancement to the sequential approach. In contrast, our work allows controllers to efficiently merge multiple network updates to handle continuous and non-blocking network changes while preserving desired properties. To the best of the authors' knowledge, our work is the first to handle multiple updates as a group.

Policy Composition: Researchers have also investigated composing policies. Several recent SDN policy languages and controller hypervisors (*e.g.*, NetKAT [20], Pyretic [21]) support taking multiple high-level policies and generating flow tables that fulfill the semantics of the sequential and parallel composition. However, network update operations are different from flow rules, and present unique challenges as well as distinct requirements. For example, as discussed in



(a) Average operation number.

(b) Average update completion time. The right figure is a close-up view of a specific area from the left figure.

Fig. 9: Benchmarking results.

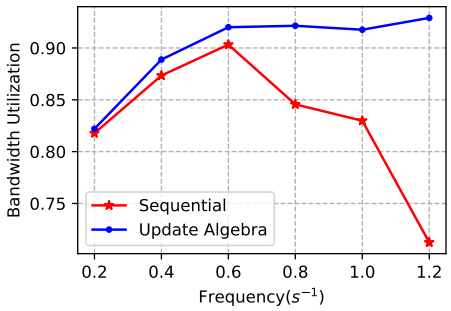


Fig. 10: Bandwidth utilization of Hedera with sequential execution and update algebra.

Section III-D, network updates may be partially executed, and controllers may not have a complete and precise up-to-date view of the update progress. In addition, network updates have distinct consistency requirements that differ from those of policy composition. Consequently, existing work on policy composition cannot be applied to composing network updates. Instead, we developed a theoretical framework that captured the unique characteristics of network updates, and allowed us to reason about their properties and composition.

Acknowledgment

We would like to thank Sanat Khurana who found a counter example of our earlier algorithm and helped with the design of a new solution. This research was supported in part by NSFC #61701347, NSFC #61702373 and NSFC #61672385; NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

References

[1] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," *INFOCOM*, 2018.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks." in *Proc. of NSDI*, 2010.

[3] Q. Xiang, H. Yu, J. Aspnes, F. Le, L. Kong, and Y. R. Yang, "Optimizing in the dark: Learning an optimal solution through a simple request interface," in *AAAI*, 2019.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. of SIGCOMM*, 2014.

[5] G. Li, Y. Qian, L. Liu, and Y. R. Yang, "JMS: Joint Bandwidth Allocation and Flow Assignment for Transfers with Multiple Sources," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 123–130.

[6] Q. Xiang, J. J. Zhang, X. T. Wang, Y. J. Liu, C. Guok, F. Le, J. MacAuley, H. Newman, and Y. R. Yang, "Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences," in *Proceedings of SC*, 2018.

[7] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *Proc. of HotNets*, San Diego, CA, 2004.

[8] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "RuleTris: Minimizing rule update latency for TCAM-based SDN switches," in *Proc. of ICDCS*, 2016.

[9] K.-T. Förster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *Proc. of IFIP Networking*, 2016.

[10] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *INFOCOM*, 2016.

[11] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies," in *Proc. of HotNets*, 2014.

[12] W. Zhou, D. K. Jin, J. Croft, M. Caesar, and P. B. Godfrey, "Enforcing customizable consistency properties in Software-Defined Networks." in *Proc. of NSDI*, 2015.

[13] P. Pereíni, M. Kuzniar, M. Canini, and D. Kostić, "ESPRES: transparent SDN update scheduling," in *Proc. of HotSDN*, 2014.

[14] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[15] D. S. Dummit and R. M. Foote, *Abstract algebra*. Wiley Hoboken, 2004.

[16] G. Li, Y. Qian, C. Zhao, Y. R. Yang, and T. Yang, "DDP: Distributed Network Updates in SDN," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[17] G. Li, Y. R. Yang, F. Le, Y. sup Lim, J. Wang, and S. Khurana, "Update Algebra: Toward Continuous, Non-Blocking Composition of Network Updates in SDN (extended version)," https://cpsc.yale.edu/sites/default/files/files/tr1548.pdf, Yale Technical Report TR1548, Tech. Rep.

[18] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open vSwitch." in *Proc. of NSDI*, 2015.

[19] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *Proc. of SIGCOMM*, 2012.

[20] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," in *ACM SIGPLAN Notices*, 2014.

[21] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker *et al.*, "Composing Software Defined Networks." in *Proc. of NSDI*, 2013.

Public Review for

On Max-min Fair Allocation for Multi-source Transmission

G. Li, Y. Qian, Y. Richard Yang

Max-min fairness is one of the classical objectives for traffic engineering or congestion control schemes. It is well understood and widely used by the community. When we received this submission, we were a bit surprised to see another paper on this rather old topic. In the classical max-min fairness formulation, each source produces its own stream of data and max-min fairness ensures a fair distribution of the ressources among competing flows. The new model proposed in this paper adapts the formulation to the more general case of data streams which can originate from multiple sources. This corresponds to the utilization of replicas such as CDNs where multiple hosts can source the same data stream. The authors formulate this problem as an optimization problem and propose a MultiSource Water-Filling algorithm (MS-WF) to compute the bandwidth allocations and the flow assignments. The proposed algorithm is described and evaluated by simulations. Unfortunately, the authors could not release this simulator and the associated results as software artifacts.

The reviewers discussed the merits of the paper. Some were convinced by the nice formulation of the problem and the initial simulation results. A drawback of the approach is that MS-WF assumes a centralized controller that has a precise information about the traffic matrix, which might be unrealistic in some deployments. The simulation results should be analyzed with this caveat in mind. Other reviewers were more concerned about the applicability and the deployability of the proposed solution. These are clearly left for further work. They also questioned whether max-min was the right goal. The paper was discussed during the rebuttal phase and the authors have revised the paper accordingly and added the proofs in a technical report.

Public review written by
Olivier Bonaventure
UCLouvain

On Max-min Fair Allocation for Multi-source Transmission

Geng Li[+*], Yichen Qian[*], Y.Richard Yang[+]
[+] Yale University, [*] Tongji University
geng.li@yale.edu,92yichenqian@tongji.edu.cn,yry@cs.yale.edu

ABSTRACT

Max-min fair is widely used in network traffic engineering to allocate available resources among different traffic transfers. Recently, as data replication technique developed, increasing systems enforce multi-source transmission to maximize network utilization. However, existing TE approaches fail to deal with multi-source transfers because the optimization becomes a joint problem of bandwidth allocation as well as flow assignment among different sources. In this paper, we present a novel allocation approach for multi-source transfers to achieve global max-min fairness. The joint bandwidth allocation and flow assignment optimization problem poses a major challenge due to nonlinearity and multiple objectives. We cope with this by deriving a novel transformation with simple equivalent canonical linear programming to achieve global optimality efficiently. We conduct data-driven simulations, showing that our approach is more max-min fair than other single-source and multi-source allocation approaches, meanwhile it outperforms others with substantial gains in terms of network throughput and transfer completion time.

CCS CONCEPTS

• **Networks → Packet scheduling**;

KEYWORDS

Max-min Fairness, Traffic Engineering, Multi-source Transmission

1 INTRODUCTION

Max-min fair is a simple, classical and well-recognized sharing principle to define fairness in the field of data networks [14]. In particular, it deals with the flow control problem in network traffic engineering (TE), where available resources (such as link bandwidth) are allocated among different traffic transfers [13]. Aiming at allocating rates to available links as evenly as possible, the max-min fair allocation is such that the result for the transfer with smallest sharing has been maximized over all feasible resource allocation solutions. As the development of software-defined networking (SDN), max-min fair has become the most widely used principle in modern datacenter networks or WANs, such as B4 [9], BwE [11], SWAN [8], OWAN [10]. Leveraging a centralized controller, max-min fairness results in more stable service quality than other principles, *e.g.*, proportional fairness.

Data replication, a technique that amends both data availability and access efficiency, is emerging increasingly in recent datacenter networks and distributed filesystems [7, 15, 17]. To maximize network utilization and improve transmission performance, it is increasingly allowed to convey data in parallel from multiple sources for a transfer with multiple data replicas, *a.k.a* a multi-source transfer [2, 16]. However, a multi-source transfer will result in multiple flows, so multiple potential bottlenecks might be considered simultaneously to achieve transfer-level fairness. Existing TE approaches are no longer applicable because the allocation becomes a joint problem to optimally allocate each flow's bandwidth as well as to assign flows among the sources [8, 9, 11].

In this paper, we present a novel max-min fair allocation approach that jointly optimizes bandwidth allocation and flow assignment. The major challenge stems from the direct formulation which is nonlinear and multi-objective. We cope with this by deriving a novel transformation with simple equivalent canonical linear programming (LP) to achieve global optimality. We perform extensive simulations, which show that our approach leads to better performance on network throughput with a gain of up to 52% and reduces transfer completion time by up to 44%, compared to other single-source and multi-source allocation approaches.

2 NETWORK MODEL AND PROBLEM FORMULATION

A network is comprised of a set of nodes and a set of links $\mathcal{L}$. Let C_j denote the capacity of link L_j ($j \in [1, M]$). Consider N data transfers, coming from one or multiple sources, so each transfer i ($i \in [1, N]$) is assigned with a set of flow paths $\{P_{i1}, ..., P_{ik}\}$, and each such path is identified with the set of links that it uses, *i.e.*, $P_{ik} \subseteq \mathcal{L}$. Now we define the data rate of transfer i as r_i, which is the sum bandwidth of the constituent flows from all sources. We use a variable set $\mathcal{X}_i = \{x_{i1}, ..., x_{ik}\}$ to express the flow assignment proportions from different sources for transfer i, so $\sum_{k=1}^{K_i} x_{ik} = 1$, where K_i is the total source number of transfer i.

Objective and constraints. Our goal is to solve a joint bandwidth allocation and flow assignment problem, *i.e.,* finding the transmission rate r_i and the assignment proportions $\mathcal{X}_i$ of each transfer. When computing allocated bandwidth, the objective is to maximize network utilization while in a max-min fair manner. A vector of transfer rate allocations $\{r_i\}$ is said to be max-min fair if and only if, for any other feasible allocation $\{r'_i\}$, the following has to be true: $\forall r'_p > r_p$ for the data transfer p, there exists another transfer q such that $p, q \in [1, N]$, $r'_q < r_q$, $r_q \leq r_p$. The constraints of this problem are given as below. Constraint (1) is called the capacity constraint, which assures that for any link L_j, its load does not exceed its capacity C_j. Constraints (2) and (3) promise the sum fractions of a transfer from all available sources equal to 1.

$$s.t. \quad \sum_{L_j \subseteq P_{ik}} r_i \cdot x_{ik} \leq C_j \qquad \forall j, \tag{1}$$

$$\sum_{k=1}^{K_i} x_{ik} = 1 \qquad \forall i, \tag{2}$$

$$0 \leq x_{ik} \leq 1 \qquad \forall i, k. \tag{3}$$

Figure 1 shows an example where the six links $\{L_1, ..., L_6\}$ have capacities $\{8, 5, 4, 5, 7, 6\}$ in *Gbps* respectively. Consider three data transfers, among which transfer 1 and 2 have a single data source

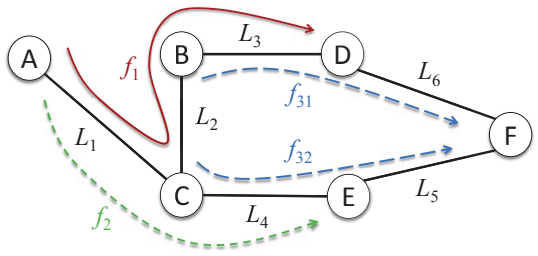


Figure 1: An example with 3 transfer requests. Transfer 3 comes from two feasible sources B and C, corresponding to two parallel flows f_{31} and f_{32}.

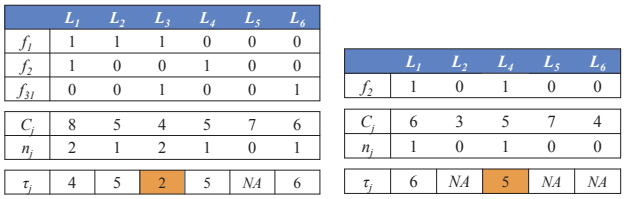


Figure 2: An example of the FL matrix for single-source transmission, where transfer 3 only uses one source. C_j is the bandwidth capacity, $n_j = \sum_i fl_{ij}$ is the total number of flows that use link L_j, and $\tau_j = C_j/n_j$ is the saturated average bandwidth share. (a) Illustration of the first iteration, where L_3 is found as the bottleneck link with minimum τ_j. (b) Illustration of the second iteration with the updated FL matrix, where L_4 is then found as the bottleneck link.

while transfer 3 has two. As a result, there are four potential flows in total, including $f_1 : A \to C \to B \to D$, $f_2 : A \to C \to E$, $f_{31} : B \to D \to F$ and $f_{32} : C \to E \to F$. Given the topology and values of link capacities, we aim at finding the max-min fair solution of the rate vector $\{r_1, r_2, r_3\}$ and the flow assignment $\{x_{31}, x_{32}\}$ for transfer 3.

3 APPROACH SPECIFICATION

One of the biggest challenges for multi-source transmission stems from changing the optimized object from an individual 5-tuple flow into a group of flows that belong to the same transfer. Hence, multiple potential bottlenecks might be considered simultaneously. To this end, we design a MultiSource Water-Filling (MS-WF) algorithm which can jointly compute bandwidth allocation and flow assignment while providing global max-min fairness. The key to MS-WF algorithm is a novel transformation with simple equivalent canonical LP. In this section, we first briefly describe the traditional water-filling algorithm and its limitation. Then we present the MS-WF with one multi-source transfer for clear illustration. A general solution that optimizes arbitrary flow transfers is provided afterward. At last we discuss some practical issues of the multi-source framework.

3.1 Flow-link Mapping Matrix and Traditional Water-Filling Algorithm

The proposed MS-WF algorithm is based on a flow-link mapping matrix (FL matrix) with solvable variables. Here we define the FL matrix, and illustrate the traditional water-filling algorithm with this matrix.

Definition 1 (Flow-link Mapping Matrix)**.** The flow-link mapping matrix *(FL matrix)* $\{fl_{ij}\}$ expresses the flow paths and the traffic assignment of each transfer in matrix form. The element $fl_{ij} \in [0, 1]$ is defined as the proportion of flow i in its belonging transfer that uses link L_j.

Water-Filling (WF) Algorithm. The traditional WF algorithm can compute the bandwidth allocations for single-source transmission. Using the FL matrix as an input, we first denote the saturated average bandwidth allocation as $\tau_j = C_j/n_j$, where n_j is the total number of flows that use link L_j. The WF algorithm iteratively finds the minimum τ^* and the corresponding bottleneck link L_{j^*}. Set the bandwidth of the flows that use link L_{j^*} to τ^*. The FL matrix is then updated by subtracting these flows and the bottleneck link to calculate a new set of $\{\tau_j\}$. Such process repeats until all transfers attain their allocated rates, *i.e.,* all flows have bottleneck links.

Example. Consider a single-source version of Figure 1, where we assume transfer 3 only uses source B, so there are 3 flows over the network. Figure 2 illustrates the allocation algorithm for this single-source example. In the first iteration, L_3 is first saturated because τ_3 is of the minimum value $\tau^* = 2$. We allocate the bandwidth of f_1 and f_{31} that traverse L_3 to 2, and then remove the rows of f_1, f_{31} and column L_3. The FL matrix is updated accordingly for the next iteration, where link L_4 is then found as the bottleneck and bandwidth of f_2 is set to 5. By this point, the ultimate solution to the rate allocation for the 3 transfers is (2, 5, 2).

The traditional WF algorithm is successful in obtaining the max-min fair allocation for single-source transmission. Though it is incapable of dealing with multi-source transfers. This is mainly due to the fact that, instead of transfers, the WF algorithm is based on flows. For the example provided in Figure 1, by using the traditional WF algorithm, transfer 3 will have double weights, which is unfair to the others. A naïve solution is to normalize the weight of each transfer, and then to assign an equal share to the flows from different sources. Regarding the example, the elements for f_{31} and f_{32} become 1/2 and 1/2 in the FL matrix, such that each transfer has the same sum weight of 1. The allocation result turns into (8/3, 10/3, 3), which is slightly better than the (2, 5, 2) - using only source B and (2.5, 4, 2.5) - using only source C, namely the single-source cases. However, as we will soon learn, simply sharing the flow weights equally is still not the optimal solution. *The transfer-level max-min fair allocation is conditioned by the optimal flow assignment.*

3.2 MS-WF Algorithm with One Multi-source Transfer

Given the FL matrix and its application, now let's consider a more complex case by adding one multi-source transfer into the network. We assume there are K_m available sources for the particular transfer m, though the flow assignment X_m is unknown and needs to be solved. The MS-WF algorithm continues to use the FL matrix, but

Algorithm 1 MS-WF Algorithm with one multi-source transfer

Input:
Flow-link mapping matrix: $FL = \{fl_{ij}\}$;
Capacity of each link: $\{C_j\}$, $1 \leqslant j \leqslant M$;
Output:
Transmission rate of each transfer: $\{r_i\}$, $1 \leqslant i \leqslant N$;
Flow assignment of transfer m: $\mathcal{X}_m = \{x_{m1}, \ldots, x_{mk}\}$;

- **Step 1:** ▷ Initiation
  $r_i \leftarrow 0, \quad \forall i = 1, \ldots, N$;
  $\mathcal{L} \leftarrow \{L_1, L_2, \ldots, L_M\}$;
- **Step 2:** ▷ Calculate the saturated average bandwidth
  $n_j(\mathcal{X}_m) \leftarrow \sum_i fl_{ij}, \quad \forall j \in \mathcal{L}$;
  $\tau_j(\mathcal{X}_m) \leftarrow C_j / n_j(\mathcal{X}_m), \quad \forall j \in \mathcal{L}$;
- **Step 3:** ▷ Find the bottleneck fair share τ^*
  if $min\{\tau_j(\mathcal{X}_m)\}$ is a constant **then**
    $\mathcal{X}^* \leftarrow \varnothing$;
  else
    $\mathcal{X}^* \leftarrow \mathcal{X}_m | max\ min\{\tau_j(\mathcal{X}_m)\}$;
  end if
  $\tau^* \leftarrow min\{\tau_j(\mathcal{X}_m)\}$;
- **Step 4:** ▷ Set data rate and update the FL matrix
  $\mathcal{L}_{j^*} \leftarrow \{L_j | \tau_j(\mathcal{X}_m) = \tau^*\}$;
  $\mathcal{L} \leftarrow \complement_{\mathcal{L}_{j^*}} \mathcal{L}$;
  for $i | P_i \cap \mathcal{L}_{j^*} \neq \varnothing$ **do**
    $r_i \leftarrow r_i + \tau^*$;
    Remove f_i from FL;
  end for
  Update FL;
- **Step 5:** ▷ Iteration
  if No transfers left **then**
    return $\{r_i\}$ and $\mathcal{X}_m$;
  else
    goto **Step 2**.
  end if

replaces the elements from 0 and 1 as in the traditional WF into the unknown variables in $\mathcal{X}_m$ for transfer m. The MS-WF algorithm with one multi-source transfer (**Algorithm 1**) is described as follows:

- **Step 1:** Start from zero allocation and build the FL matrix with variables $\mathcal{X}_m$.
- **Step 2:** Compute the saturated average bandwidth $\tau_j(\mathcal{X}_m)$ on each link L_j.
- **Step 3:** Find the bottleneck fair share $\tau^* = min\{\tau_j(\mathcal{X}_m)\}$ by solving $\mathcal{X}^* = \mathcal{X}_m | max\ min\{\tau_j(\mathcal{X}_m)\}$.
- **Step 4:** Set the data rate to τ^* for the flows that traverse the bottleneck links, and update the FL matrix by removing those flows and links.
- **Step 5:** Stop if there are no transfers left; otherwise return to Step 2.

In Step 2, similar to the traditional WF algorithm, MS-WF computes $n_j(\mathcal{X}_m)$ by summarizing the elements of column j in the FL matrix. Here $n_j(\mathcal{X}_m)$ is the number of transfers that traverses link L_j. Since transfer m comes from multiple parallel flows, there might be only a part of it using link L_j. Therefore, n_j becomes a function of $\mathcal{X}_m$ instead of an integer value as in the traditional WF. After that, MS-WF calculates the average bandwidth $\tau_j(\mathcal{X}_m) = C_j / n_j(\mathcal{X}_m)$, which is also a function of $\mathcal{X}_m$.

	L_1	L_2	L_3	L_4	L_5	L_6
f_1	1	1	1	0	0	0
f_2	1	0	0	1	0	0
f_{31}	0	0	x	0	0	x
f_{32}	0	0	0	$1-x$	$1-x$	0
C_j	8	5	4	5	7	6
n_j	2	1	$1+x$	$2-x$	$1-x$	x
τ_j	4	5	$\frac{4}{1+x}$	$\frac{5}{2-x}$	$\frac{7}{1-x}$	$\frac{6}{x}$

Figure 3: An example of MS-WF, where transfer 3 comes as two flows. x is the flow assignment variable to be calculated. τ_j in orange cell is found as the minimum bottleneck share.

In Step 3, potential bottleneck links are found given $\{\tau_j(\mathcal{X}_m)\}$ on the current link set. Specifically, the flow assignment is determined by $\mathcal{X}^* = \mathcal{X}_m | max\ min\{\tau_j(\mathcal{X}_m)\}$. We use τ^* to denote the minimum bandwidth share, and the set of $\{L_{j^*}\}$ is found as the bottleneck links. As all the variables $\mathcal{X}_m$ are within a certain range $[0, 1]$, $min\{\tau_j(\mathcal{X}_m)\}$ is sometimes a constant value. In that case, no flow assignment variable is calculated, *i.e.*, $\mathcal{X}^* = \varnothing$.

Back to the example in Figure 1, transfer 3 has two available sources to access, leading to two parallel flows f_{31} and f_{32}. Without loss of generality, we use only one variable x to denote the proportion of f_{31}, so the proportion of f_{32} will be $1 - x$. Figure 3 illustrates the FL matrix, followed by the saturated average bandwidths $\{\tau_j(\mathcal{X}_m)\}$.

In MS-WF, Step 3 finding $\mathcal{X}^*$ that maximizes $min\{\tau_j(\mathcal{X}_m)\}$, is the major challenge. Specifically, it is to find $x^* = x | max\ min(4, 5, 4/(1+x), 5/(2-x), 7/(1-x), 6/x)$ in Figure 3. First, as formulated in **Problem 1**, it is a nonlinear programming problem, which can not be directly solved. Second, even though there is a linear expression, the solution needs long sequences of LPs for the max-min objective (multi-objective), which are computationally intense in practice.

Problem 1 (The nonlinear optimization problem in Step 3)**.**

$$max \quad min\{\tau_j(\mathcal{X}_m)\}, \tag{4}$$

$$s.t. \quad \sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{5}$$

$$0 \leq x_{ik} \leq 1 \qquad \forall i, k. \tag{6}$$

To this end, we propose a novel transformation which converts the nonlinear optimization problem into a canonical form of LP problem based on **Theorem 1**. The equivalent canonical LP problem is defined in **Problem 2**, which can be efficiently solved under limited computational complexity.

Problem 2 (The equivalent canonical LP problem in Step 3)**.**

$$min \quad t \tag{7}$$

$$s.t. \quad t \geq \tau'_j(\mathcal{X}_m) \qquad \forall j, \tag{8}$$

$$\sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{9}$$

$$0 \leq x_{ik} \leq 1 \qquad \forall i,k. \tag{10}$$

Theorem 1. *Problem 1 is equivalent to Problem 2 as a canonical LP problem, where* $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$.

Proof. Given an arbitrary instance of the FL matrix, $\tau_j(\mathcal{X}_m)$ satisfies two conditions: i) $\tau_j(\mathcal{X}_m) \geq 0$, and ii) the inverse of $\tau_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$. So we let $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$, and the objective of $max\, min\{\tau_j(\mathcal{X}_m)\}$ is then equivalent to $min\, max\{\tau'_j(\mathcal{X}_m)\}$, which becomes linear accordingly. Next, we introduce a temporary variable $t = max\{\tau'_j(\mathcal{X}_m)\}$, and use a sequence of inequality constraints $t \geq \tau'_j(\mathcal{X}_m)$ for all j to express t. Since $\tau'_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$, the constraint (8) as a set of inequalities is also linear. In the end, the optimization problem in Step 3 (**Problem 1**) turns into an equivalent canonical LP problem (**Problem 2**), where the decision variables to be solved are the flow assignment set $\mathcal{X}_m$ and t. □

Consequently, the optimization problem in Figure 3 can be transformed into a simple equivalent LP problem shown as follows.

$$min \quad t \tag{11}$$

$$s.t. \quad t \geq \frac{1}{4}, \tag{12}$$

$$t \geq \frac{1}{5}, \tag{13}$$

$$4t - x \geq 1, \tag{14}$$

$$5t + x \geq 2, \tag{15}$$

$$7t + x \geq 1, \tag{16}$$

$$6t - x \geq 0, \tag{17}$$

$$0 \leq x \leq 1. \tag{18}$$

The results of the above LP come out as $x = 1/3$ and $t = 1/3$. So $\tau^* = 1/t = 3$ is the minimum fair share, and L_3 and L_4 are the bottleneck links that are saturated in this iteration. We set $r_1 = r_2 = 3$ as the bandwidth of f_1 and f_2, and $r_3 = 3$ as the sum bandwidth of f_{31} and f_{32}. Meanwhile, the data volume assignment of transfer 3 concludes with 1/3 from source B and 2/3 from source C. The final rate allocation to the 3 transfers are (3, 3, 3), which is more max-min fair than the allocation (2, 5, 2) where transfer 3 uses only source B (as in Figure 2), as well as the allocation (2.5, 4, 2.5) where transfer 3 uses only source C. In addition, if we don't consider the impact of date volume and assume each transfer has the equal volume of $3Gbits$, then MS-WF outperforms the single source approaches in terms of the average completion time (MS-WF: (3/3+3/3+3/3)/3=1, source B: (3/2+3/5+3/2)/3=1.2) and source C: (3/2.5+3/4+3/2.5)/3=1.05), as well as total completion time (MS-WF: 3/3=1, source B: 3/2=1.5 and source C: 3/2.5=1.2).

3.3 General MS-WF

Having solved the preliminary instance with one multi-source transfer, now we consider the general MS-WF with arbitrary transfer combinations. Here the variables become the flow assignments $\{\mathcal{X}_i\}$ ($i \in [1, N]$) for all transfers. Except for the transfers with a single source, whose $\mathcal{X}_i = \{1\}$ for all the time, the rest of $\{\mathcal{X}_i\}$ are to be computed by MS-WF. The main challenge is that the flow assignments of different transfers correlate to each other and can not be calculated independently. One transfer's assignment plan may affect another's optimal decision. *Therefore, the max-min fair allocation requires joint calculation for all flow assignments.*

The general MS-WF mainly follows the procedures in **Algorithm 1**. Exceptionally, we put all sets of the variables $\{\mathcal{X}_i\}$ into the FL matrix, such that τ_j turns into a function of $\{\mathcal{X}_1, ..., \mathcal{X}_N\}$. By the same token, we transform the nonlinear optimization problem in Step 3 into an equivalent canonical LP problem with the help of one additional decision variable t. In each iteration, parts of the flow assignment sets are solved by LP based on **Theorem 2**. Then we plug the values into the FL matrix and remove them from $\{\mathcal{X}\}$. Continue iterating until all flow assignment variables $\mathcal{X}_i$ are determined. Finally, we sum up the constituent flow rates as the multi-source transfer rate, *i.e.,* $r_i = \sum_{k=1}^{K_i} r_{ik}$, where K_i is the total source number of transfer i. The detailed proof of **Theorem 2** can be found in our technical report [1].

Theorem 2. *Multiple sets of variables* $\mathcal{X}_i$ *can be jointly calculated by MS-WF.*

3.4 Discussion

Extension. The MS-WF algorithm is scalable and extensible for more complex use cases. For instance, differentiated qualities of service lead to transfers with variations in priority or other requirements, such that MS-WF is capable of supporting weighted fair allocation by taking priority factors into account. The max-min fair principle can also be applied to different optimization objectives (*e.g.,* transfer completion time), and the adaption of MS-WF with consideration of data size can likewise yield the optimal results for multi-source transfers.

Applicability. The driving application scenario for the MS-WF algorithm is the Large Hadron Collider (LHC) network, which requires deadline scheduling of large-scale datasets (*e.g.,* petabytes) to be transferred around over 180 member sites all over the world [5]. Fairness among large-scale science dataflows is one of the most important metrics for the LHC network, and the current scheduling system performs poorly in terms of fairness due to missing a fairness-aware scheduling framework for the multi-source transfers. Our multi-source transmission framework is part of the preproduction deployment of Unicorn [18], an SDN geo-distributed data analytic system in the CMS (one of the largest scientific experiments in the LHC network).

The experimental SDN system relies on a logically centralized controller to orchestrate bulk transfers. Since the system is for large-scale datasets, the average flow duration may vary from hours to several days. The computation complexity is significantly reduced in MS-WF by transforming a nonlinear multi-objective problem into a single LP. Experimental results show that the computation time for 1000 concurrent flows in MS-WF is at most 40 seconds, and it can be further reduced by removing the redundant constraints in implementation [12]. Therefore, our approach is scalable to practically handle large-scale datasets and networks.

4 PERFORMANCE EVALUATION

To evaluate the performance of MS-WF algorithm on a large-scale network, we develop a simulator on a datacenter network.

4.1 Simulation Methodology

Topologies: Our experiments were conducted by emulating a 3-tier datacenter network topology with 8:1 oversubscription. The topology contains 64 servers; the capacity of each edge link is $1Gbps$; the capacity of the aggregated link is $10Gbps$.
Workloads: We synthesize a stream of transmission requests with a total number of 1000. A Poisson process is used to model the arrival of requests; the arrival rate λ is defined as the average number of new transfers per time slot. We set the slot length to be only a second for fast simulation. A transfer has multiple sources with probability ρ. We assume that a multi-source transfer have a random number of replicas between [2,5], which are randomly placed in servers. We ignore the fluctuation of transfer size in the simulations, and assume a uniform size V for all transfers.
Performance metrics: We use *average transfer completion time* and *network throughput*to show the improvements of MS-WF.
Alternative approaches: We compare the following bandwidth allocation approaches, each of which adopts the traditional WF algorithm.

- **Best-source:** This approach selects a best replica source based on the algorithm in [16] for multi-source transmission.
- **Equal-share:** This approach splits a transfer across different sources equally. For instance, each replica will send 1/3 of the data, if a transfer has 3 replicas.
- **Random-source:** This approach randomly selects an available source for each transfer.

4.2 Simulation Results

Figure 4 displays the simulation results of network throughput, for which the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$ for all of the 1000 transfers. Shown by the results, Random-source approach, disregarding the dissimilarity of the sources, performs the worst, and retains a constant throughput value. Equal-share approach takes advantage of source diversity in a naïve manner; when we have limited diversity to a small number of multi-source transfers (at low multi-source probabilities), equal flow sharing can approximate the optimal assignment, therefore obtaining a similar performance as MS-WF. But as the multi-source proportion rises, the 1000 transfers lead to more potential flows. The effect of "bad flows" enlarges, and therefore pulls down the overall throughput improvement. This way, Best-source approach outperforms Equal-share. MS-WF achieves a much higher throughput than the others by jointly optimizing the bandwidth allocation and flow assignment, leading to higher network utilization. When all transfers have multiple sources ($\rho = 1$), compared with the Random-source transmission, MS-WF obtains a substantial throughput gain for up to 52%.

Figure 5 compares the transfer completion time versus three factors respectively: the multi-source probability ρ, the transfer size V and the transfer arrival rate λ. We observe directly from the figure: across all parameter configurations, MS-WF achieves the smallest transfer completion time. This improvement is mainly acquired from a more max-min fair allocation by MS-WF.

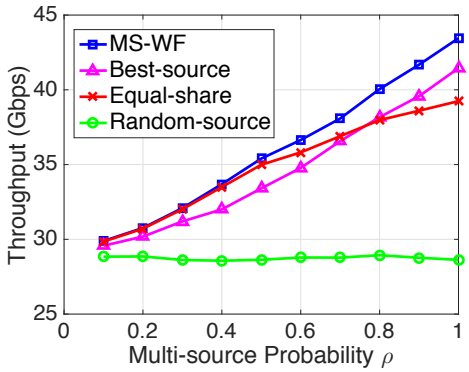


Figure 4: Network throughput vs. multi-source probability ρ, where the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$.

Figure 5(a) illustrates the impact of ρ, which approximately equals to the proportion of multi-source transfers. The constant gap between Random-source and the other approaches demonstrates that, leveraging multiple sources is more efficient for data transmission and can perform much better by placing more replicas in the network. Compared to single-source transmission, MS-WF cuts the average completion time down by up to 44%. Figure 5(b) shows the relationship with V. As transfer size goes up, the transfer backlog begins to cause more bottleneck links, which results in degradation of transmission rates. Therefore, the completion time surges super-linearly along with the transfer size for all the allocation approaches. But by completing transfers as quickly as possible, MS-WF is able to achieve almost linear completion time growth. This way, it is capable of optimizing data transfers for both small and large files. Figure 5(c) illustrates the impact of λ. As expected, at higher arrival rates, the number of flows is likely to be increased, and links are more likely to become congested. Accordingly, the performance degrades quickly for all approaches other than MS-WF. The smaller growth in completion time demonstrates that, by effectively avoiding the congestion point, MS-WF manages to handle a relatively larger amount of traffic without degrading the performance.

5 RELATED WORK

Distributed filesystems. Several high-performance distributed filesystems with sufficient data replicas have been developed, including GFS [7], HDFS [17] and Quantacast File System [15]. Leveraging SDN, Mayflower [16] performs global optimizations to make intelligent replica selection and flow scheduling decisions based on both filesystem and network information. Nevertheless, current solutions focus heavily on best replica selection and data replication placement, instead of multi-source transmission as in our work. As shown in the proceeding sections, single-source transmission fails to achieve transfer-level max-min fairness, therefore provides sub-optimal performance.
Coflow scheduling. The works that schedule parallel flows have been developed to optimize transfers at the level of coflow rather than individual ones. Coflow [3], Varys [4] and Barrat [6] improve

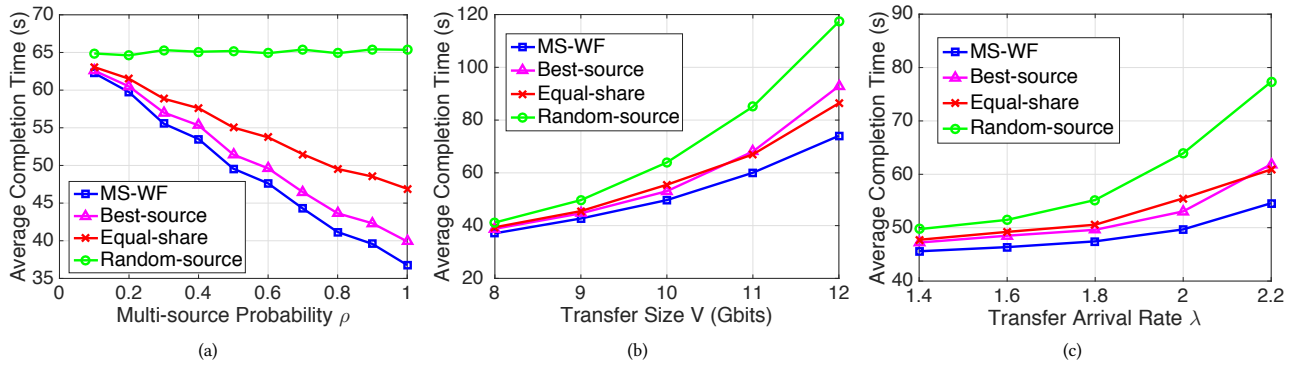


Figure 5: Impact of (a) the multi-source probability ρ, (b) the data size V and (c) the transfer arrival rate λ on average completion time. In each subfigure, we adjust one factor and fixed the other two. The three default values are $\rho = 0.5$, $V = 10$ and $\lambda = 2$.

application-level performance by minimizing coflow completion times and guaranteeing predictable completions. However, their basic assumption is that the flows are streamed for different data and the volume of each flow is designated in advance, so they can easily predict the completion time and allocate the rate to meet their deadlines. The improvement of our approach over them is that the flow volume assignment is jointly optimized with bandwidth allocation to achieve global optimality.

6 CONCLUSION

We present a novel max-min fair allocation approach for multi-source transmission which conveys data in parallel from multiple sources and dynamically adjusts the flow volumes to maximize network utilization. The allocation relies on a MultiSource Water-Filling algorithm that jointly computes the bandwidth allocation and flow assignment with simple equivalent canonical LP to achieve global optimality. Extensive simulations validate that, compared to other single-source and multi-source allocation approaches, our approach achieves a better throughput gain of up to 52% and decreases transfer completion time by up to 44% for large-scale transfers. We believe this approach is applicable to various traffic management systems that orchestrate arbitrary bulk transfers. Developing such systems will be the next step of this research.

ACKNOWLEDGMENT

This research was supported in part by NSFC #61701347, NSFC #61702373 and NSFC #61672385; NSF grant #1440745, CC*IIE Integration; Google Research Award, SDN Programming Using Just Minimal Abstractions. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Anoymous Technical Report. https://github.com/technical-report-2018/CCR2018.
[2] Mosharaf Chowdhury, Srikanth Kandula, and Ion Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *ACM SIGCOMM CCR*, volume 43, 2013.
[3] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of HotNet*, 2012.
[4] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM CCR*, 2014.
[5] CMS Collaboration et al. The cms experiment at the cern lhc (journal of instrumentation 3 s08004, 2008) p. 158.
[6] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM CCR*, 2014.
[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, 2003.
[8] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *SIGCOMM CCR*, 2013.
[9] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *SIGCOMM CCR*, 2013.
[10] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. Optimizing bulk transfers with software-defined optical wan. In *Proceedings of SIGCOMM 2016 Conference*.
[11] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *SIGCOMM CCR*, 2015.
[12] Geng Li, Yichen Qian, Lili Liu, and Y Richard Yang. Jms: Joint bandwidth allocation and flow assignment for transfers with multiple sources. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018.
[13] Qingming Ma, Peter Steenkiste, and Hui Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, 1996.
[14] Dritan Nace and Michal Pioro. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications Surveys & Tutorials*, 10(4):5–17, 2009.
[15] Michael Ovsiannikov, Silvius Rus, Damian Reeves, Paul Sutter, Sriram Rao, and Jim Kelly. The quantcast file system. *Proceedings of the VLDB Endowment*, 2013.
[16] Sajjad Rizvi, Xi Li, Bernard Wong, Fiodar Kazhamiaka, and Benjamin Cassell. Mayflower: Improving distributed filesystem performance through sdn/filesystem co-design. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, 2016.
[17] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, 2010.
[18] Qiao Xiang, X. Tony Wang, J. Jensen Zhang, Harvey Newman, Y. Richard Yang, and Y. Jace Liu. Unicorn: Resource Orchestration for Multi-Domain, Geo-Distributed Data Analytics. https://datatracker.ietf.org/doc/draft-xiang-alto-multidomain-analytics, 2018.

An Objective-Driven On-Demand Network Abstraction for Adaptive Applications

Kai Gao, *Member, IEEE*, Qiao Xiang, *Member, IEEE*, Xin Wang, Yang Richard Yang, *Member, IEEE*, and Jun Bi, *Senior Member, IEEE*

***Abstract*— Revealing an abstract view of the network is essential for the new paradigm of developing network-aware adaptive applications that can fully leverage the available computation and storage resources and achieve better business values. In this paper, we introduce ONV, a novel abstraction of flow-based on-demand network view. The ONV models network views as linear constraints on network-related variables in application-layer objective functions, and provides "equivalent" network views that allow applications to achieve the same optimal objectives as if they have the global information. We prove the lower bound for the number of links contained in an equivalent network view, and propose two algorithms to effectively calculate on-demand equivalent network views. We evaluate the efficacy and the efficiency of our algorithms extensively with real-world topologies. Evaluations demonstrate that the ONV can simplify the network up to 80% while maintaining an equivalent view of the network. Even for a large network with more than 25 000 links and a request containing 3000 flows, the result can be effectively computed in less than 1 min on a commodity server.**

***Index Terms*— Software-defined networking, routing algebra, quality of service, resource abstraction.**

I. Introduction

LARGE-SCALE distributed systems, such as geo-distributed data centers [1] and international scientific research programs [2], [3], have components (data centers, sites, etc.) located in different cities, countries and even continents. To ensure connectivity and minimal performance guarantees, these components are usually connected by tunnels with resource reservations.

Advanced network management technologies such as Software Defined Networking (SDN) have enabled network service providers to provide on-demand resource reservations, such as AT&T's Domain 2.0 [4] and ESNet's OSCAR system [5]. Network tenants can adjust the reservations flexibly to better match their demands.

However, demands of large-scale distributed systems are usually not fixed because of data replications and service load balancing – the same transfer job can be done using different components. In the meantime, orchestration systems such as Microsoft's Clarinet system [6] have been developed to optimize the large-scale query jobs across different geo-distributed data centers based on real-time inter-connection qualities, *i.e.*, *the optimal demands of such systems depend on the available resources*.

It is quite common that tenants decide their optimal demands based on *a certain objective function of available resources and end-to-end metrics*, such as high tunnel utilization [1], flow completion time [7], job completion time [6], throughput [8], etc. Without the ability to accurately know the available resources, a tenant can only make blind guesses which may lead to conservative or unrealistic reservations, and hurt the tenant's quality of service. Thus, it's becoming increasingly important that network service providers offer *on-demand resource abstractions* to help tenants better exploit the flexibility of *on-demand resource reservations*.

SDN enables a network to collect information from all the devices and construct a global view, which may contain essential quality of service (QoS) metrics such as available bandwidth, loss rate and routing cost values, which are critical to performance of distributed applications.

Unfortunately, while northbound APIs for "apps" (management programs) to access the global view have been provided by many SDN controllers (*e.g.*, [9]–[12]), they are usually not open to non-administrative parties. Major concerns include privacy and security, because the global view contains sensitive information that can be leveraged to conduct attacks on the SDN infrastructure [13]. Also, the global view is not friendly to program with because it can contain a lot of redundant information and lead to unnecessary communication overhead.

Thus, a problem arises on how to provide an abstract network view which can both eliminate these drawbacks and

Manuscript received November 4, 2017; revised June 9, 2018; accepted January 25, 2019; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Mascolo. Date of publication March 26, 2019; date of current version April 16, 2019. This work was supported in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Grant W911NF-16-3-0001, in part by the National Science Foundation under Grant CC-IIE 1440745, in part by the National Natural Science Foundation of China under Grant 61472213 and Grant 61502267, and in part by the National Key Research and Development Plan of China under Grant 2017YFB0801701. *(Corresponding author: Kai Gao.)*

K. Gao was with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Department of Computer Science, Yale University, New Haven, CT 06511 USA. He is now with the College of Cybersecurity, Sichuan University, Chengdu 610065, China (e-mail: kaigao@scu.edu.cn).

Q. Xiang and Y. R. Yang are with the Department of Computer Science, Yale University, New Haven, CT 06511 USA (e-mail: qiao.xiang@yale.edu; yry@cs.yale.edu).

X. Wang is with the Department of Computer Science and Technology, Tongji University, Shanghai 201804, China, and also with the Key Laboratory of Embedded System and Service Computing, Ministry of Education, Beijing 100816, China (e-mail: 13xinwang@tongji.edu.cn).

J. Bi is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: junbi@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TNET.2019.2899905

still provide high-quality information. It is non-trivial because of the following challenges:

- *Feasibility*: A decision made with the abstract view should also be feasible in the original network. Infeasible reservations are either rejected, or lead to over-subscribed tunnels which may eventually lead to congestion.
- *Generality*: The abstract view should be general enough to provide fine-grained information and suffice the demands of applications with heterogeneous objectives.
- *Optimality*: A decision made with the abstract view should be as optimal as with the original network information. A suboptimal solution will affect the quality of service and cannot fully utilize the network resources.
- *Privacy*: The abstract view must be able to protect the privacy of the network service provider, making it difficult for malicious applications to infer the global information.
- *Efficiency*: The abstract view should not introduce too much computation/communication overhead, even with moderately large networks and workloads.

Existing abstractions [14]–[23] usually target at a certain type of scenario and cannot support applications which require fine-grained QoS metrics. For example, many of these abstractions cannot accurately represent bottlenecks shared by multiple correlated flows in an arbitrary network, which is critical in emerging use cases such as geo-distributed data centers [1], [24], [25] and scientific computing platforms [26].

In this paper, we take the first step towards providing high-quality network information for network-aware adaptive applications with ONV, an abstraction for flow-based On-demand Network View. Based on the observation that network information is eventually used by applications to conduct optimizations, ONV provides *equivalent* network view which satisfies the aforementioned properties simultaneously.

The main contributions in this paper include:

- We systematically investigate the problem of providing on-demand network view for network-aware adaptive applications with heterogeneous optimization goals, a missing functionality from current SDN northbound API design.
- We address the challenges by proposing ONV, an abstraction for flow-based on-demand network view. ONV is based on the concept of *equivalent network view*. We derive the criteria of equivalent network view and gives the lower bound of number of links.
- We propose two algorithms which conduct equivalent network transformations to obtain the equivalent network view.
- We implement a prototype of ONV and evaluate its performance using real applications over simulated networks of real topologies. Evaluations show that ONV guarantees both feasibility and optimality, improves privacy by reducing information leak, and reduces the communication overhead by a factor of 1.25 to 5 even for large networks (real ISP network topologies with up 10000 nodes and 30000 links) and large workloads (more than 3000 flows).

The rest of the paper is organized as follows. We summarize the demands for fine-grained network views, existing abstractions and their limitations in Section II. Formal descriptions of the abstraction problem and the *equivalent network view* are then given in Section III and Section IV respectively, followed by the transformation algorithms in Section V. We evaluate the prototype and analyze the results in Section VI. Finally, we discuss the related work and give conclusions in Section VII and Section VIII respectively.

II. Motivation

In this section we discuss the motivation that drives our research. See the network in Fig. 1(a). Assume an application (a web service provider) has three services colored in red, blue and brown respectively on the left-hand side, while there are six clients using different services on the right-hand side. Assume the red service is live streaming, blue is video subscribing and brown is large file downloading. All three services require high bandwidth so it is important to know the bottlenecks in the network. Thus, the application sends a request on the bandwidth correlation of the six flows to the network. Meanwhile, the application does not want the network to know about how it would manage the services. Thus, it does not provide any further information.

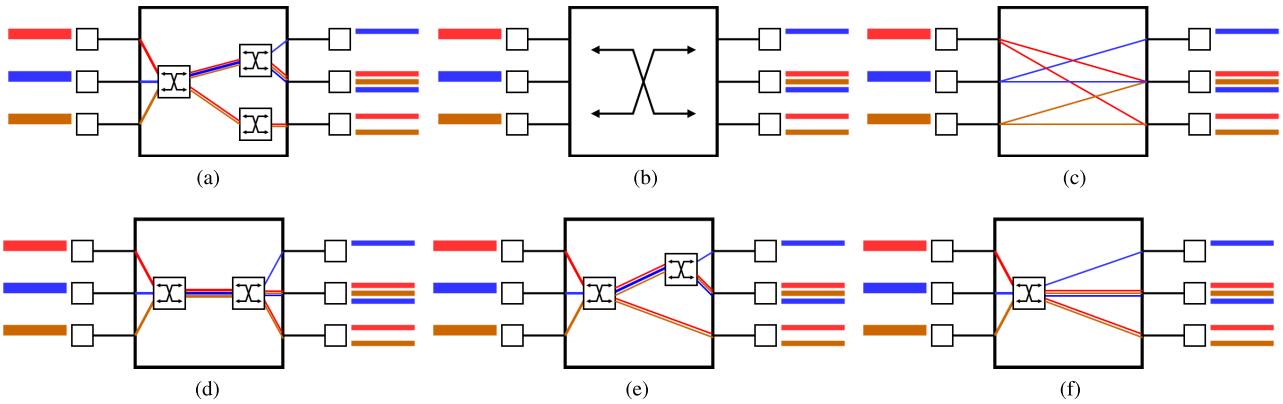
The naive approach returns the slice containing 1) all the links on the flow paths with the associated bandwidth information, and 2) how the links are shared by the flows, as shown in Fig. 1(a) in this case. However, this can lead to information leaks so that malicious applications may leverage this service to infer the network information, which jeopardizes the privacy of the network service provider. Also as the network size increases, the topology cannot be effectively represented.

The hose model, also known as the *one-big-switch* abstraction, returns the network as a single big switch as demonstrated in Fig. 1(b). However, the application would only know the available bandwidth on the ingress/egress "port". If the bottleneck is the upper middle link, it is not propagated to the application. Thus, the application may incorrectly increase the traffic for the blue flows without knowing that they are correlated with r_1, which leads to congestion. Because of the TCP congestion control mechanism, congestion would reduce the throughput of all the flows sharing the same link, leading to an overall performance degradation.

The end-to-end abstraction as demonstrated in Fig. 1(c) is completely useless in this case. It has the same problem as the *one-big-switch* abstraction that information about the bottleneck within the network cannot be accurately provided to the application.

Topology aggregation [22] is a common technique to reduce the topology size. However, it also suffers from the incapability of providing accurate information about the flow correlations. What is worse, if not aggregated correctly as in Fig. 1(d), it may introduce unnecessary bottlenecks that lead to suboptimal decisions right in the beginning.

A simple observation is that the lower middle link and the lower right link are both shared by r_2 and y_2 only. Thus, we can aggregate them together as a new virtual link, as demonstrated in Fig. 1(e). One may think we can just delete one of them. However, if the application asks for the



The application has 6 flows: 2 red, 2 blue and 2 brown. The flows are labeled as b_1, r_1, y_1, b_2, r_2 and y_2 from top to bottom.

Fig. 1. Comparison between different network view abstractions. (a) The sliced network view. (b) One-big-switch abstraction. (c) End-to-end abstraction. (d) Incorrect topology aggregation. (e) Simple equivalent aggregation. (f) Advanced equivalent transformation.

end-to-end routing metrics such as hop count at the same time, deletion would return incorrect values for the two flows.

Meanwhile, if we already know that the upper middle link would not be the bottleneck, the network view can be further reduced, as demonstrated in Fig. 1(f). It is worth noticing that just like the case with the simple equivalent aggregation, we cannot just delete it if end-to-end metrics are also requested.

Thus, the question arises on how we can determine what kind of links can be reduced and how to reduce them correctly. In order to answer this question, we introduce the concept of *equivalent network view* and propose ONV with efficient algorithms to compute the equivalent network view.

III. On-Demand Network Views

In this section, we formally define the model of *on-demand network views* and the theoretical foundations – the variant routing metric algebra, and the unified network element. We also discuss how to encode on-demand network views.

A. Basic Settings

We first formally define the models for the networks and applications discussed in this paper, as summarized in Table I.

1) Network: A network is a graph of arbitrary topology consisting of a set of M unified network elements or simply *element* (defined in Section III-C). For each element, the network can provide two kinds of routing metrics:

- A *flow-independent* metric represents a metric whose value is *independent* of the flow correlations, *i.e.*, the existence of a flow would not affect the value of another flow's flow-independent metric sharing the same network element. Common flow-independent metrics used in QoS routing [27] include hop count, delay, and loss rate.[1] We also require these metrics to be *linearly addictive*, *i.e.*, for a given metric w and two network elements a and b, $w(a+b) = w(a) \oplus w(b)$.

[1]Delay, and loss rate are sensitive to traffic volumes so their real time values are not flow independent. However, they are considered flow independent when measured *statistically*, as used in network tomography [28].

TABLE I
Symbols for Network View Abstraction

Scope	Symbol	Meaning
Network	M	Number of network elements
	$\mathcal{E}$	Set of network elements, $\mathcal{E} = \{e_1, \ldots, e_M\}$
	K_i	Number of flow-independent metrics
	K_c	Number of flow-correlated metrics
	$p_{j,k}$	The k-th flow-independent metric of e_j
	P	The matrix of flow-independent metrics
	$q_{j,k}$	The k-th flow-correlated metric of e_j
	Q	The matrix of flow-correlated metrics
Application	N	The number of flows (potential tunnels)
	$\mathcal{F}$	The set of flows $\mathcal{F} = \{f_1, \ldots, f_N\}$
	$x_{i,k}$	The k-th end-to-end metric for f_i
	X	The matrix of end-to-end metric
	$y_{i,k}$	f_i's allocation of the k-th resource
	Y	The matrix of resource allocations
	K_p	Number of private variables
	z_k	The k-th private variable
	Z	The 1-row matrix of private variables
	U	The objective function
	K_{pc}	Number of private constraints
	g_k	The k-th private constraint function
Routing	$a_{i,j}$	Whether element j appears in flow i's path
	A	Routing matrix
Routing algebra	$\mathcal{P}$	Set of paths
	$\mathcal{S}$	Set of metrics
Network view	V	Network view represented by a triplet
	$V_j(\boldsymbol{v_j})$	The j-th component (and index vector) of V
	T	The view-abstraction transformation
Common	mat^i	The i-th column of matrix *mat*
	mat_i	The i-th row of matrix *mat*

- A *flow-correlated* metric represents a network resource that is shared among flows. Bandwidth is the most common and also the most important *flow-correlated* metric. Other shared resources such as flow entries or middlebox-related metrics may also exist in certain scenarios. We require the resource constraints to be linear, *i.e.*, for a given resource and two flows f_1 and f_2, $w(\{f_1\}) + w(\{f_2\}) = w(\{f_1, f_2\})$.

Without loss of generality, we number the elements from 1 to M where the j-th element is denoted as e_j. Let $\mathcal{E} = \{e_1, \ldots, e_M\}$. Assume each element has K_i flow-independent metrics and K_c flow-correlated metrics.

TABLE II
THE VARIANT ROUTING METRIC ALGEBRA

S	Weight function (w)	$w(p_1)$	$w(p_2)$	$w(p_1 \circ p_2) = w(p1) \oplus w(p_2)$	$N \otimes w(p_1)$	Identity (e)	Zero ($\mathbf{0}$)
$\mathbb{N}^+$	Hopcount	h_1	h_2	$h_1 + h_2$	$N \cdot h_1$	0	$+\infty$
$\mathbb{R}^+$	Bandwidth	b_1	b_2	$\min(b_1, b_2)$	b_1	$+\infty$	0
$\mathbb{R}^+$	Delay	d_1	d_2	$d_1 + d_2$	$N \cdot d_1$	0	$+\infty$
$[0, 1]$	Loss rate	r_1	r_2	$1 - (1 - r_1)(1 - r_2)$	$1 - (1 - r_1)^N$	0	1

For the j-th element e_j, the k-th flow-independent metric is denoted as $p_{j,k}$ and the k-th flow-correlated metric is denoted as $q_{j,k}$. Let $P = (p_{j,k})_{M \times K_i}$ and $Q = (q_{j,k})_{M \times K_c}$.

2) Application: Each application contains a set of N unidirectional flows, which is based on the observation that applications may have asymmetric traffic demands. An application has a *private* objective function, which depends on three types of information as listed below and is subject to some private constraints:

- Per-flow end-to-end metrics: An end-to-end metric represents the end-to-end performance a certain flow can obtain, such as delay, loss rate, etc. End-to-end metrics correspond to the flow-independent metrics.
- Per-flow resource allocations: A per-flow resource allocation represents how much resource (e.g., bandwidth) a given flow can use, and corresponds to a flow-correlated metric.
- Private variables: A private variable represents information that is not known by the network provider, such as CPU utilization, available memory or even bandwidth constraints in a private network.

The flows are numbered from 1 to N and the i-th flow is denoted as f_i. Let $\mathcal{F} = \{f_1, \ldots, f_N\}$. Assume K_i represents the number of end-to-end metrics, K_c represents the number of resources, and K_p represents the number of private variables. For the i-th flow, we use $x_{i,k}$ to denote its k-th end-to-end metric and $y_{i,k}$ to denote its allocation for the k-th resource. We number the private variables from 1 to K_p and denote the k-th variable as z_k.

We use $U(X, Y, Z)$ to represent the private objective function, where $X = (x_{i,k})_{N \times K_i}$, $Y = (y_{i,k})_{N \times K_c}$, and $Z = (z_k)_{1 \times K_p}$. Without loss of generality, we assume smaller objective values indicate better results. For objective functions that an application wants to maximize, we can easily construct $U'(X, Y, Z) = -U(X, Y, Z)$ and use U' as the objective function instead.

Assume there are K_{pc} private constraints, and the i-th private constraint is denoted as $g_i(X, Y, Z) \leq 0$. Both U[2] and g_i are arbitrary functions with the mild assumption that the application can find the minimum objective under all private constraints and additional linear constraints on X and Y.

3) Routing: We consider the case where the tunnels are simple paths, *i.e.*, there are no loops or branches. Let $a_{i,j}$ denote whether f_i traverses e_j and routing matrix $A = (a_{i,j})_{N \times M}$.

[2]Even though the correctness of this paper does not have any additional assumption on U as long as the application can solve it under linear constraints and the given set of private constraints, a common assumption in practice is that the objective function is concave.

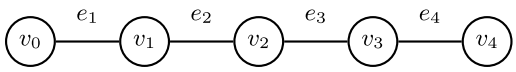


Fig. 2. Topology to demonstrate different definitions of path.

B. The Variant Routing Metric Algebra

The routing metric algebra, introduced by Sobrinho to implement QoS-based routing [29], is based on *path concatenation*. It can be represented as $(\mathcal{P}, \mathcal{S}, w, \circ, \oplus, \preceq)$. $\mathcal{P}$ is the set of paths and $\mathcal{S}$ is a closed set of metrics. The weight function w maps a path from $\mathcal{P}$ to a given metric in $\mathcal{S}$. The concatenation operator $\circ$ is a binary operator on $\mathcal{P}$, which takes two paths and returns a new one. Operator $\oplus$ is a binary operator on $\mathcal{S}$ and $\preceq$ is a binary relation on $\mathcal{S}$.

In this paper, we introduce a variant of the routing metric algebra. First, to better formulate the constraints on *flow-independent* metrics, we introduce a new $\otimes : \mathbb{N} \times \mathcal{S} \mapsto \mathcal{S}$ operator to linearize the metric calculation. The operator basically means the same link is traversed for multiple times.

Second, we relax the constraint on path concatenation in the original algebra, by extending the meaning of path from a walk of nodes to a set of *unified network elements* (defined in Section III-C). Consider the network in Fig. 2, the only valid path from v_0 to v_4 in the original QoS algebra is $\langle v_0, v_1, v_2, v_3, v_4 \rangle$, but with our algebra a path can be *any* permutation of $\{e_1, e_2, e_3, e_4\}$ (24 combinations).

We use $(\mathcal{P}, \mathcal{S}, w, \circ, \oplus, \preceq, \otimes)$ to describe our variant routing algebra. $(\mathcal{S}, \oplus)$ is a semigroup so that $\oplus$ is *commutative* and *associative*. We also require that the $\otimes$ operator is *distributive* over $\oplus$. Concrete examples of some common routing metrics are demonstrated in Table II.

C. Unified Network Elements

Traditional graph representations of a network would treat links and nodes differently because the routing capability is only provided on nodes (switches/routers/middleboxes). However, since routing is not a mandatory functionality in our network view definition, we can generalize the nodes and links as *unified network elements* to simplify the representation.

For example, a deep packet inspection (DPI) middlebox may have a maximum processing speed, which yields a constraint on the total throughput passing through this DPI node. From the applications' perspective, it is not different from a bottleneck link.

However, certain metrics may only appear on certain types of network elements. To guarantee that these unified network elements would not affect the results of routing metric algebra,

we must alter the weight function w as:

$$w^*(p) = \begin{cases} w(p) & \text{if } w(p) \text{ exists} \\ e & \text{otherwise} \end{cases}$$

where e is the *identity* of w and some concrete examples are given in Table II.

Links, nodes and middleboxes are transformed into unified elements differently. For example, a duplex link should be treated as two unified network elements because flows could traverse it from both directions. Meanwhile, a middlebox should be treated as a single element because flows from different directions are throttled by the computation capability – a single bottleneck.

Unified network elements have several benefits. First, this unified representation of links/nodes greatly simplifies our analysis. Second, it provides a high-level abstraction which focuses on the metrics' semantics rather than how the metrics are computed/constrained.

D. Abstract Network View

We use the symbols defined in Section III-A and formally define the on-demand network view.

Flow-independent metrics are formulated as *equations* according to the variant routing metric algebra. For example, the hop count between two end hosts is equal to the sum of hop counts of each link on the path. We have:

$$x_{i,k} = \bigoplus_j a_{i,j} \otimes p_{j,k} \Leftrightarrow X = A \times P.$$

If f_i consumes the same resource on all network elements it traverses, the *flow-correlated* metrics are formulated as *linear constraints*. The total resource consumption of all the flows on a single element must not exceed the available amount, i.e.,

$$\sum_i a_{i,j} y_{i,k} \leq q_{j,k} \Leftrightarrow A^{\mathrm{T}} Y \leq Q.$$

Thus, the network view can be formulated as a tuple of three elements: $V = (A, P, Q)$. Based on this network view model, an *abstraction* can be defined as follows:

Definition 1 (Network View Abstraction): A network view abstraction is a transform function T which takes the original network view V and returns an *abstract* view V', i.e.,

$$V'(A, P, Q) = T\left(V(A, P, Q)\right)$$

E. Encoding Abstract Network Views

We use *flow path map* and *element map* to efficiently encode an abstract network view. The flow path map, as the name suggests, is a dictionary object which maps a flow identifier to its flow path. Each flow path is a list of *element identifiers*. Each element identifier uniquely represents a unified network element, and each appearance in a flow path indicates that the flow traverses the corresponding network element once. The element map is a dictionary object which maps an element identifier to its properties. Properties are encoded in a `key: value` style. The Backus-Naur Form (BNF) for our abstract network view encoding is given in Fig. 3. An example is given

⟨AbstractNetwokView⟩ ::= { ⟨FlowPathMap⟩ , ⟨ElementMap⟩ }
⟨FlowPathMap⟩ ::= "flow-path-map" : {⟨FlowPathEntries⟩}
⟨FlowPathEntries⟩ ::= ⟨FlowPathEntry⟩ | ⟨FlowPathEntry⟩ , ⟨FlowPathEntries⟩
⟨FlowPathEntry⟩ ::= *FlowId* : ⟨FlowPath⟩
⟨FlowPath⟩ ::= *ElementId* | *ElementId* , ⟨FlowPath⟩
⟨ElementMap⟩ ::= "element-map" : { ⟨ElementEntries⟩ }
⟨ElementEntries⟩ ::= ⟨ElementEntry⟩ | ⟨ElementEntry⟩ , ⟨ElementEntries⟩
⟨ElementEntry⟩ ::= *ElementId* : ⟨ElementAttributes⟩
⟨ElementAttributes⟩ ::= ⟨ElementAttribute⟩ | ⟨ElementAttribute⟩ , ⟨ElementAttributes⟩
⟨ElementAttribute⟩ ::= *AttributeName* : *AttributeValue*

Fig. 3. The BNF for abstract network view encoding.

```
{
  "flow-path-map": {
    "f1": ["e1", "e2", "e3", "e4"],
    "f2": ["e5", "e6", "e7", "e8"]
  },
  "element-map": {
    "e1": { "routingcost": 1, "bandwidth": "100Mbps" },
    "e2": { "routingcost": 1, "bandwidth": "100Mbps" },
    "e3": { "routingcost": 1, "bandwidth": "50Mbps"  },
    "e4": { "routingcost": 1, "bandwidth": "100Mbps" },
    "e5": { "routingcost": 1, "bandwidth": "100Mbps" },
    "e6": { "routingcost": 1, "bandwidth": "100Mbps" },
    "e7": { "routingcost": 1, "bandwidth": "50Mbps"  },
    "e8": { "routingcost": 1, "bandwidth": "100Mbps" }
  }
}
```

Fig. 4. Encoding an abstract network view for Fig. 2.

in Fig. 4, which uses the topology in Fig. 2 with one end-to-end metric ("routingcost") and one flow-correlated metric ("bandwidth"). There are two flows, one from v_0 to v_4 and the other from v_4 to v_0. Each link is denoted as two elements, where $e_i = (v_{i-1}, v_i)$ for $i \in [1, 4]$ and $e_i = (v_{i-4}, v_{i-5})$ for $i \in [5, 8]$. However, this abstract network view is not optimal because it contains some redundant information.

Since the number of flows is fixed, the size of an abstract network view is mostly determined by the number of elements and their appearances in the flow path map. We use $\|V\|$ to denote the size of a view, which is measured by the number of unified network elements.

IV. Equivalent Network View

In this section, we introduce *equivalent network view* and an effective criterion to verify the equivalence. Furthermore, we give a lower bound of the number of unified network elements contained in an equivalent on-demand network view.

A. Equivalence of On-Demand Network Views

A key insight is that the returned information is the input parameters of an objective function, whose result can help applications make decisions. If the applications can make the same optimized decision with an abstract network view, we can say the abstract network view is *equivalent*.

Consider an application as we model in Section III-A, the optimization problem can be formulated as

$$\begin{aligned} U_{\min} = \ & \min U(X, Y, Z) \\ & s.t\ X = A \times P, \\ & \quad O \leq A^{\mathrm{T}} Y \leq Q, \\ & \quad g_i(X, Y, Z) \leq 0, \quad \forall i \in [1, K_{pc}]. \end{aligned}$$

Let $U_{\min}(V)$ denote the optimal objective for a given network view V. We define "equivalence" as follows:

Definition 2 (Network View Equivalence): Two network views V and V' are *equivalent*, if and only if for any application with flows $\mathcal{F}$ and objective function U, $U_{\min}(V) = U_{\min}(V')$.

We use the symbol "$\sim$" to represent the network view equivalence. It can be easily proved that the network view equivalence is an *equivalence relation*. We also propose the criterion in Theorem 1 to simplify the verification.

Theorem 1 (Network View Equivalence Criterion): Two network views $V(A, P, Q)$ and $V'(A', P', Q')$ are *equivalent* if and only if for any application with flows $\mathcal{F}$

$$A \times P = A' \times P' \tag{1a}$$

$$R = \left\{Y \mid A^{\mathrm{T}}Y \leq Q\right\} = \left\{Y \mid A'^{\mathrm{T}}Y \leq Q'\right\} = R' \tag{1b}$$

Proof: For two network views V and V', we use $V \sim^* V'$ and $V \sim V'$ to represent that V and V' are equivalent by the criterion and are equivalent by definition respectively. It can be easily proved that if $V \sim^* V'$, $V \sim V'$ because they have exactly the same domain space and thus the same image space. For the other direction, we prove it by contradiction.

We first assume that there exists $V \sim V'$ but $V \nsim^* V'$, i.e., either Equation 1a or Equation 1b does not hold.

Assume Equation 1a does not hold. For any $X = A \times P \neq A' \times P' = X'$, we find one entry that is not equal in X and X', say $x_{i,k} \neq x'_{i,k}$ and construct an objective function $U(X, Y, Z) = x_{i,k}$ or $U(X, Y, Z) = -x_{i,k}$ based on whether smaller $x_{i,k}$ is better. Thus, the two network views have different optimal values and are not equivalent by definition, which contradicts with our assumption.

Assume Equation 1b does not hold, $\exists \hat{Y} \in (R \setminus R') \cup (R' \setminus R)$. Without loss of generality, let $\hat{Y} \in R \setminus R'$. Since $\hat{Y} \notin R'$, there exist j, k such that $(A'^{\mathrm{T}}\hat{Y})_{j,k} = \hat{u} > q_{j,k}$. Now we construct a linear objective function $U(X, Y, Z) = -(A'^{\mathrm{T}}Y)_{j,k}$, and assume the optimal objectives are u and u' respectively. We have $u \leq -\hat{u} < -q_{j,k} = u'$ which means the objective function has different optimal objective values for the two network views. Again, we get a contradiction.

Thus, we can conclude that if $V \sim V'$, $V \sim^* V'$ and the criterion is both sufficient and necessary. □

We have proved that the network views satisfying this criterion also satisfy Definition 2. Thus, it allows us to effectively verify two network views are equivalent for applications with arbitrary objective functions and arbitrary fine-grained routing metrics as long as they fit in the *variant routing metric algebra*.

B. Lower Bound of $\|V\|$

For a given on-demand network view, there exists a set of equivalent network views which construct an equivalent class. To improve privacy and reduce communication overhead, one might want to return the *minimal* equivalent network view.

In this section, we present two extreme cases:

1) only flow-independent information is requested, and
2) only flow-correlated information is requested.

As we demonstrate in Section V-B, the general case can be processed in two stages and each stage corresponds to an extreme case.

1) Lower Bound of $\|V\|$ for Flow-Independent Metrics: When only flow-independent information is requested, the equivalent network view only needs to provide the same X for each flow. Since $\|V\|$ equals the column rank of A, the equivalent network view with the minimal $\|V\|$ has the smallest column rank of A. The problem is equivalent to the *optimal non-negative matrix factorization problem*, which has been proved to be NP-Hard [30].

2) Lower Bound of $\|V\|$ for Flow-Correlated Information: When only flow-correlated information is requested, equivalent network views should return the *same feasible region*, which is determined by a set of linear constraints. Since each constraint is essentially one network element, $\|V\|$ is directly related to the number of constraints.

We first introduce the definition of redundant linear constraint by Telgen [31] and propose Theorem 2.

Definition 3 (Redundant Linear Constraint – elgen [31]): For a linear system whose feasible region $R = \left\{\vec{x} \mid A\vec{x} \leq \vec{b}\right\}$, the k-the constraint $A_k\vec{x} \leq b_k$ is redundant if and only if the feasible region $R_k = \{\vec{x} \mid A_i\vec{x} \leq b_i, i \neq k\}$ is equal to R, i.e. $R_k = R$.

Theorem 2: If only flow-correlated information is requested, $\|V\|$ is minimal if and only if the corresponding constraint set $C = \{c_j | c_j : A_j^{\mathrm{T}}Y \leq Q_j\}$ has no redundant constraints.

Proof: $\Rightarrow$: Consider the opposite that $\|V\|$ is minimal but C contains redundant constraints. According to the definition of *redundant constraints* by Telgen [31], we can remove the redundant constraints but still obtain the same feasible region. Thus, we have a network view with a smaller $\|V\|$ and it leads to a contradiction.

$\Leftarrow$: Consider the opposite that C contains no redundant constraints but $\|V\|$ is not minimal. Let C' represents the equivalent constraint set of the minimal size and $\|C'\| < \|C\|$. Since C and C' have the same feasible region, they also have the same feasible region as $C \cup C'$. Since C contains no redundant constraints, there exists at least one $c^* \in (C \cup C') \setminus C$ which is redundant. Thus, C' contains a redundant constraint and cannot have the minimal number of element, which leads to a contradiction with our assumption. □

The problem of finding redundant linear constraints has been widely studied. For example, Paulraj and Sumathi [32] have summarized several algorithms to find the redundant constraints. Since there exists polynomial time algorithms for linear programming [33], the problem can also be solved in polynomial time.

V. Equivalent Network View Transformations

In this section, we introduce ONV, the *On-demand Network View Abstraction* which conducts equivalent transformations to obtain an equivalent network view. ONV consists of two algorithms, namely *equivalent element aggregation* and *equivalent element decomposition*. We prove both algorithms guarantee the *equivalence* condition, and analyze how they can improve *efficiency* and *privacy*.

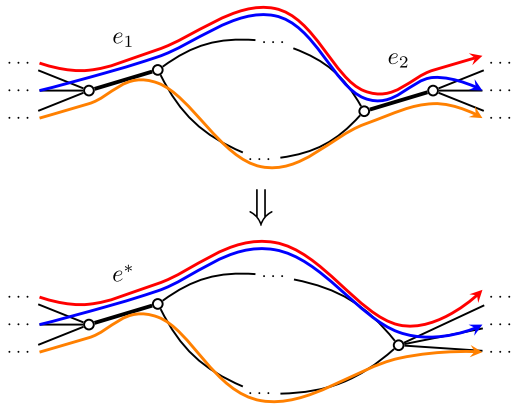


Fig. 5. Equivalent element aggregation.

Algorithm 1: Equivalent Element Aggregation

Input: $V(A, P, Q)$
Output: $V'(A', P', Q')$
1 **Function** EQUIVAGGREGATION*(V)*
2 $\mathcal{V} \leftarrow \{V_j \mid V_j \leftarrow (A^j, P_j, Q_j), 1 \le j \le M\}$;
3 $\mathcal{G} \leftarrow \text{GROUPBY}(\mathcal{V}, V_j \Rightarrow (v_j \leftarrow A^j, V_i))$;
4 **for** $G_j \in \mathcal{G}$ **do**
5 $V'_j \leftarrow \text{AGGREGATE}(v_j, \{V_{m_{j,1}}, \ldots, V_{m_{j,|G_j|}}\})$;
6 $M' \leftarrow |\mathcal{G}|$;
7 $V' \leftarrow \left(\left[A'^1 \cdots A'^{M'}\right], \begin{bmatrix} P'_1 \\ \vdots \\ P'_{M'} \end{bmatrix}, \begin{bmatrix} Q'_1 \\ \vdots \\ Q'_{M'} \end{bmatrix}\right)$;
8 **return** V'
9 **Function** AGGREGATE*(*$v_j, \{V_{m_{j,1}}, \ldots, V_{m_{j,|G_j|}}\}$*)*
10 $A'^j = v_j$;
11 $P'_j \leftarrow \left[\bigoplus_i p_{m_{j,i},1}, \ldots, \bigoplus_i p_{m_{j,i},K_i}\right]$;
12 $Q'_j \leftarrow \left[\min_i q_{m_{j,i},1}, \ldots, \min_i q_{m_{j,i},K_c}\right]$;
13 **return** (A'^j, P'_j, Q'_j)

A. Equivalent Element Aggregation

In this section, we introduce the *equivalent aggregation*. The example in Fig. 5 demonstrates the intuition: There are three flows with different colors and only they traverse both e_1 and e_2, so that we "aggregate" them together as a single element e^*. We give an algorithm in Algorithm 1 which guarantees that the resulted network view is equivalent to the original one. We analyze its efficiency and prove its correctness.

The network view is represented as row vectors (components), as demonstrated in Line 2. Line 3 groups the j-th *component* $V_j(A^j, P_j, Q_j)$ using the j-th row vector in A, A^j, as the key. M' denotes the number of groups, and the index of the k-th member in the j-th group is denoted as $m_{j,k}$. Line 5 computes the aggregation of the components in each group. Finally Line 7 constructs the new network view by merging all the aggregated components. For each component V_j, the time complexity for the grouping and the aggregation is $O(N(K_i + K_c))$ and $O(N(K_i + K_c))$ respectively while the MERGE process is totally logical, which yields a total time of $O(MN(K_i + K_c))$.

Now we prove the element aggregation algorithm is correct, in the sense that it maintains the equivalence condition.

Theorem 3: $V' \leftarrow \text{EQUIVAGGREGATION}(V)$, $V' \sim V$.

Proof: Assume $g_{j,i}$ represents the index of the i-th components in G_j, and let $b_j \leftarrow \min_i m_{j,i}$ and $c_{j,k} \leftarrow \arg\min q_{m_{j,i},k}$.

First we check Equation 1a is met. Since $X = A \times P$, we have

$$
\begin{aligned}
& x_{i,k} \\
&= \bigoplus_j a_{i,j} \otimes p_{j,k} = \bigoplus_{1 \le j' \le M'} \left(\bigoplus_{1 \le l \le |G_{j'}|} a_{i,m_{j',l}} \otimes p_{m_{j',l},k} \right) \\
&= \bigoplus_{1 \le j' \le M'} \left(\bigoplus_{1 \le l \le |G_{j'}|} a_{i,b_{j'}} \otimes p_{m_{j',l},k} \right) \\
&= \bigoplus_{1 \le j' \le M'} a_{i,b_{j'}} \otimes \left(\bigoplus_{1 \le l \le |G_{j'}|} p_{m_{j',l},k} \right) \\
&= \bigoplus_{1 \le j' \le M'} a_{i,b_{j'}} \otimes p'_{j',k} = x'_{i,k}.
\end{aligned}
$$

The key steps are based on the facts that $\oplus$ is transitive and commutative, $\otimes$ is distributive over $\oplus$, and $\forall l \in [1, |G_{j'}|], a_{i,b_{j'}} = a_{i,m_{j',l}}$.

Now we check Equation 1b. We have

$$
\begin{aligned}
R &= \left\{Y \mid A_j^{\mathsf{T}} Y^k \le q_{j,k}, \forall j, k\right\} \\
R' &= \left\{Y \mid A'^{\mathsf{T}}_j Y^k \le q'_{j,k}, \forall j, k\right\} \\
&= \left\{Y \mid A'^{\mathsf{T}}_{c_{j',k}} Y^k \le q_{c_{j',k},k}, \forall j', k\right\}.
\end{aligned}
$$

Since the constraints of R' is a subset of R, $R \subseteq R'$. If $R' \neq R$, $\exists \hat{Y} \in R' \setminus R$, meaning $\hat{Y}$ at least violates one constraint in R, say for $\hat{j}$ and $\hat{k}$, *i.e.*,

$$
A_{\hat{j}}^{\mathsf{T}} \hat{Y}^{\hat{k}} > q_{\hat{j},\hat{k}} \ge \min_l q_{m_{\hat{j},l},\hat{k}} = q_{c_{\hat{j},\hat{k}},\hat{k}},
$$

which means $\hat{Y}$ also violates one constraint in R' and leads to contradiction with our assumption. So we have $R = R'$.

By Theorem 1, $V' \sim V$. □

B. Equivalent Element Decomposition

In this section, we introduce the details of *equivalent element decomposition*, which can substantially improve the performance of *equivalent element aggregation*.

Algorithm 1 guarantees the equivalence condition which is important to prove the correctness of ONV, however, the condition to aggregate components is not easy to be satisfied without further processing. Thus, we need to conduct another equivalent transformation in practice, namely *equivalent element decomposition*. An example of equivalent element decomposition is given in Fig. 6, where only the red flow traverses e_a, only the blue flow traverses e_b and only the red

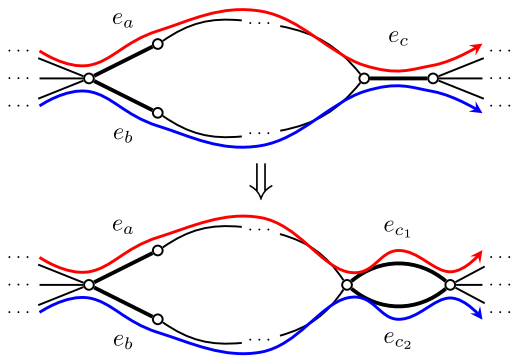


Fig. 6. Equivalent element decomposition.

and the blue flows traverse e_c. The three elements have the following metrics:

$$e_a : \textit{routingcost} = 1, \quad \textit{bandwidth} = 100\text{Mbps}$$
$$e_b : \textit{routingcost} = 2, \quad \textit{bandwidth} = 100\text{Mbps}$$
$$e_c : \textit{routingcost} = 3, \quad \textit{bandwidth} = 200\text{Mbps}$$
$$A^a = \begin{bmatrix}1\\0\end{bmatrix}, \quad A^b = \begin{bmatrix}0\\1\end{bmatrix}, \; A^c = \begin{bmatrix}1\\1\end{bmatrix}.$$

According to grouping condition, there will be three different groups. But we can make the observation that since the constraint for e_c: $bw_1 + bw_2 \leq 200$ is *redundant*, we can decompose e_c as two unified network elements e_{c_1} and e_{c_2} where

$$e_{c_1} : \textit{routingcost} = 3, \quad \textit{bandwidth} = 200\text{Mbps}$$
$$e_{c_2} : \textit{routingcost} = 3, \quad \textit{bandwidth} = 200\text{Mbps}$$
$$A^{c_1} = \begin{bmatrix}1\\0\end{bmatrix}, \quad A^{c_2} = \begin{bmatrix}0\\1\end{bmatrix}.$$

After e_c is decomposed, we can invoke EQUIVAGGREGATION (Algorithm 1) and e_{c_1} and e_{c_2} can be aggregated with a and b respectively.

Theorem 4 gives the condition when an element can be safely decomposed without affecting the equivalence condition. The efficiency and privacy of equivalent decomposition depend on 1) how to identify redundant components, and 2) how to find the "basis". The first step corresponds to finding the minimal equivalent network view with only flow-correlated information, as discussed in Section IV-B.2, while the second step is similar to finding the minimal equivalent network view with flow-independent information, with the constraints that non-redundant network elements must be contained. Since the second step has been proved to be NP-Hard, in this paper, we use a heuristic approach which aims to simplify the selection of basis, as introduced in Algorithm 2.

Theorem 4: For $V_j(A^j, P_j, Q_j)$, we say V_j is redundant if and only if $A_j^{\intercal} Y^k \leq q_{j,k}$ is redundant for all k. If and only if V_j is redundant, we can construct an equivalent network view $V' = V \setminus V_j \cup \{V_{j,l}\}$ where V_j is decomposed as $V_{j,l}(A'^l, P'_l, Q'_l)$ with $A^j = \sum_l A'^l$, $P_j = P'_l$ and $Q_j = Q'_l$.

Proof: We still consider the criteria Equation 1a and Equation 1b and use the same symbols in Theorem 3.

First we can prove criterion Equation 1a holds whether V_j is redundant or not.

$$\begin{aligned} x_{i,k} &= \bigoplus_u a_{i,u} \otimes p_{u,k} = \bigoplus_{u \neq j} a_{i,u} \otimes p_{u,k} + a_{i,j} \otimes p_{j,k} \\ &= \bigoplus_{u \neq j} a_{i,u} \otimes p_{u,k} + \left(\sum_l a'_{i,l} \right) \otimes p_{j,k} \\ &= \bigoplus_{u \neq j} a_{i,u} \otimes p_{u,k} + \bigoplus_l a'_{i,l} \otimes p_{j,k} \\ &= \bigoplus_{u \neq j} a_{i,u} \otimes p_{u,k} + \bigoplus_l a'_{i,l} \otimes p'_{l,k} = x'_{i,k}. \end{aligned}$$

For Equation 1b, first we consider the case when V_j is redundant but $V \nsim V'$. V_j is redundant so that $\forall k$, $A_j^{\intercal} X^k \leq q_{j,k}$ is redundant. According to Definition 3, we have feasible regions $R = R_j = R'$ for all k. Since we have already proved that Equation 1a holds, according to Theorem 1, $V \sim V'$ which leads contradiction.

If V_j is not redundant but $V \sim V'$, we can similarly construct a contradiction between the definition of redundancy and the equivalence criterion.

Thus, we have proved that V_j can be equivalently decomposed if and only if V_j is redundant. □

We introduce the concept of *dominance* of components. From Theorem 4, we can easily conclude that if an element can be decomposed, it dominates all the elements in the basis.

Definition 4 (Dominance of Components): We say a component $\boldsymbol{V_j}$ is *dominated* by another component $\boldsymbol{V_{j'}}$, if and only if, $\forall i$, $a_{i,j} \leq a_{i,j'}$.

Now we present the details of Algorithm 2. Line 2 identifies the set of decomposable components $\mathcal{D}$ according to Theorem 4, i.e. $\forall V_j \in \mathcal{D}$, V_j is redundant. In each iteration (Line 4-11), we try to decompose a decomposable elements into other network elements greedily, in the sense that $\forall l$, V_l is dominated by V_j, we decompose V_j to V_l and its complement. If the routing matrix for V_j is empty, it means V_j is decomposed to a set of V_ls. Otherwise, V_j cannot be

Algorithm 2: Equivalent Element Decomposition

Input: $V(A, P, Q)$
Output: $V'(A', P', Q')$
1 **Function** EQUIVDECOMPOSITION$(V, \mathcal{F})$
2 $\quad \mathcal{D} \leftarrow$ FINDEQUIVDECOMPOSABLE(V)
3 $\quad V' \leftarrow V$
4 $\quad$ **foreach** $V_j \in \mathcal{D}$ **do**
5 $\quad\quad V' \leftarrow V' \setminus \{V_j\}$
6 $\quad\quad$ **foreach** $V_l \in V'$ **do**
7 $\quad\quad\quad$ **if** V_j *can be decomposed to* V_l **then**
8 $\quad\quad\quad\quad V_l \leftarrow (A^l, P_l \oplus P_j, Q_l)$
9 $\quad\quad\quad\quad V_j \leftarrow (A^j - A^l, P_j, Q_j)$
10 $\quad\quad$ **if** $A^j \neq \vec{0}$ **then**
11 $\quad\quad\quad V' \leftarrow V' \cup \{V_j\}$
12 $\quad$ **return** V'

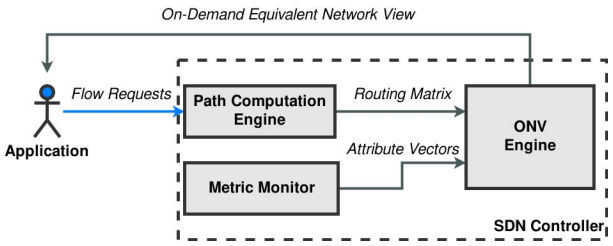


Fig. 7. The architecture of ONV.

decomposed to existing components so its remnant must be added back (Line 11).

Finally we invoke EQUIVAGGREGATION(V') to aggregate V'_j with the same A^j, which is also proved to maintain the equivalence condition as in Theorem 3. Thus, Algorithm 2 returns the equivalent network view.

For each iteration, the decomposition needs to check whether a component is dominated by another one, which may take $O(N)$ time and yield a total running time of $O(M^2N^2)$. Since we have assumed that flows only traverse simple paths, we encode the routing matrix for a component, which is a binary vector of size $N \times 1$, as a binary integer. This pre-processing step takes $O(MN)$ time. Thus, the dominance can be verified in $O(\log(N))$ time using bit `and` operation. The update on Line 8 takes only $O(K_i)$ time. The subtraction on Line 9 can be done using the bit `xor` operation, which also takes $O(\log(N))$ time. The routing matrix A^j can be reconstructed in $O(N)$ time in the outer loop. There are at most M^2 inner iterations and M outer iterations, so the total execution time is $O(M^2(\log(N) + K_i) + MN)$.

C. Privacy Enhancement

The equivalent aggregation and equivalent decomposition are equal to matrix transformations. While the application can only infer the network elements which cannot be decomposed without jeopardizing feasibility or optimality, it is impossible to infer the complete original network state without knowing the exact value of the transform matrix. Thus, Algorithm 2 can improve the privacy and reduce information leak.

It is worth noting that ONV does not require applications to specify private information, e.g. private constraints and objective functions. Thus, it also protects the privacy of the applications.

D. System Implementation

The architecture of ONV is demonstrated in Fig. 7. User requests are first sent to the path computation engine, which obtains the routing matrix A. The ONV engine also pulls the attribute vectors, *i.e.*, P and Q from a monitoring component.

EQUIVAGGREGATION is first executed to avoid corner cases in finding redundant resource constraints. If no flow-correlated information is requested, the FINDEQUIVDECOMPOSABLE function returns all network elements. Otherwise, it uses the algorithm in [31] to find network elements with redundant constraints. Finally, EQUIVDECOMPOSITION is executed and the resulted equivalent network view is encoded as in Section III-E and returned to the user.

VI. EVALUATION

In this section, we evaluate ONV extensively to answer the following questions: 1) Can ONV achieve feasibility and optimality for heterogeneous objective functions? 2) How much can ONV simplify the network view? 3) How fast can ONV compute the abstract on-demand network view?

A. Experimental Setup

In this section, we introduce the general experimental setup of our evaluations and leave the methodologies for each experiment to the corresponding section.

Topology and Metrics: We use real-world topologies from two data sets: the topology zoo [34] and rocketfuel [35]. If a topology already has bandwidth information, we use the values directly. Otherwise, we generate stepped values for links from edge to core. We allocate the "routingcost" randomly, following the standard distribution around the reciprocal of bandwidth. The values are multiplied by a given constant to avoid precision issues. For each topology, we generate three different routing cost distributions.

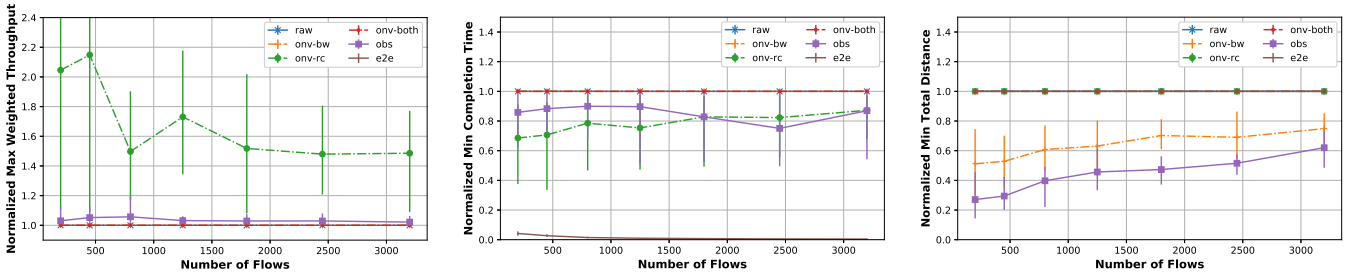
Algorithms to Find Redundant Constraints: To find the redundant network elements, we use the linear programming method introduced in [32].

Abstractions: We use six different network abstractions.

1) The *raw* network view is computed by the naive approach which contains all network elements on the paths.
2) Three various network views are computed by ONV with different guarantees. The *onv-bw* view only guarantees the equivalence of flow-correlated network resources. All decomposable network elements can be directly removed and its $\|V\|$ is equal to the number of non-decomposable network elements. The *onv-rc* view only guarantees the equivalence of flow-independent metrics so it considers all network elements to be decomposable, *i.e.*, Line 2 of Algorithm 2 returns V. The *onv-both* view guarantees equivalent network views where Line 2 of Algorithm 2 finds redundant network elements using an paralleled implementation of the algorithm in [31].
3) The *one-big-switch* (as in SDX [36]) view removes all network elements except the ingress/egress ones.
4) The *e2e* view (as in ALTO [21]) creates a virtual network element for each flow whose attributes are calculated with the variant routing algebra.

Flow Requests: We have 7 groups with different numbers of flows. There are three types of requests, depending on the metrics: *routingcost-only* (*rc*) *bandwidth-only* (*bw*), and *hybrid*. As the names suggest, they represent the cases where 1) only flow-independent metrics are requested, 2) only flow-correlated metrics are requested, and 3) both metric types are requested. The flows used in our benchmark are randomly generated based on the *server-client* communication pattern. We select a given subset of endpoints as servers, and for each server, we pick random endpoints as clients.

Runtime and Data Collection: The prototype system is built with Python and uses the PuLP framework. The *COIN Branch*



(a) (b) (c)

Fig. 8. Normalized results for different objective functions using different abstractions. (a) Results for weighted throughput (wpt). (b) Results for flow completion time (fct). (c) Results for total routing cost (trc).

and Cut (CBC) solver is used solve linear programming problems. The evaluations are emulated on a server with Linux kernel 3.19.0-25, 6 quad-core Intel(R) Xeon(R) E5-2620 v3 @2.40GHz CPU and 128 GB memory.

We collect the results for each ⟨topology+metrics, flow size+distribution, metric types⟩ combination. For each combination, we generate 10 different samples and calculate the average, standard deviation, the minimum and the maximum.

B. Optimality and Feasibility for Heterogeneous Objectives

To understand how different network views may have an impact on the optimality and feasibility of an application's objective functions, we conduct the following evaluation with multiple objective functions.

Objective Functions: We consider three objective functions from existing researches: 1) For the *wtp* objective function case, we give each flow a random weight and optimize the weighted throughput [37]. 2) For the *fct* objective function case, we give each flow a random data size and minimizes the total flow completion time [38]. 3) For the *trc* objective function case, we divide hosts into client and server groups. For each host in the client group, it selects the server node with the smallest routing cost. We sum the total routing cost as the value of the objective function.

Topology and Metrics: We use the Kdl topology (752 nodes, 1790 links) from the topology zoo. Since the coefficients of the objective functions are generated randomly, the values of objective functions may not be useful. Thus, we normalize the results as follows:

$$\text{Normalized} = \frac{\text{Optimal objective using a given view}}{\text{Optimal objective using } \textit{raw}}.$$

The normalized results serve as indicators of whether the corresponding network view guarantees optimality and feasibility. Consider the optimization problem is to maximize a given objective function (as in *wpt*), if the normalized objective value $opt > 1$, it means that the objective value is larger than the real optimal value and thus the value is *infeasible*. On the other hand, if the normalized objective value $opt < 1$, it means that the objective value is smaller than the real optimal one and thus the value is *suboptimal*. For optimization problems that minimize a given objective function (*fct* and *trc*), the conclusion is the opposite. The conditions of whether a network view is *feasible* and *optimal* are listed in Table III.

TABLE III
CONDITIONS FOR FEASIBILITY AND OPTIMALITY

Objective function (normalized)	*Feasible*	*Optimal*
max weighted throughput (*wpt*)	$opt \leq 1$	$opt \geq 1$
min coflow completion time (*fct*)	$opt \geq 1$	$opt \leq 1$
min total routing cost (*trc*)	$opt \geq 1$	$opt \leq 1$

As demonstrated in Fig. 8, we can see that 1) one-big-switch abstraction can lead to *infeasible* solutions in all three cases; 2) end-to-end abstraction achieves both optimality and feasibility for *trc* but can lead to *infeasible* solutions for *wpt* and *fct* objective functions, indicating that it can provide accurate flow-independent information but very inaccurate information on flow-correlated resource (shared bottlenecks); 3) *onv-both* achieves both optimality and feasibility for all cases while the two variants also achieves optimality and feasibility for their targeted use cases, which indicates that ONV can provide accurate information on both flow-correlated information (*onv-bw* and *onv-both*), flow-independent information (*onv-rc* and *onv-both*) and the two types of metrics combined (*onv-both*).

C. Network Simplification

In this section, we demonstrate how much ONV can simplify the network and reduce the communication overhead with the following settings:

Metrics: We use the normalized $\|V\|$ to evaluate how much a network view is simplified and use the number of bytes in the encoded JSON string to evaluate the communication overhead of an abstract network view.

As we can see form Fig. 9, ONV can simplify the network significantly. Specifically, the *bandwidth-equivalent* network view only uses less than 40% of the network elements in the original one for all six topologies except *Colt*. It even uses less network elements than the *one-big-switch* abstraction in certain topologies. While the *routingcost-equivalent* and the *completely equivalent* network views typically contain more elements, they can still reduce 50% to 80% of the network information for 200 flows, and 20% to 60% of the network information even for more than 3,000 flows.

We also analyze the factors that may determine the simplification results. From Fig. 9, we can see that the number of elements increases as the number of flows increases. Thus, we consider the largest flow requests, *i.e.*, with 3,200 flows

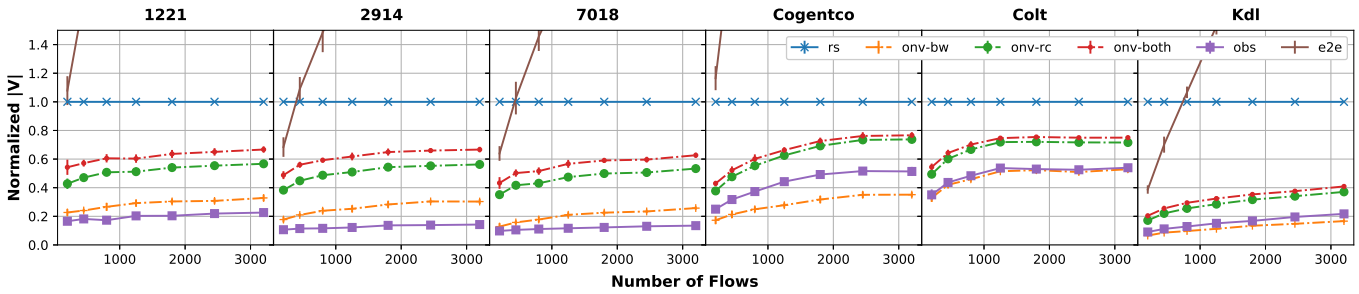
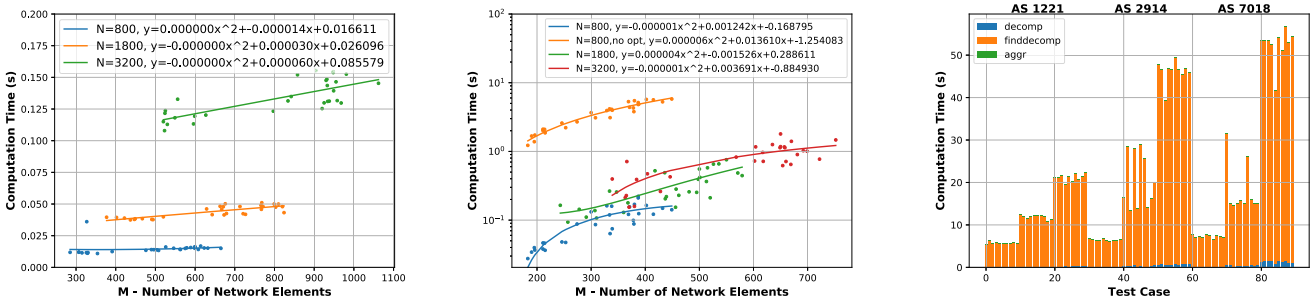


Fig. 9. Normalized $\|V\|$ for different topologies.

TABLE IV
TOPOLOGIES AND THE ABSOLUTE RESULTS FOR 3200 FLOW REQUESTS

Topology	*#Nodes*	*#Edges*	Original View (*raw*)		*Bandwidth-equivalent* View (*onv-bw*)		*End-to-end-equivalent* View (*onv-rc*)		*Equivalent* View (*onv-both*)	
			$\|V\|$	Relative size	$\|V\|$	Relative size	$\|V\|$	Relative size	$\|V\|$	Relative size
AS 1221	5023	6259	566.00±40.63	-	185.40±14.39	0.32	320.80±23.69	0.57	377.20±26.21	0.67
AS 2914	10820	16422	883.90±59.85	-	267.80±20.74	0.30	497.50±40.67	0.56	589.10±42.70	0.67
AS 7018	20593	25199	919.60±25.61	-	236.90±13.32	0.27	490.70±17.22	0.56	576.40±21.86	0.65
Cogentco	197	243	285.90±17.36	-	100.30±5.70	0.35	210.60±8.87	0.74	218.90±10.98	0.77
Colt	153	177	172.30±5.81	-	91.10±3.60	0.53	123.30±4.60	0.71	129.20±5.03	0.75
Kdl	754	895	955.70±39.25	-	159.00±9.68	0.17	354.30±15.34	0.37	390.60±17.40	0.41



(a) (b) (c)

Fig. 10. Computation time. (a) Computation time of EQUIVAGGREGATION. (b) Computation time of EQUIVDECOMPOSITION. (c) Computation time breakdown.

and summarize the absolute values in Table IV.[3] The relative size of a given view is calculated as proportion of network elements compared with the one in the *raw* view, *i.e.*,

$$\text{relative size} = \frac{\|V\| \text{ of a given view}}{\|V\| \text{ of the } \textit{raw} \text{ view}}.$$

As shown in Table IV, the network elements revealed in abstract network views are much less than the ones contained in the topologies, suggesting that on-demand network views can protect the privacy of network providers while providing useful information to network-aware adaptive applications.

D. Computation Time

Methodology: The time complexity depends on both the number of flows N and the number of network elements M. We choose 3 different numbers of flows: 800 (20 servers and 40 clients per server), 1800 (30 servers and 60 clients per server) and 3200 (40 servers and 80 clients per server). For each M, we generate 10 samples for topologies AS 1221, AS 2914 and AS 7018. For equivalent decomposition, we also turn on/off the binary code optimization.

[3]Edges are bidirectional so the total number of elements in a topology is twice the number of edges.

As we can see in Fig. 10(a), the time curve fits well with a linear function of the number of network elements M. The coefficients of x also roughly grows linearly as the number of flows N, which demonstrates the time complexity analysis for Algorithm 1 is correct.

Also, we can also see that in Fig. 10(b), the time curve also fits with a quadratic function of M. While we cannot derive directly from the coefficients that the time complexity is linear with the logarithm of N, the comparison to an unoptimized implementation (*i.e.*, using linear scan as in Definition 4) demonstrates an improvement of 40 to 60.

Finally we demonstrate how much each component contributes to the total execution time. As we can see in Fig. 10(c), the total execution is less than a minute even for resource reservation for a lot of flows in very large scale networks. In particular, both EQUIVAGGREGATION (denoted as *aggr*) and EQUIVDECOMPOSITION (denoted as *decomp*) only take a very small proportion. While the time is sufficient to traditional traffic engineering which may take hours or days, the operator can optionally skip the equivalent decomposition procedure if the application-layer scheduler demands smaller traffic engineering intervals.

E. Summary

In this section, we evaluate the performance of ONV thoroughly. We demonstrate that ONV guarantees both feasibility and optimality for heterogeneous objective functions. ONV can substantially reduce the number of leaked information and also the communication overhead (up to 4.5x improvement). It can produce an abstract on-demand network view within a minute for very large scale networks (>20,000 nodes and >25,000 edges) and flow requests (>3,000 flows). Thus, ONV effectively enables collaborative optimization with non-administrative network-aware adaptive applications.

VII. Related Work

A. Demands for Network Views

The demands for being network-aware are quite common. For services built on top of the Internet, user experience depends heavily on the quality of networking service [39]–[41]. Previous studies [42] have already shown that obtaining end-to-end metrics can significantly improve the user experience of peer-to-peer services and content delivery networks.

Meanwhile, several studies [38], [43], [44] have also addressed the need to conduct flow scheduling over the network, suggesting the importance of obtaining the correlations between different data transfers. Such demands are usually associated with traffic with large volumes, such as inter-data center communication, e.g, Google's globally-deployed B4 [1] system and global data intensive science networks [26]. Feeding these applications with more accurate network information allows them to make more intelligent operating decisions. Network information is also used to optimize video streaming for multiple objectives, such as QoE [45] and fairness [46].

Another example where being aware of the network performance can be beneficial is fine-grained routing. Latest approaches such as the Software Defined Internet Exchange point (SDX) [36] have enabled Autonomous Systems to set up fine-grained forwarding rules. With the ability to query the expected network performance, an AS would be able to make routing decisions based not only on the cost, but also on the real-time quality of service. Meanwhile, such information can also be provided to QoS-based routing protocols [27], [29].

SOL [47] and CoFlow [38] are SDN-based network optimization frameworks which provide abstractions to simplify the modeling of network optimization problems. However, it would require the optimizer to provide all the information to the network, which jeopardizes the privacy. General collaborative optimization [48]–[50] typically protects the privacy by multiplying a monomial matrix. ONV enables collaborative optimization by providing the network views to the optimizer, while conducting equivalent transformations to reduce the communication overhead as well as protect the privacy.

B. Providing Network View

The most straight-forward way of providing network views is to use its graph representation. Several routing protocols [17]–[20] including OSPF and IS-IS conceptually provide such an abstraction of the network and it is also adopted by the I2RS (Interface to Routing System) IETF Working group [51]. Modern SDN controllers [9]–[12] also provide the global view using the annotated graph model.

The hose model [15] is first introduced for VPN provisioning in 1999. Each endpoint is associated with a *hose* in this model and the details of the actual VPN tunnels are hidden. It is sometimes referred to as the *one-big-switch* in the context of SDN because the network is abstracted as a single logical switch in this model. Because of its simplicity, the hose model is widely used, for example, by many network programming languages [52], [53]. SDX also uses this model to encapsulate the underlying network topology. Data center fabrics are highly customized for scalability [16] and can be modeled as a non-blocking switch where congestion only occurs on access links [23], thus, the *one-big-switch* abstraction is also widely used for data center flow scheduling and tenant resource provisioning [38], [43], [44].

The mesh model is mostly used by web-based applications or measurement frameworks, which have no control over the network. The mesh model consists of several flows (host pairs) and provides a single mesh for each flow (pair) with the associated metrics. PerfSONAR [54], Meridian [55] and ClosestNode [56] are some concrete examples which provide such end-to-end network views based on measurement, while P4P [42] and the ALTO (Application-Layer Traffic Optimization) protocol [21] are leveraging the network providers' information.

ONV is similar to ALTO in the sense that in both cases information is provided by the network to non-administrative applications, which is likely to achieve better accuracy. Meanwhile, we overcome the limitations of ALTO by adopting the equivalence abstraction to provide fine-grained metrics, in particular the *flow correlations*, which makes it possible to suffice the demands from a broader range of applications. This underlying philosophy also distinguishes ONV from other (especially QoS related) routing protocols and network views based on topological aggregation [19].

Recently Nikolenko *et al.* [57] has introduced an algorithm to simplify the network topologies, which is also based on the principle of *equivalence* and equivalent network transformations. Compared with their work, we have a different definition of equivalence originated from the applications' perspective. While their work is still transforming the topology, our equivalent transformations are based on a more abstracted network representation which allows us to conduct more sophisticated transformations beyond the topological constraints. Similar ideas are also applied in some newer researches [58], [59].

VIII. Conclusion

In this paper, we systematically study the problem of providing an accurate on-demand network view for application-layer multi-flow optimization. Our abstraction is based on the principle of *equivalence* which guarantees generality, feasibility and optimality. We design the ONV framework to construct equivalent network views and evaluate its performance compared with some well-known network view abstractions. Currently, ONV leverages the SDN technology to provide the network

view service in a centralized way, and leaves distributed on-demand network view as a future extension.

ACKNOWLEDGMENT

The authors would like to thank C. Gu, J. Zhang, S. Chen, X. Lin, H. Wang, and H. Du for their help during the preparation of the paper. This paper is an extension to a conference paper [60], which is also published as a poster earlier [61]. They would also like to thank the anonymous TON reviewers for their valuable feedback. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)* New York, NY, USA, 2013, pp. 3–14.

[2] *Official Public Website for the ATLAS Experiment at CERN*, CERN, Meyrin, Switzerland, 2018. [Online]. Available: https://home.cern/science/experiments/cms

[3] *The Compact Muon Solenoid Experiment*, CERN, ATLAS Exp., Geneva, Switzerland, 2018. [Online]. Available: https://atlas.cern/

[4] *AT&T Vision Alignment Challenge Technology Survey—AT&T Domain 2.0 Vision White Paper*," AT&T, Dallas, TX, USA, 2013.

[5] *OSCARS*, Lawrence Berkeley Nat. Lab., Energy Sci. Netw., Berkeley, CA, USA, 2018. [Online]. Available: https://www.es.net/engineering-services/oscars/

[6] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: WAN-aware optimization for analytics queries," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*. Savannah, GA, USA: USENIX Association, 2016, pp. 435–450.

[7] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *Proc. ACM Conf. SIGCOMM*, New York, NY, USA, 2014, pp. 443–454.

[8] J. Rehn *et al.*, "PhEDEx high-throughput data transfer management system," in *Proc. Comput. High Energy Nucl. Phys. (CHEP)*, 2006, pp. 1–4.

[9] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[10] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, 2010, pp. 1–6.

[11] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, 2014, pp. 1–6.

[12] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2014, pp. 1–6.

[13] S. Gao, Z. Peng, B. Xiao, A. Hu, and K. Ren, "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.

[14] Y. Rekhter, T. Li, and S. Hares, "*A Border Gateway Protocol 4 (BGP-4)*, document RFC 4271, 2005.

[15] N. G. Duffield *et al.*, "A flexible model for resource management in virtual private networks," in *Proc. ACM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, New York, NY, USA, 1999, pp. 95–108.

[16] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. SIGCOMM ACM SIGCOMM Conf. Data Commun.*, New York, NY, USA, 2009, pp. 51–62.

[17] J. Moy, *OSPF Version 2*, document RFC 2178, 1997. [Online]. Available: https://tools.ietf.org/html/rfc2178

[18] D. Oran, *OSI IS-IS Intra-domain Routing Protocol*, document RFC 1142, 1990.

[19] T. Vu, A. Baid, H. Nguyen, and D. Raychaudhuri, "EIR: Edge-aware interdomain routing protocol for the future mobile Internet," WINLAB, Rutgers Univ., New Brunswick, NJ, USA, Tech. Rep. WINLAB-TR-414, 2013.

[20] W. C. Lee, "Topology aggregation for hierarchical routing in ATM networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 2, pp. 82–92, 1995.

[21] R. Alimi, Y. Yang, and R. Penno, *Application-Layer Traffic Optimization (ALTO) Protocol*, document RFC 7285, 2014.

[22] S. Uludag, K.-S. Lui, K. Nahrstedt, and G. Brewster, "Analysis of topology aggregation techniques for QoS routing," *ACM Comput. Surv.*, vol. 39, no. 3, p. 7, 2007.

[23] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. O. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. WIOV*, 2011, pp. 1–8.

[24] A. Kumar *et al.*, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, New York, NY, USA, 2015, pp. 1–14.

[25] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 854–862.

[26] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science DMZ: A network design pattern for data-intensive science," *Sci. Program.*, vol. 22, no. 2, pp. 173–185, 2014.

[27] H. Geng, X. Shi, X. Yin, Z. Wang, and H. Zhang, "Algebra and algorithms for efficient and correct multipath QoS routing in link state networks," in *Proc. IEEE 23rd Int. Symp. Qual. Service (IWQoS)*, Jun. 2015, pp. 261–266.

[28] M. Rabbat, R. Nowak, and M. Coates, "Multiple source, multiple destination network tomography," in *Proc. IEEE INFOCOM*, vol. 3, Mar. 2004, pp. 1628–1639.

[29] J. L. Sobrinho, "Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 541–550, Aug. 2002.

[30] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM J. Optim.*, vol. 20, no. 3, pp. 1364–1377, Jan. 2010.

[31] J. Telgen, "Identifying redundant constraints and implicit equalities in systems of linear constraints," *Manage. Sci.*, vol. 29, no. 10, pp. 1209–1222, 2002.

[32] S. Paulraj and P. Sumathi, "A comparative study of redundant constraints identification methods in linear programming problems," *Math. Problems Eng.*, vol. 2010, Sep. 2010, Art. no. 723402.

[33] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. 16th Annu. ACM Symp. Theory Comput.*, 1984, pp. 302–311.

[34] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[35] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.

[36] A. Gupta *et al.*, "SDX: A software defined Internet exchange," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 551–562, 2015.

[37] Y. Bartal *et al.*, "Online competitive algorithms for maximizing weighted throughput of unit jobs," in *Proc. Annu. Symp. Theor. Aspects Comput. Sci.* Berlin, Germany: Springer, 2004, pp. 187–198.

[38] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw. (HotNets-XI)*, New York, NY, USA, 2012, pp. 31–36.

[39] R. K. Mok, E. W. W. Chan, and R. K. C. Chang, "Measuring the quality of experience of HTTP video streaming," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM) Workshops*, May 2011, pp. 485–492

[40] N. Zhang, Y. Lee, M. Radhakrishnan, and R. K. Balan, "GameOn: P2P gaming on public transport," in *Proc. MobiSys*, 2015, pp. 105–119.

[41] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network QoS among peer Xbox 360 game players," in *Proc. Int. Conf. Passive Active Netw. Meas.* Berlin, Germany: Springer, 2008, pp. 41–50.

[42] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4P: Provider portal for applications," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)* New York, NY, USA, 2008, pp. 351–362.

[43] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, vol. 10, 2010, p. 19.

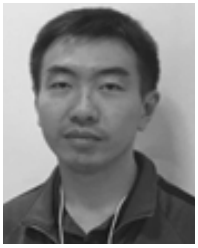
[44] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, New York, NY, USA, 2013, pp. 435–446.

[45] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proc. ACM SIGCOMM Workshop Future Hum.-Centric Multimedia Netw. (FhMN)*, New York, NY, USA, 2013, pp. 15–20.
[46] G. Cofano *et al.*, "Design and performance evaluation of network-assisted control strategies for HTTP adaptive streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 3s, pp. 42:1–42:24, Jun. 2017.
[47] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using SOL," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 223–237.
[48] Y. Hong, "Privacy-preserving collaborative optimization," Ph.D. dissertation, Dept. Manage. Sci. Inf. Syst., Rutgers Univ., New Brunswick, NJ, USA, 2013.
[49] J. Vaidya, "Privacy-preserving linear programming," in *Proc. ACM Symp. Appl. Comput. (SAC)*, New York, NY, USA, 2009, pp. 2002–2007.
[50] J. Li and M. J. Atallah, "Secure and private collaborative linear programming," in *Proc. Collaborative Comput., Netw., Appl. Worksharing, IEEE CollaborateCom Int. Conf.*, Nov. 2006, pp. 1–8.
[51] J. Medved *et al.*, *A Data Model for Network Topologies*, document draft-ietf-i2rs-yang-network-topo-20.txt, Internet Engineering Task Force, Internet-Draft draft-ietf-i2rs-yang-network-topo-11, Feb. 2017.
[52] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Lombard, IL, USA: USENIX Association, 2013, pp. 1–13.
[53] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, New York, NY, USA, 2013, pp. 87–98.
[54] A. Hanemann *et al.*, "PerfSONAR: A service oriented architecture for multi-domain network monitoring," in *Proc. Int. Conf. Service-Oriented Comput.* Berlin, Germany: Springer, 2005, pp. 241–254.
[55] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, New York, NY, USA, 2005, pp. 85–96.
[56] B. Wong and E. G. Sirer, "ClosestNode.Com: An open access, scalable, shared geocast service for distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 62–64, Jan. 2006.
[57] S. I. Nikolenko, K. Kogan, and A. F. Anta, "Network simplification preserving bandwidth and routing capabilities," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.
[58] Q. Xiang *et al.*, "Optimizing in the dark: Learning an optimal solution through a simple interface," in *Proc. AAAI*, Nov. 2018.
[59] Q. Xiang *et al.*, "Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences," in *Proc. Int. Conf. High Perform. Comput, Netw., Storage, Anal. (SC)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 5:1–5:13.
[60] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "NOVA: Towards on-demand equivalent network view abstraction for network optimization," in *Proc. IEEE/ACM 25th Int. Symp Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10.
[61] K. Gao *et al.*, "ORSAP: Abstracting routing state on demand," in *Proc. IEEE 24th Int Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–2.

Kai Gao received the B.S. and Ph.D. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2018.

He is currently an Assistant Research Scientist with the College of Cybersecurity, Sichuan University. His research interests include programming languages and distributed runtime systems for newly emerged networking techniques such as software-define networking and network function virtualization.

Qiao Xiang received the bachelor's degree in information security and the bachelor's degree in economics from Nankai University in 2007, and the master's and Ph.D. degrees in computer science from Wayne State University in 2012 and 2014, respectively. From 2014 to 2015, he was a Post-Doctoral Fellow with the School of Computer Science, McGill University. He is currently an Associate Research Scientist with the Department of Computer Science, Yale University. His research interests include software defined networking, resource discovery and orchestration in collaborative data sciences, interdomain routing, and wireless cyber-physical systems.



Xin Wang received the B.S. degree in computer science from Tongji University, Shanghai, China, in 2013, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology. He is also with the Key Laboratory of Embedded System and Service Computing, Ministry of Education, Beijing, China. His research interests include computer networks, software-defined networks, and distributed computing.

Yang Richard Yang received the B.E. degree in computer science and technology from Tsinghua University in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin in 1998 and 2001, respectively. He is currently a Professor of computer science and electrical engineering with Yale University. His work has been implemented/adopted in products/systems of major companies (e.g., AT&T, Alcatel-Lucent, Cisco, Google, Microsoft, and Youku) and featured in mainstream media, including *Economist*, *Forbes*, *Guardian*, *Information Week*, *MIT Technology Review*, *Science Daily, USA Today, Washington Post,* and *Wired*, among others. His research was supported by both the U.S. government funding agencies and leading industrial corporations. His research interests span areas including computer networks, mobile computing, wireless networking, and network security. His awards include the CAREER Award from the National Science Foundation and the Google Faculty Research Award.

Jun Bi (S'98–A'99–M'00–SM'14) received the B.S., C.S., and Ph.D. degrees from the Department of Computer Science, Tsinghua University, Beijing, China. He is currently a Changjiang Scholar Distinguished Professor with Tsinghua University, where he serves as the Director of the Network Architecture Research Division and the Deputy Dean of the Institute for Network Sciences and Cyberspace. He is also the Director of the Future Network Theory and Application Research Division, Beijing National Research Center for Information Science and Technology. He has published over 200 research papers and 20 Internet RFCs or drafts. He held 30 innovation patents. He has successfully led tens of research projects. His current research interests include Internet architecture, SDN/NFV, and network security. He is a Distinguished Member of the China Computer Federation.

Precedence: Enabling Compact Program Layout By Table Dependency Resolution

Christopher Leet[1], Shenshen Chen[12], Kai Gao[3], Yang Richard Yang[12]
[1] Yale University, [2] Tongji University, [3] Sichuan University

ABSTRACT

The rise of the programmable switching ASIC has allowed switches to handle the complexity and diversity of modern networking programs while meeting the performance demands of modern networks. Exploitation of the flexibility of these switches, however, has exploded routing program size: recently proposed programs contain more than 100 [11] or even 1000 [10] tables. Realizing these programs in a programmable switch requires finding layouts with minimal depth: if a layout has more match-action stages than a switch's pipeline provides, the switch must recirculate, cutting throughput. Even if a layout fits a switch's pipeline, since most commercial pipelines cannot allocate memory freely to stages, non-compact pipelines can result in underloaded stages and significant memory underutilization. While inter-table control and data dependencies critically limit the ability of compilers to lay out tables compactly, no switch architecture which can fully resolve dependencies has been proposed. To address this problem, we introduce **precedence**, an extension of the RMT switching ASIC, which enables tables linked by dependencies to be executed in parallel or even out-of-order. Precedence can resolve nearly **70%** of switch.p4 [11]'s dependencies (a real-world routing program), reduce its pipeline depth by **48%**, and only modestly increases silicon area.

CCS CONCEPTS

• **Networks** → **Routers**; • **Hardware**;

1 INTRODUCTION

The rise of the programmable switching ASIC was a landmark development in modern networking. Programmability allows switches to handle the complexity and diversity of modern routing applications while achieving the performance required of modern networks. As programmers have increasingly exploited the flexibility promised by programmable switches, however, routing program size and complexity has ballooned. Recent routing programs like DC.p4 [23] (43 tables), switch.p4 [11] (163 tables) and HyPer4 [10] (1091 tables) have incredible sophistication.

Realizing a modern program on a programmable switch pipeline, however, requires finding a pipeline layout with minimal depth. Current pipeline targets, such as RMT [3], rarely have more than 32 match-action stages; a program layout which exceeds its switch target's depth forces the switch to recirculate packets, dropping throughput. Even when a program layout can fit into its target pipeline, there are good reasons for minimizing its depth. Longer pipelines have higher latency and must power more stages, resulting in high power consumption. Moreover, since most pipeline targets either disallow or strictly constrain stages from sharing memory, an unnecessarily long pipeline layout can lead to underutilization of each stage's resources, increasing its effective demand for costly SRAM and TCAM.

A critical limitation on the ability of compilers to generate compact datapath layouts is the inter-table data and control hazards, or **dependencies**, in routing programs. Under current pipeline architectures, if a table, T_1, has an output read by a second table, T_2, T_2 must be placed after T_1 in the datapath layout. Modern routing programs, reliant on complex systems of small, interconnected tables to perform computation, are burdened with cumbersome webs of dependencies. switch.p4 has 244 dependencies connecting its 163 tables!

To achieve compact pipeline layout, a pipeline architecture must allow parallel and even out of order execution of tables linked by dependencies, eliminating all constraints on pipeline layout except hardware constraints. Dependency resolution, however, raises the following challenges:

(1) **Correct parallel table execution.** Parallel execution of tables linked by dependencies can introduce race conditions [21] which must be handled correctly.
(2) **Efficient parallel table execution.** Switching ASICs must meet demanding throughput, latency, and power requirements. Any ASIC which permits parallelism must not compromise these requirements.

No public switching ASIC architecture resolves routing program dependencies. While work on speculative execution

SOSR '19, April 3–4, 2019, San Jose, CA, USA

ACM ISBN 978-1-4503-6710-3/19/04…$15.00
https://doi.org/10.1145/3314148.3314348

in CPU/GPU based systems [8, 15, 17] has yielded insights into handling race conditions in parallelism, these insights cannot be directly applied to switching ASICs.

In this paper, we introduce **precedence**, an extension of the RMT architecture which **novelly** allows tables linked by both data and control dependencies to be executed in parallel and even out-of-order by speculative execution. Precedence's underlying idea is not new [20], but we are the first to introduce it to the design of switching ASICs. Precendence can resove nearly **70%** of switch.p4's dependencies, reduces DC.p4 and switch.p4's pipeline depths by **40%** and **48%**, and only requires a modest increase in silicon area.

2 PROBLEM STATEMENT

To allow better parallelism of logical tables, our objective is to resolve as many dependencies between logical programming tables as possible. We focus on extending the RMT [4] pipeline to resolve dependencies between logical tables in P4 [2] programs, but we emphasize that our approach is generic and could be applied to other logical pipeline descriptions and hardware layouts.

2.1 Pipeline Dependency Model

We model the graph of dependencies between logical routing tables with Jose et al.'s [13] notion of a Table Dependency Graph (TDG). A TDG is a directed, acyclic graph (DAG) representing a pipeline's logical tables as vertices and the dependency between them as directed edges.

The edges in a TDG can be grouped into four classes, corresponding to the four types of data and control dependency. [13, 20] Each class has different characteristics and requires a different resolution. These classes are:

- *Read-After-Write (RAW) data dependency:* Table 1 writes a field read by Table 2.
- *Write-After-Write (WAW) data dependency:* Table 1 writes a field subsequently written by Table 2.
- *Write-After-Read (WAR) data dependency:* Table 1 reads a field subsequently written by Table 2.
- *Control dependency:* Table 1 determines whether program control passes to Table 2 or not.

As an example, Figure 1 gives a TDG for the L2 L3 Simple MTag benchmark program implemented by Jose et al. Each table is depicted as a box, each control dependency as dashed edge, each RAW data dependencies as a red solid edge and the lone WAW dependency as a blue solid edge. Since there may be multiple types of dependencies between two tables, a TDG may be a multigraph.

2.2 Switch Architecture

We present precedence as an extension of the RMT switch architecture: - a real, high-performance, programmable switch

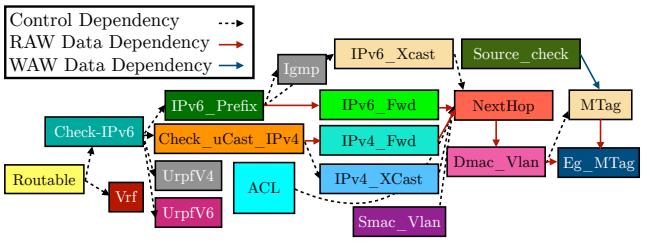


Figure 1: The TDG of the L2 L3 Simple MTag benchmark program implemented by Jose et al.

ASIC. Like traditional fixed-table ASICs, RMT processes packets with a linear pipeline of match action stages. Each stage contains a local SRAM and TCAM memory cluster to store custom match tables, and an action engine to carry out custom actions composed of a series of primitives.

3 PRECEDENCE

Precedence is a mechanism for resolving control and data dependencies between pipeline tables, allowing the tables to be placed on the same stage. It assigns every match-action rule a weighting, its precedence, which can either be a constant or the value of a metadata field. If multiple tables in a stage execute actions which store a value in the same metadata field, the conflict is resolved by storing the value of the action with the highest precedence. A rule's precedence field is treated similarly to any other table metadata field, and can be managed by runtime software.

Example: Consider the tables `Source_Check` (dark-green) and `MTag` (tan) shown in Figure 1. `Source_Check` always overwrites `MTag` if and only if it has an action to execute. In theory, the two tables could be placed in one stage by merging them into a single table, but in practice this is undesirable because the combined table requires a rule for every pair of rules in `Source_Check` and `MTag`; a combinatorial explosion of rules which consumes memory needlessly. Precedence allows both tables to share a stage by assigning `Source_Check`'s writes precedence 1 and `MTag`'s writes precedence 2. When both tables write, `MTag`'s write will take priority.

3.1 Architectural Model

Precedence is implemented as an extension of the RMT architecture by placing a new hardware component, the action output selector, between the action units' output and the outgoing metadata bus (Figure 2). The action output selector reads each action unit's output write and that output's precedence, which can either be sourced from a constant stored in the Very Long Instruction Word (VLIW) Memory, **or** a metadata field forwarded by the action input selector, **or** the output of another action unit. If multiple action units

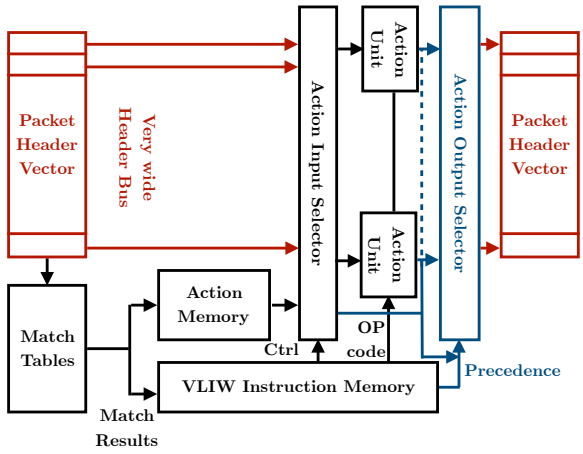


Figure 2: The RMT architectural model extended by precedence. New architecture added by the extension is colored in dark blue.

store their output in the same field the action output selector forwards the output with the highest precedence.

An action output selector for an n action unit RMT stage is constructed with a crossbar of n MUXes, each accepting each action unit's output as an input and sending their output to the outgoing metadata bus. Each MUX's SELECT is wired to a n-way comparator which can read an action unit's precedence from the output of another action unit, the action input selector, or the VLIW Instruction Memory.

3.2 Dependency Resolution

We now show how precedence can resolve the control and dataflow dependencies described in Section 2.

WAW data dependency resolution: WAW-dependencies, such as the `IPv4_UCast_LPM` example given above, can be resolved by giving each rule in the overwritten table a precedence of 1 and each rule in the overwriting table a precedence of 2. When the two tables are placed on the same stage, the overwriting table will overwrite the overwritten table.

Space Complexity: Any dependency between two tables can be naively resolved by merging the tables. In the worst case, merging two tables generates a rule for every pair of rules in the tables. If the two tables are denoted T_1 and T_2, and their rule number $|T_1|$ and $|T_2|$, then table merge generates $|T_1| \cdot |T_2|$ rules. By comparison, precedence resolves WAW-dependencies with no additional cost beyond the action output select, and thus only needs $|T_1| + |T_2|$ rules.

WAR data dependency resolution: Two tables linked by a WAR data dependency can be executed in a single stage.

RAW data dependency resolution: One might naively think that RAW data dependencies are intractable without merging. If table T_2 reads table T_1's output, one might reason, then T_1's output must be known before executing T_2. Consider, however, the case that T_1 performs a computation with a small number of outputs, like modulus or a conditional. Copies of T_2 for each possible outcome of t_1 could be executed speculatively in parallel, and precedence subsequently used to read T_1's result and select the execution of T_2 to store.

Example: Consider a data center routing protocol on an end-of-row switch indirectly connected to 3 core switches. To avoid directing all outgoing flows to a single core switch, the router computes mod 3 of each inbound packet's source IP and uses the result to choose between these three paths. This protocol could be implemented by two tables below:

```
t1: rnd_val <- src_IP % 3;
t2: next_hop <- route_tbl[dst_IP, rnd_val];
```

These tables can be placed in one stage using precedence by: **(a)** modifying T_1 to set three one bit flags `rnd_val_is_0`, `rnd_val_is_1`, and `rnd_val_is_2` according to `rnd_val`'s value, and **(b)** dividing T_2 into three tables which compute `route_tbl[dst_IP, 0]`, `route_tbl[dst_IP, 1]` and `route_tbl[dst_IP, 2]`, where `route_tbl[dst_IP, i]`'s output's precedence is `rnd_ele_is_i`. At runtime, T_2 is speculatively executed for all three possible values of `rnd_val` in parallel, and subsequently the action output selector looks at which `rnd_ele_is_i` flag is set to pick the correct version of T_2's output to store.

Space Complexity: To resolve a RAW-data dependency between two tables T_1 and T_2, precedence requires worst case $dom(T_1 \textit{ outputs read by } T_2) \cdot |T_2|$ rules, since one copy of T_2 needs to be made for every possible operand vector it could read from T_1. When $dom(T_1 \textit{ outputs read by } T_2)$ comparable to $|T_2|$, precedence's rule number is comparable to table merge (and requires many more action units). When $dom(T_1 \textit{ outputs read by } T_2)$ is a small constant, however, precedence only requires $O(|T_1| + |T_2|)$ rules.

In the previous example, merging the two tables in the code snippet above produces a single table with $|$`src_IP`$| \cdot |$`dst_IP`$|$ rules, which can be very large if $|$`src_IP`$|$ and $|$`dst_IP`$|$ are substantial. Resolving this dependency by precedence, however, only requires $|$`src_IP`$| + 3|$`dst_IP`$|$ rules.

Control dependency resolution: To understand how precedence can resolve a control dependency, consider the pipeline shown in Fig 3. This pipeline has no dataflow dependencies, and each arrow indicates a control flow dependency. For example, tables T_1, T_2 and T_3 form a fully generic control flow dependency: after executing T_1, program control is either passed to T_2 or T_3. Despite needing to know T_1's output to determine whether to execute T_2 or T_3, all three tables can be placed in the same stage by speculatively executing T_2 and T_3 and using precedence to select which output to store.

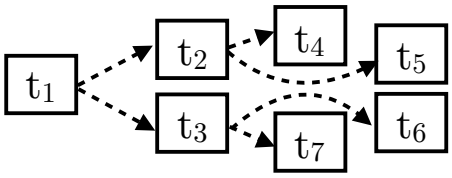


Figure 3: A pipeline with only control dependencies.

First, the control flow dependency can be converted into a data flow dependency by predication, replacing each of T_1's `jump(T2)` and `jump(T3)` actions with writes to the boolean predicate `is_jump_T2` and `is_jump_T3`. Next, T_2 and T_3's rules' precedence are set to their respective predicates. Finally, T_2 is instructed to write the special value `not_written` to any variable neither it nor T_1 originally wrote but T_3 did, and vice versa. At runtime, the action output selector can choose between T_2's and T_3's outputs because only one of `is_jump_T2` and `is_jump_T3` is set.

Space complexity: Precedence requires no additional rules to resolve control dependencies.

3.3 Combined Dependency Resolution

Not only can precedence resolve WAW, control flow, and certain RAW dependencies individually, it can also resolve any combination of these dependencies:

- To resolve a combined WAW and RAW dependency between two tables, T_1 and T_2, the precedence of T_1's rules is set to 0 so that its output will be overwritten by the selected speculatively executed copy of T_2.
- To resolve a combined WAW and control flow dependency between the parent table T_1 and the child tables T_2 and T_3, the parent table's rules are assigned precedence $\frac{1}{2}$, so that its output will overwrite its unchosen speculatively executed child and be overwritten by its chosen speculatively executed child.
- To resolve a combined RAW and control flow dependency between the parent table T_1 and the child tables T_2 and T_3, a copy of T_2 is made for each output of T_1 it could read, and similarly for T_3. Each predicate specifies both a table and an output of T_1.
- To resolve all three dependencies, proceed as the RAW & control flow case, but set T_1's precedence to $\frac{1}{2}$.

No change is required to resolve a WAR dependency in combination with any other set of dependencies. (To simply implementation, precedence values are scaled to be integers.)

3.4 Dependency Chain Resolution

Now that we have analyzed single dependencies, we turn to dependency chains. Consider a chain of WAW data dependencies $T_1 \rightarrow T_2 \rightarrow \ldots T_n$. Every table in such a chain can be executed in the same stage by assigning the i-th table's rules precedence i, so that T_i overwrites all tables before it in the chain and is overwritten by all tables after it.

Unfortunately, however, precedence cannot resolve chains of RAW data dependencies or control dependencies. Consider, for example, the pipeline TDG of control-dependencies shown in Figure 3. One might think that this pipeline could be executed in a single stage as we showed T_1, T_2 and T_3 could be previously by replacing T_2's jumps with the predicates `is_jump_T4` and `is_jump_T5`, and so forth. However, the value of `is_jump_T4` is not determined until the action output selector has processed T_2 and T_3's outputs; until then it could either be `true`, `false` or `not_written`. This means that T_4's precedence will not be known until the action output selector has chosen between the speculative executions of T_2 and T_3, so T_4 must be left to the next stage. In general, precedence cannot place tables in a 2 or more link RAW data/control dependency chain in the same stage.

3.5 Out-of-Order Execution

Thus far, precedence has only been used to place tables linked by a dependency in the same stage. Precedence can also be extended to enable out-of-order execution, where a table can be executed at any stage prior to a table it is dependent on.

If a speculatively executed table is placed in a stage prior to the table it depends on, its output is written to a set of temporary variables on the metadata bus. Then, in the stage containing the table it depends on, a small one rule table is added which reads each temporary variable and writes it, with its corresponding precedence, to the appropriate field. While this out-of-order execution strategy does take up an extra action unit for each out-of-order speculatively executed table, its memory overhead is minimal.

4 EVALUATIONS

We now evaluate precedence against a set of benchmark programs. We give the experimental setup (Section 4.1), and then measure the percentage of dependencies precedence resolves in the benchmark programs (Section 4.2). Next, we analyze its impact on the compiled depth of these programs (Section 4.3) and finally its hardware cost (Section 4.4).

4.1 Experimental Setup

Benchmark Programs. We benchmark on: **(1)** two real world, open source P4 programs, DC.p4 and switch.p4 and **(2)** four P4 programs used as benchmarks by Jose et al., L2 L3 Simple, L2 L3 Complex, L2 L3 Simple MTag and L3 DC. Table 1 gives key program attributes.

Target Switch Architecture. Programs are compiled to a precedence enabled RMT pipeline. The numeric values for this pipeline's attributes are chosen following Bosshart et al.'s recommendations in [4]. In particular, each match stage contains 106 SRAM blocks of 1K entries × 112b, 16 TCAM blocks of 2K entries × 40b, and a separate 640b crossbar for SRAM and TCAM, capable of querying 8 tables at once.

Program	Table Number	Maximum Table Size	Longest Dep. Chain
switch.p4	163	«1 stage	12
DC.p4	43	«1 stage	10
L2L3Simple	16	10 stages	6
L2L3Complex	25	4 stages	11
L2L3SimpleMTag	19	10 stages	8
L3DC	11	<1 stage	8

Table 1: Benchmark Program Table Number, Maximum Table Size, and Longest Dependency Chain.

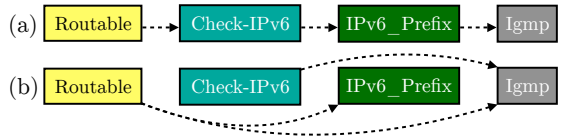


Figure 4: A control dependency chain in L2 L3 Simple MTag (a) without and (b) with precedence.

Compiler. The benchmark programs are compiled to the precedence enabled RMT pipeline using Jose et al.'s [13] compiler, after modifying it for precedence in two ways. **First**, all WAW-data dependency constrains are removed from each program's TDG. **Second**, for each chain of control dependencies in a program, each table's dependency on the next table in the chain is removed, and new dependencies between it and every other subsequent table in the chain added. For example, Figure 4 shows a chain of control dependencies from L2L3SimpleMTag without (Figure 4 (a)) and (Figure 4 (b)) with precedence. In (b), the dependency between Routable and Check_IPv6 (the next table in the chain) is removed and dependencies between IPv6_Prefix and Igmp (subsequent tables in the chain) added. All RAW data dependencies are assumed intractable unless otherwise stated.

4.2 Resolvable Dependency Frequency

First, we examine the percentage of dependencies in real-world routing programs that precedence can resolve. The number of RAW, WAW and control dependencies in each program are shown in Table 2. The programs are dominated by control and RAW dependencies. Precedence can reliably resolve every dependency except for RAW data dependencies. Figure 5 lists the number of resolvable dependencies in each program. Precedence can resolve a substantial minority of dependencies in all programs, and a majority in half. The more complex a program's control structure, the better precedence performs.

4.3 Pipeline Stage Reduction

Each benchmark program's pipeline depth with and without precedence is given in Figure 6. Precedence reduces the depth of DC.p4, switch.p4 and L3.DC by 40%, 48%, and 42%, while achieving more modest depth reductions on L2 L3 Simple, L2 L3 Simple MTg, and L2 L3 Complex (5%, 5% and 10%). Note

Program	RAW Deps.	WAW Deps.	Control Deps.
switch.p4	83	27	134
DC.p4	33	1	12
L2L3Simple	4	0	22
L2L3Complex	25	1	16
L2L3SimpleMTag	6	1	16
L3DC	7	3	1

Table 2: Benchmark Program Dependencies.

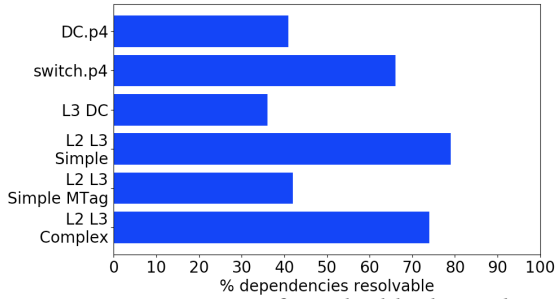


Figure 5: Percentages of resolvable dependencies.

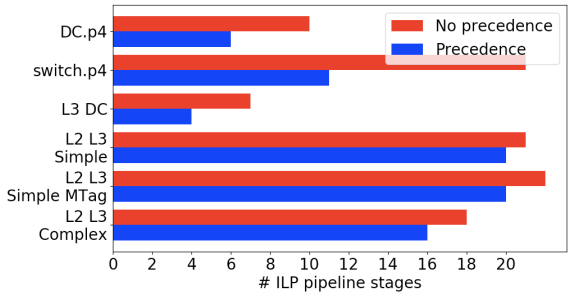


Figure 6: Layout depth decrease with precedence.

that switch.p4 requires double the match keys per stage for full depth reduction.

Precedence's performance difference between these two sets of programs is caused by their different structure. Figure 7 shows the layouts of DC.p4 and L2 L3 Simple without (left) and with (right) precedence. Each vertical column in Figure 7 represents a stage's memory, and each colored place in a column represent a table whose height represents the number of entries it contains. Two blocks with the same color on different stages indicate a table spanning multiple stages. Uncolored space indicates unused memory.

DC.p4 consists of 43 small tables joined by a complex web of dependencies. These dependencies limit table placement, resulting in significant unused space (Figure 7, DC.p4 (left)). By removing them, precedence recovers the space, achieving significant depth reduction (Figure 7, DC.p4 (right)). L2 L3 Simple's layout without precedence (Figure 7, L2 L3 Simple (left)), however, contains relatively very little unused space. Its depth is dominated by the size of its tables - in particular, two 160,000 entry tables (dark blue and grey) dominate 16 out of its 21 initial stages! Precedence's effects are more modest here because dependencies are not the main driver

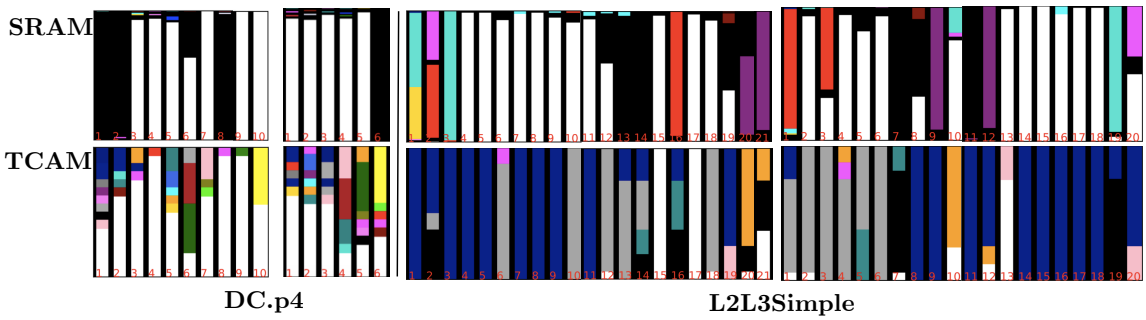


Figure 7: DC.p4 and L2L3Simple's layouts when compiled to a pipeline without (left) and with (right) precedence.

of pipeline depth. Notably, when the size of L2 L3 Simple's tables are halved, depth reduction increases from 5% to 20%. We argue, however, that future routing programs will look more like DC.p4 than L2 L3 Simple, using large numbers of (necessarily small) tables rather than a few monolithic tables to compute. Notably, precedence performs best on DC.p4 and switch.p4, the two real world programs.

DC.p4 also demonstrates the power of RAW dependencies resolution. DC.p4 contains two critical RAW dependencies: one on a table with four rules, and one on a table with nine. Even if no other control or WAW dependencies are removed, removing these RAW dependencies decreases DC.p4's layout to 6 stages. When tables are small, replication provides a practical way to remove RAW bottlenecks.

4.4 Hardware Cost

Finally, we evaluate the hardware cost of the output selector to show these benefits can be obtained feasibly. The output selector was modelled as a $224 \times 224 \times 19$b crossbar, connecting each of RMT's 224 ALU's outputs to the 224 fields on its 4Kb bus. The crossbar's wiring was synthesized with the ORION 3.0 [14] power and area simulator; its comparator logic is small and is neglected. ORION 3.0 synthesis gives a crossbar area of 0.95mm^2 using 54 nm process technology.

RMT stage logic (excluding memory) is estimated in [6] as 1.243mm^2 using 16 nm process technology. Scaling the synthesized area by 16/45 to match, the output selector adds 0.34mm^2 to each stage's logic, a 21% increase. Stage logic, however, is only a small fraction of RMT's chip area: the action engine is 7.4% of its area and its match crossbar is less. The output selector thus only increases RMT's area modestly.

5 RELATED WORK

Physical Layout Design. Many techniques have been proposed to optimize the datapath layout of a routing program. Jose et al. [13] and Dai et al. [7] give compilers to efficiently map a logical programming application into pipelined stages. Alternatively, Hardware Design Languages such as [18] and [24] allow programs to directly optimize datapath layout. Unlike these approaches, precedence's optimizations are primary hardware based and focus on dataplane architecture.

Switching ASIC Architecture Design. Several variants on RMT's switching ASIC architecture have been proposed. dRMT [6] disaggregates RMT's memory, placing it in a memory pool accessible over a centralized crossbar. dRMT removes most dependency constraints on table placement, but not on access, and requires a monolithic crossbar between its processors and memory. Banzai [22] equips each stage with more complex logic, potentially allowing more compact layout, but does not directly address dependency constrains. Many commercial switches, such as Cavium XPliant [5], Intel FlexPipe [12] and Broadcom Trident 3 [1], also use an augmented match-action pipeline architecture. While their precise architectures are not available, their capabilities are. Broadcom silicon, for example, can address WAW, WAR and control dependencies by combining the notion of logical tables with a concept similar to precedence.

Speculative Execution. Outside the scope of programmable switches, speculative execution [9, 16, 19, 21] is a well-studied method for achieving different granularities of parallelism. Further, the rise of general-purpose graph processing units (GPGPU) has led to the study of speculative parallelism in GPU-based systems. Diamos et al. [8] adopts thread-level speculation to automatically convert a sequential program into parallel execution blocks. Liu et al. [15] explores the benefits of using GPU to achieve software value prediction.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of Broadcom, Inc. to this paper in providing a conceptual and practical basis for precedence-based data-path and control-path execution in a (non-public) programmable switch ASIC pipeline, and further demonstrating how dependency resolution can be defined in a high-level network programming language. While this paper demonstrates how precedence can substantially augment the processing efficiency of a public switching architecture such as RMT, the authors acknowledge that commercial, non-public programmable switch ASIC architectures currently available or in development may leverage similar techniques to provide such advantages over conventional match-action pipelines.

REFERENCES

[1] Broadcom Trident 3. [n. d.]. XPliant Ethernet Switch Product Family. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series/. Accessed: 2018-11-15.

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[3] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 99–110.

[4] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 99–110. https://doi.org/10.1145/2486001.2486011

[5] Cavium. [n. d.]. XPliant Ethernet Switch Product Family. https://www.cavium.com/xpliant-ethernet-switch-product-family.html. Accessed: 2018-11-15.

[6] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, et al. 2017. drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 1–14.

[7] Jinquan Dai, Bo Huang, Long Li, and Luddy Harrison. 2005. Automatically Partitioning Packet Processing Applications for Pipelined Architectures. *SIGPLAN Not.* 40, 6 (June 2005), 237–248. https://doi.org/10.1145/1064978.1065039

[8] G. Diamos and S. Yalamanchili. 2010. Speculative execution on multi-GPU systems. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. 1–12. https://doi.org/10.1109/IPDPS.2010.5470427

[9] Lance Hammond, Mark Willey, and Kunle Olukotun. 1998. Data speculation support for a chip multiprocessor. *ACM SIGOPS Operating Systems Review* 32, 5 (1998), 58–69.

[10] David Hancock and Jacobus van der Merwe. 2016. HyPer4: Using P4 to Virtualize the Programmable Data Plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 35–49. https://doi.org/10.1145/2999572.2999607

[11] Barefoot Inc. 2019. switch.p4. https://github.com/p4lang/switch/blob/master/p4src/switch.p4

[12] Intel. [n. d.]. Intel Ethernet Switch Silicon. https://www.intel.com/content/www/us/en/products/network-io/ethernet/switches.html. Accessed: 2018-11-15.

[13] Lavanya Jose, Lisa Yan, George Varghese, and Nick McKeown. 2015. Compiling Packet Programs to Reconfigurable Switches. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA, 103–115. http://dl.acm.org/citation.cfm?id=2789770.2789778

[14] Andrew B Kahng, Bill Lin, and Siddhartha Nath. 2012. Explicit modeling of control and data for improved NoC router estimation. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. IEEE, 392–397.

[15] Shaoshan Liu, Christine Eisenbeis, and Jean-Luc Gaudiot. 2011. Value Prediction and Speculative Execution on GPU. *International Journal of Parallel Programming* 39, 5 (Oct. 2011), 533–552. https://doi.org/10.1007/s10766-010-0155-0

[16] Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, and Roger A. Bringmann. 1992. Effective Compiler Support for Predicated Execution Using the Hyperblock. In *Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO 25)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 45–54. http://dl.acm.org/citation.cfm?id=144953.144998

[17] J. Menon, M. de Kruijf, and K. Sankaralingam. 2012. iGPU: Exception support and speculative execution on GPUs. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. 72–83. https://doi.org/10.1109/ISCA.2012.6237007

[18] Rishiyur Nikhil. 2004. Bluespec System Verilog: efficient, correct RTL from high level specifications. In *Formal Methods and Models for Co-Design, 2004. MEMOCODE'04. Proceedings. Second ACM and IEEE International Conference on*. IEEE, 69–70.

[19] Jeffrey T Oplinger, David L Heine, and Monica S Lam. 1999. In search of speculative thread-level parallelism. In *Parallel Architectures and Compilation Techniques, 1999. Proceedings. 1999 International Conference on*. IEEE, 303–313.

[20] David A. Patterson and John L. Hennessy. 1990. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[21] B Ramakrishna Rau and Joseph A Fisher. 1993. Instruction-level parallel processing: history, overview, and perspective. In *Instruction-Level Parallelism*. Springer, 9–50.

[22] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 15–28.

[23] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. 2015. Dc. p4: Programming the forwarding plane of a data-center switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2.

[24] Donald Thomas and Philip Moorby. 2008. *The Verilog® Hardware Description Language*. Springer Science & Business Media.

TRIDENT: Toward a Unified SDN Programming Framework with Automatic Updates

Kai Gao
Tsinghua

Taishi Nojima
Yale

Y. Richard Yang
Yale/Tongji

ABSTRACT

Software-defined networking (SDN) and network functions (NF) are two essential technologies that need to work together to achieve the goal of highly programmable networking. Unified SDN programming, which integrates states of network functions into SDN control plane programming, brings these two technologies together. In this paper, we conduct the first systematic study of unified SDN programming. We first show that integrating asynchronous, continuously changing states of network functions into SDN can introduce basic complexities. We then present TRIDENT, a novel, unified SDN programming framework that introduces programming primitives including stream attributes, route algebra and live variables to remove these complexities. We demonstrate the expressiveness of TRIDENT using realistic use cases and conduct an extensive evaluation of its efficiency.

CCS CONCEPTS

• **Networks** → **Programming interfaces**; **Network management**; *Middle boxes / network appliances*;

KEYWORDS

SDN, Network functions, Network programming, Stream attributes, Live variables, Route Algebra

ACM Reference Format:
Kai Gao, Taishi Nojima, and Y. Richard Yang. 2018. TRIDENT: Toward a Unified SDN Programming Framework with Automatic Updates. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3230543.3230562

1 INTRODUCTION

Bringing together software-defined networking (SDN) and network functions (NF) is an essential step toward highly programmable networking. In particular, SDN introduces the ability of logically centralized, global network routing; network functions (*e.g.*, deep packet inspection, DPI), on the other hand, introduce the ability of network elements conducting general-purpose, programmable packet processing beyond the network layer. Only by combining SDN and network functions can one realize programmable, cross-layer networking. The importance of integrating SDN and network functions has motivated multiple studies (*e.g.*, [1–7]).

Unified SDN programming, which integrates information extracted by network functions, such as application layer HTTP headers, into SDN programming, is an approach to bringing SDN and network functions together. By exposing such information, which we refer to as *network function state information* or *network function state* for short, to SDN programming, unified SDN programming can bring many benefits such as adaptive, cross-layer SDN control.

Despite the benefits, realizing unified SDN programming, however, is non-trivial, due to 3 basic programming complexities that are not fully addressed before:

1) *How to naturally integrate network function state into SDN programming.* Current SDN programming frameworks (*e.g.*, [8–12]) make decisions on the headers of each individual packet. A network function state, on the other hand, may not appear in a single packet but span multiple packets, and hence need to be extracted by a network function using a finite state machine. For example, a DPI can extract the HTTP `URI` field [13] for an SDN program to better route traffic (*e.g.*, large files are routed differently). The extraction of the field, however, is a progress: the field can be *unknown* for a *non-deterministic* amount of time due to TCP three-way handshake and fragmentation. Existing SDN programming frameworks do not have a model to expose cross-packet, *asynchronous* state.
2) *How to flexibly construct consistent, correlated routes to utilize network function state.* One benefit of SDN is the ability to realize complex, advanced routing, such as traffic engineering routing and fast rerouting. Integrating network function states into SDN routing can exacerbate the complexity. For example, a network function may have route symmetry as a consistency requirement: both forward and return traffic go through it to update its finite state machine reliably. Computing advanced routing and at the same time enforcing consistency for network functions can then become a substantial programming complexity.

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5567-4/18/08…$15.00
https://doi.org/10.1145/3230543.3230562

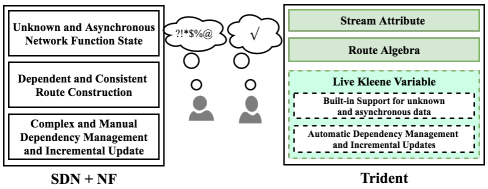


Figure 1: Complexities and TRIDENT Primitives.

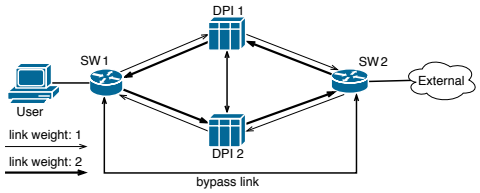


Figure 2: Example Network.

Although there are excellent previous studies in SDN routing (*e.g.*, [5–7, 14–19]), they are lacking in supporting efficient, flexible routing for unified SDN programming.

3) *How to handle dynamicity of unified SDN programming.* In the general case, the state of a network function can continuously change: a DPI updates the URI when a persistent HTTP connection requests the next content; a security network function updates the security level of an existing connection. When the state changes, the routes may also need to change. Thus, accurate and efficient dependency management is crucial in achieving correct, efficient utilization of network function states. Existing approaches use either manual identification of data dependencies (*e.g.*, [6, 20, 21]), which can be complex and error-prone, or simple automatic dependency tracking and complete recompilation (*e.g.*, [22, 23]), which can lead to unnecessary recomputation and large latency.

In this paper, we introduce TRIDENT, a novel SDN programming framework that addresses the preceding 3 complexities with 3 powerful, simple-to-use programming primitives. A mapping between complexities and solution primitives are shown in Figure 1. Specifically, the 3 primitives include:

1) *stream attribute* (§5), a simple abstraction for modeling cross-packet states extracted by network functions;
2) *route algebra* (§6), a simple yet powerful abstraction to flexibly express consistent, advanced routing;
3) *live Kleene variable* (§4), a unified data model and the foundation of TRIDENT, which enables built-in support for unknown and asynchronous data, as well as *semantic dependency tracking* and automatic incremental updates.

We implement ((§7) a prototype of TRIDENT and conduct an extensive evaluation (§8) to demonstrate that TRIDENT can be realized cleanly and efficiently.

2 MOTIVATION

We start with a simple example to illustrate how an SDN program may use network function state and the 3 complexities introduced in §1. The example uses a simplistic network shown in Figure 2, in which two DPIs are deployed. We use an algorithmic SDN programming model [10], due to its flexibility and simplicity. At a high level, the example SDN program routes traffic on the bypass link if the internal user accesses an HTTP URI that is not "sensitive"; otherwise, the traffic should be continuously monitored by a DPI.

C1: Naturally Integrating Network Function State into SDN. As simple as the objective of the SDN program is, existing high-level SDN programming frameworks (*e.g.*, [8–10, 24]) typically do not support routing using layer-7 information such as "`http_uri`", as such information is unlike packet header fields, which are contained in every packet. Instead, such information can be extracted only by a network function, potentially after reassembling multiple packets.

Despite the reassembling need, one might think that the issue would be trivial to solve: network functions and SDN controllers share a common data store, and the network functions update the data store. This leads to a simple SDN program as below, where "`http_uri`" is a variable that DPIs update for each HTTP flow in the data store:

```
1  if (pkt.http_uri === "sensitive")) {
2    // This branch can never be reached
3  } else {
4    // Bypass link
5  }
```

As simple as it is, the program does not achieve the objective. When the first packet (TCP SYN) of a "sensitive" HTTP flow enters the network, a DPI cannot decide whether the packet belongs to an HTTP flow with a "sensitive" URI. Thus, the result is *unknown*. Consider two approaches handling unknown. First, assume an unblocking (*i.e.*, asynchronous) design in that a DPI assigns an initial value null for such a case. Using standard logic, the controller will evaluate the condition of the if statement as false, and hence the packet should be handled by the else branch, resulting in that the packet uses the bypass link, which has no DPI at all. With no DPI to inspect and update, "`http_uri`" will remain null, and all packets will use the bypass link. Second, assume a blocking design, in that the SDN program blocks when reading "`http_uri`", if the value is not available. However, with no route returned, the whole system blocks, with no update on "`http_uri`", leading to a deadlock.

Summary: Cross-packet, unknown and asynchronous network function states can be complex to handle.

C2: Constructing Consistent, Correlated Routes to Utilize Network Function States. Assume that one can fix the preceding complexity, so that the initial packets of each HTTP flow from the user go through a DPI. Below is a potential code segment showing route computation:

```
1  if (pkt.http_uri === "sensitive")) {
2    // compute shortest path from src to dst, must use a DPI
3  } ...
```

A problem of the preceding route computation, however, is that it does not enforce routing consistency required by network functions. In particular, a network function may have symmetry as a consistent requirement, in that the return packets go through the same network function instance (*e.g.*, Pico Replication [25]). Assume directed link weights shown in Figure 2, the shortest path algorithm computes SW1→DPI1→SW2 for the forward (user to outside) packets, and SW2→DPI2→SW1 for the return (outside to user) packets, leading to inconsistency (different DPIs).

One might think that the inconsistency problem would be easy to fix: when setting SW1→DPI1→SW2 as the route for the forward packets, setting at the same time an inverse of it for the return packets. Although this works in a simple setting, in a practical setting where both a primary route and a backup route are installed to implement fast rerouting, the technique may not work: when SW1 switches the route of the forward packets from primary (SW1→DPI1→SW2) to backup (SW1→DPI2→SW2) when link SW1→DPI1 is down, there are no existing mechanisms to specify that the return packets need to automatically change route as well.

Summary: Although there are previous studies on the construction of correlated routes, for example, Genesis [16] introduces the ability to specify disjoint routes, a mechanism for systematic construction and enforcement of consistent routes supporting unified SDN programming is missing.

C3: Handling Dynamicity of Unified Programming. The preceding discussions on **C**1 and **C**2 already touch on the complexity of dynamicity: in **C**1, when a network function updates its state, the execution path of the SDN program may change; in **C**2, when a link fails, the backup routes will need to consistently replace the primary routes. In the general setting, other data can trigger updates as well.

Consider a revision of the SDN program shown below, where the program uses source IP to determine `user`, and each `user` has a customized `white_list`:

```
1  if (user.white_list.contains(http_uri)) {
2    ...
3  }
```

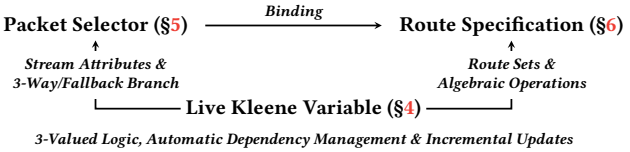
In this example, network function state (`http_uri`), configuration state (`white_list`), and network topology state can all lead to program outcome changes. Thus, identifying dependencies and ensuring the consistency between control plane state and outcomes can become extremely complex and error-prone (*e.g.*, [20, 21, 26–28]. Even with an event-driven system [6, 29], specifying the states and transitions can also be complex, because the number of states grows exponentially with the number of data.

3 OVERVIEW

With an understanding of the key complexities, we now introduce TRIDENT. We first introduce the programming primitives in TRIDENT that address these complexities. We then go over TRIDENT programming workflow using an example.

3.1 TRIDENT Programming Primitives

To understand not only how TRIDENT programming primitives address the complexities in §2 but also how they fit into general SDN programming, consider the "match-action" paradigm of SDN programming: each SDN program need to specify (1) the selection of packets (match), (2) the action for each selection, and (3) the binding between match and action. TRIDENT introduces (1) stream attributes to extend traditional packet selection to support network function states; (2) route algebra to specify consistent routing actions; and (3) live variables to achieve consistent binding between match and action despite dynamicity. The diagram below illustrates how TRIDENT extends SDN programming:



Stream Attribute. Traditional SDN programming uses headers in each packet to select packets, but as we identified in **C**1, network function states can be cross-packet. Hence, TRIDENT introduces stream attribute as an abstraction for cross-packet network function state. Hence, a TRIDENT program can use both packet headers and stream attributes to select packets.

Specifically, a stream attribute not only exposes a network function state but also specifies the group of packets that must be sent to the network function to compute the state. For example, to specify "`http_uri`" as a stream attribute, one must specify that it takes a complete TCP flow (TCP5TUPLE); on the other hand, a stream attribute such as "`is_endhost_infected`" may need all packets from/to the same endhost. Hence, a stream attribute exposes not only the output (state) but also the input of a network function, allowing TRIDENT to make consistent, efficient decisions.

One key complexity of exposing stream attributes, as we discussed in **C**1, is that their values can be unknown and continuously change. Hence, in TRIDENT, stream attributes can use only unknown-conforming operations: They can be either the *IS_UNKNOWN* operator `?` (examples in later sections), or unknown-enforced control structures including *3-way branch* and *fallback branch*. The examples below illustrate the two control structures and the left hand handles the example in **C**1: Now all possible values of "`http_uri`" are explicitly enumerated, but the program still has an intuitive synchronous programming structure. TRIDENT will automatically re-execute the program as asynchronous "`http_uri`" changes arrive (see below on live Kleene variables).

```
if (pkt.http_uri === "sensitive") {
  // The true branch
} else {
  // The false branch
} unknown {
  // The unknown branch
} // 3-way branch example
```

```
iff (pkt.is_analytics_job) {
  // The true branch
} else {
  // The false branch
  // and the unknown branch
} // fallback branch example
```

Route Algebra. TRIDENT route algebra introduces systematic route construction primitives to address **C**2. As we discussed in **C**2, a network function can have consistency requirements on how packets traverse it, but such requirements are not fully handled in existing systems. Hence, TRIDENT route algebra introduces a grammar called *network function indicator*, which selects routes satisfying traversal requirements. Consider the example code below:

```
1  val X = SimplePath(G, H :-: DPI :-: ISP)
2  ...
3  iff (pkt.http_uri === "sensitive") {
4      val x = any((X where { capacity >= 100 Gbps }) >> X)
5      val y = inv(x)
```

The expression `H :-: DPI :-: ISP` is a network function indicator. It means that among all simple paths (*i.e.*, a link is used at most once) in graph `G`, the result `X` only contains those connecting any host (`H`) and any external Internet service provider (`ISP`), and passing exactly one `DPI`.

TRIDENT route algebra also introduces generic, algebraic route constructions for flexible SDN routing. In the example above, L4 constructs a route `x` using 3 algebraic operators: (1) the selection operator (`where`) selects a subset of `X` only those with a capacity greater than or equal to 100 Gbps; (2) the preference operator (`>>`), which is key to implement backup routing, specifies that routes selected in step (1) are preferred than those only in the general set `X`; (3) the arbitrary selection operator (`any`) then selects one from routes given to it. Consider the example network in §2 where all links are 100 Gbps except SW1→DPI2, which is only 50 Gbps. Then `x` will record that SW1→DPI1→SW2 is the chosen route, if available, and SW1→DPI2→SW2 is the backup. L5 uses the inversion operator (`inv`) to specify that `y` is the inverse of `x`. It is important to note that route algebra not only computes routes but also records how routes are computed, to systematically address **C**2, as discussed next.

Live Kleene Variable. Both stream attribute and route algebra are high-level abstractions, and a reader may already wonder how they can be realized, in particular due to dynamicity. In TRIDENT, it is through *live Kleene variables* that stream attributes and route algebra are realized. Further, live Kleene variables go beyond stream attributes and routes to provide a generic mechanism handling dynamicity. Hence live Kleene variables are foundation to address **C1** to **C3**.

The basic function of a live Kleene variable is simple: TRIDENT keeps track of dependencies for it and automatically updates it when its dependency changes.

We use an example including both a stream attribute and a route algebra operation to illustrate the foundational role of live Kleene variables. Figure 3 shows the example. Assume that when a packet with source IP 10.0.1.5 arrives, the network function providing `is_endhost_infected` does not know whether 10.0.1.5 is infected. From the programmer's point of view, the fallback branch of the program is executed: a route `r_1 + r_2`, is constructed using the route algebra concatenation operator `+` from route `r_1` and route `r_2`, and then the TRIDENT built-in function `bind` specifies that the packet be forwarded using the constructed route. Under the hood in TRIDENT, both routes and bindings are live Kleene variables and hence TRIDENT keeps track of dependencies for them, resulting in a tracking record shown at Figure 3(a). Now assume `r_1` is changed to a new value (*i.e.*, a different route), TRIDENT will automatically update the computation, generating a new concatenated route to forward packets for 10.0.1.5, as shown Figure 3(b). Instead of route changes, consider a stream attribute change: the network function updates that 10.0.1.5 is infected. Then TRIDENT re-executes the program, blocking all packets of 10.0.1.5. With no packets for 10.0.1.5 and if the only input that can trigger the network function to update `is_endhost_infected` is packets, then 10.0.1.5 will be blocked indefinitely. On the other hand, the network function can have an administrative interface to reset `is_endhost_infected`, for example, after an operator disinfects the endhost. The reset will then automatically trigger TRIDENT re-execution.

```
1 iff (pkt.is_endhost_infected) {
2   drop(pkt)
3 } else {
4   bind(pkt, r_1 + r_2)
5 }
```

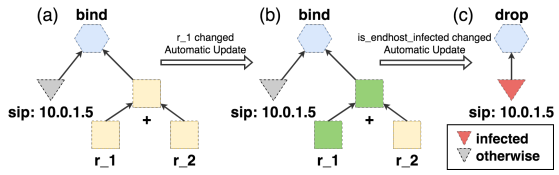


Figure 3: Packet Selector and Route as Live Variables.

3.2 TRIDENT Programming Workflow

The preceding TRIDENT programming primitives are generic abstractions and hence may be realizable in multiple languages or systems. Below we illustrate TRIDENT programming using a domain-specific language embedded in a Scala-like language, to utilize its language features to improve code readability. We also choose an algorithmic SDN programming setting (*e.g.*, Maple [10] or SNAP [30]).

A complete system integrating SDN and network functions can have additional complexities such as placement and life cycle management of network functions. To focus on TRIDENT features, we assume that the network functions are already deployed. Further, complex network functions can have complex behaviors such as reading the states of (remote) SDN controllers or other network functions. Although TRIDENT can be extended to handle such additional cases, we focus on independent, write-only network functions, where by an independent, write-only network function, we mean one that does not read the state of either the controller or another network function.

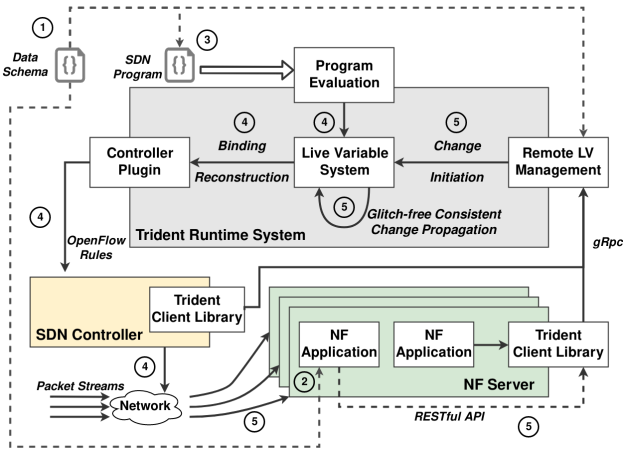


Figure 4: TRIDENT Programming Workflow.

With the preceding clarification, programming using TRIDENT is simple, with few steps, as shown in Figure 4.

Step 1: Declare Stream Attributes (Programmer). Integrating stream attributes is a key component for unified SDN programming, and in TRIDENT, using stream attributes is simple. As we discussed in §3.1, a stream attribute has both output (state) and input. Hence, a declaration of `http_uri`, as shown below, specifies that the output is a string with name "`http_uri`", and that the responsible network function needs packets of a complete TCP flow:

```
// trident.example.DPI - object DPI
val http_uri = StreamAttribute[String]("http_uri", TCP5TUPLE)
```

Step 2: Implement Stream Attribute in Network Function (Programmer). A network function providing a stream attribute needs to be modified to continuously publish the state to TRIDENT runtime. TRIDENT provides a simple client library to simplify the publishing process, and as we will show in §8, this typically takes only a few lines of code.

Step 3: Write Unified SDN Program (Programmer). Writing a unified SDN program using stream attributes is straightforward, and below is an example program:

```
1  // trident.example.DemoProgram
2  object DemoProgram extends SDNProgram {
3    import trident.example.DPI.http_uri
4
5    val capacity = NetworkAttribute[Link, BandwidthUnit]("capacity")
6    val label = NetworkAttribute[Vertex, String]("label")
7
8    val DPI = Waypoint(V where { label === "dpi" }) expose http_uri
9    val H = Waypoint(V where { label === "host" })
10   val ISP = Waypoint(V where { label === "isp" })
11   val X = SimplePath(G, H :-: DPI :-: ISP)
12   val Y = SimplePath(G, H :-: ISP)
13
14   override def onPacket(pkt: Packet) = program {
15     iff (pkt.http_uri === "sensitive") {
16       val x = any((Y where { capacity >= 100 Gbps }) >> Y)
17       bind(pkt, x)
18       bind(inv(pkt), inv(x))
19     } else {
20       bind(pkt, any(X))
21       bind(inv(pkt), any(X))
22     }
23   }
24 }
```

With an understanding of the primitives and logically centralized SDN programming, the program is quite straightforward: the `onPacket` function (L4) defines the behavior for each packet; L15 uses a stream attribute to select packets; L8-L12 use route algebra to construct static routes; L16 uses route algebra to construct consistent, dynamic routes; L17-L18 bind constructed routes for both traffic directions.

Step 4: Deploy Program (TRIDENT Runtime). After submitting the program to TRIDENT, the TRIDENT runtime will execute the program. The value of the `pkt` can be either reactively obtained from `packet-miss` messages or proactively constructed using *symbolic execution*. After the execution, the new bindings are added to the live variable system, and are further *translated* into datapath OpenFlow rules. A key step of the translation is to generate the "match" from the packet selector, which we will discuss in §5.1. The datapath deployment can be handled by many systems (*e.g.*, [31–33]).

Step 5: Automatic, Asynchronous Updates (TRIDENT Runtime). As packet streams enter and traverse the network according to the SDN program, network functions or switch agents update the values of certain stream attributes or network attributes, which eventually change the output of program. These asynchronous updates are sent to TRIDENT through the client library. TRIDENT runtime automatically captures the asynchronous data changes and checks whether the resulted bindings are still consistent with the program. Invalid routes and bindings are removed automatically, along with the corresponding datapath configurations. Meanwhile, the program is re-evaluated to compute new consistent bindings. Invalidation and recovery of the same stream are conducted as a single transaction to avoid inconsistency.

4 LIVE KLEENE VARIABLE

As introduced in §3, live Kleene variables (or simply *live variables*) are the foundation of the abstractions in TRIDENT. Now we formally specify the live Kleene variable in detail.

4.1 Live Variable System

Live Variable. A live variable x is specified as follows:

$$x : \langle type, value, deps, operator, operands \rangle, \tag{1}$$

where each property is defined as follows:

- *type* (T_x): the data type for x's value, which defines the valid operations and the domain (D_x) for x's value;
- *value* (v_x): x's value, which is either a valid data in the domain D_x, or *unknown*, *i.e.*, $v_x \in D_x \cup \{unknown\}$;
- *deps* ($\mathcal{D}_x$): a set of live variables on which x depends. Let $\prec$ denote the dependency relationship, *i.e.*, $\forall d \in \mathcal{D}_x$, $d \prec x$.
- *operator* (f_x) and *operands* ($\mathcal{Y}_x$): a function which defines how x's value is computed, and a sequence of live variables which are the input of the *operator* f_x. Let $y_{x,i}$ denote the i-th operand in $\mathcal{Y}_x$, then f_x has the following signature: $f_x : D_{y_{x,1}} \times \ldots, D_{y_{x,|y_x|}} \mapsto D_x$.

It is important to note that one can attach new properties to a live variable and enable functionalities such as efficient change propagation introduced in §7.3.

Consistency Guarantee. Let $\mathcal{X}$ denote the set of all live variables. We define two important consistency properties:

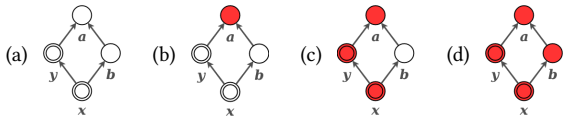
Definition 1 (Synchronized [34]). $\forall x, x' \in \mathcal{X}$ such that $x' \prec x$, x is *synchronized* with x' if and only if: *1)* $\forall d \in \mathcal{D}_x$, if $x' \prec d$, d is synchronized with x'; *2)* $\forall y \in \mathcal{Y}_x$, if $x' \prec y$, y is synchronized with x'; and *3)* $v_x = f_x(v_{y_{x,1}}, \ldots, v_{y_{x,|\mathcal{Y}_x|}})$.

Definition 2 (Glitch-free [34]). $\forall x \in \mathcal{X}$, x is *glitch free* if and only if $\nexists x', y, y' \in \mathcal{X}$ such that $x' \prec y \prec x$ and $x' \prec y' \prec x$, where *1)* y is synchronized with x', *2)* y' is not synchronized with x', and *3)* x is synchronized with both y and y'.

To achieve the two properties, TRIDENT manages the following procedures of each live variable $x \in \mathcal{X}$:

- *Dependency tracking*: to determine the dependency set $\mathcal{D}_x$, given an operator f_x and operands $\mathcal{Y}_x$.
- *Change propagation*: to determine whether v_x should be updated, and if so, compute the new value of x, given an operator f_x; operands $\mathcal{Y}_x$; a live variable $\kappa \prec x$ whose value has changed, which we call a *root cause*; and a subset of x's dependent live variables $\mathcal{D}_x(\kappa) \subseteq \mathcal{D}_x$ where $\forall y \in \mathcal{D}_x(\kappa)$, v_y is synchronized with the new v_κ.

The following example illustrates the dependency relationships and change propagation processes for four live variables, a, b, x and y, where x depends on y and b, and both y and b depend on a. In this example, a is a root cause, and x and y are exposed results. Live variables with different colors means that they are not synchronized. In this example, x and y in (a), (b) and (d) are all glitch-free.



4.2 Basic Live Variable

We now introduce three types of basic live variables, which are later extended to different programming primitives.

Simple Live Variable. A simple live variable x represents the result of an idempotent function and depends on all of its operands. When the value of a live variable $\kappa \prec x$ is updated, x's value must be recomputed in a glitch-free way, *i.e.*,:

- *Simple dependency tracking:* Given an operator f_x and operands $\mathcal{Y}_x$, $\mathcal{D}_x \leftarrow \mathcal{Y}_x$.
- *Simple change propagation (SCP):* Given an operator f_x, operands $\mathcal{Y}_x$, a root cause $\kappa \prec x$, and a set of dependent live variables $\mathcal{D}_x(\kappa)$, if and only if $\mathcal{D}_x(\kappa) = \mathcal{D}_x$, $v_x \leftarrow f_x(v_{y_{x,1}}, \ldots, v_{y_{x,|\mathcal{Y}_x|}})$.

Proposition 1. SCP guarantees glitch-free consistency.

PROOF. **Sketch** This can be proved by checking Definition 2 for x and its dependencies recursively. □

Remote Live Variable. A remote live variable x represents the most recent value of a remote data source, such as an internal state of a network function, and is specified as follows:

$$x : \langle type, value, deps \equiv \emptyset, operator \equiv \textbf{\textit{remote}}, operands \equiv \emptyset, source, uuid \rangle, \quad (2)$$

where *type*, *value*, *deps*, *operator* and *operands* are the same as in (1), and the new properties are specified as follows:

- *source*: a sequence S_x of values provided by a remote data source, where new values are continuously appended to S_x. The domain of each value v in S_x is specified by *type*, *i.e.*, $v \in D_x \cup \{unknown\}$;
- *uuid*: a universally unique identifier (id_x) for x *i.e.*, for remote live variables x and y, $id_x = id_y \Leftrightarrow x = y$.

A remote live variable x has a special operator ***remote***, which does not take any operand; x thus does not depend on any other live variables and is managed by the rules below:

- *Remote dependency tracking*: As specified in (2), $\mathcal{D}_x \leftarrow \emptyset$.
- *Remote change propagation*: The value of x is updated whenever there is a new value in S_x, and $v_x \leftarrow S_x \downarrow$, where $\downarrow$ means fetching the last value of a sequence.

Programmers can avoid working with the *source* directly by using predefined extensions of remote live variables, *e.g.*, stream attributes (§5.1) and network attributes (§6.1).

Snapshot Live Variable. In TRIDENT, the *operator* of a live variable can be a complex routing function. However, such a function can take a very long execution time, during which the live variable's value becomes unknown. To make the last known value always available, we introduce a snapshot live variable $\tilde{x}$, which is used to hold the last *known* value of another live variable y until y is synchronized; however, $\tilde{x}$ can have *unknown* value when the first known value of y is not yet available. $\tilde{x}$ is specified as follows:

$$\tilde{x} : \langle type, value, deps, operator \equiv :=, operands \rangle. \quad (3)$$

A snapshot live variable $\tilde{x}$ has a special operator called *snapshot assignment* (denoted as *:=*), and has exactly one operand y. $\tilde{x}$ must have the same type as y, *i.e.*, $T_{\tilde{x}} = T_y$ and is managed using the following rules:

- *Snapshot dependency tracking*: Given the operand y, if and only if y is synchronized, $\mathcal{D}_{\tilde{x}} \leftarrow \{y\}$; otherwise $\mathcal{D}_{\tilde{x}} \leftarrow \emptyset$.
- *Snapshot change propagation*: Given the operand y, if and only if y is synchronized, $v_{\tilde{x}} \leftarrow v_y$; otherwise, the value of $\tilde{x}$ does not change.

A snapshot live variable "cuts off" the dependencies conditionally to guarantee that the result is *always* glitch-free. In our examples, due to the limitation of Scala, the *snapshot assignment* is actually written as a function `snapshot`.

Example. Figure 5 includes a minimal abstract example, which involves a source S with the initial value $\langle 4 \rangle$; a remote live variable x with $id_x = 1$; a simple live variable y; and a snapshot live variable z. The right side of the figure illustrates how the values change over time. At t_4 when S has a new value, the value of x is also updated, which immediately triggers the recomputation of y. When y is not synchronized

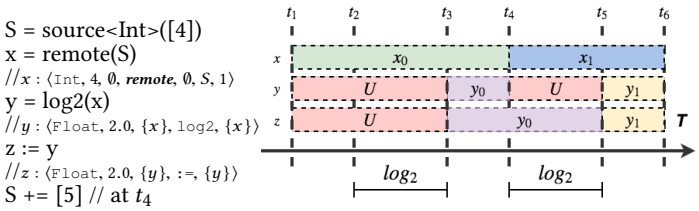


Figure 5: Example of Basic Live Variable Types.

(during the execution of the $\log_2$ function), z keeps the old value. Once y is synchronized, $v_z \leftarrow v_y$.

5 PACKET SELECTOR

On top of the live variable system, *stream attribute* and *packet selector* enable fine-grained packet selection utilizing internal states in network functions. We now provide the details.

5.1 Stream Attribute

Stream. A stream $\delta_{\mathcal{H},\Pi}$ represents a sequence of packets with common header field values. $\delta_{\mathcal{H},\Pi}$ is uniquely defined by a sequence of header fields, $\mathcal{H} \triangleq \langle h_1, \ldots, h_{|\mathcal{H}|}\rangle$, and a sequence of header field values, $\Pi \triangleq \langle \pi_1, \ldots, \pi_{|\Pi|}\rangle$, where $\forall i$, π_i is a valid value of h_i. $\mathcal{H}$ is called the *stream type* and can contain any standard packet header field, *e.g.*, source IPv4 address and TCP/UDP destination port number.

For a packet pkt and a header field h, we use $pkt.h$ to denote the value of h in pkt, *e.g.*, $pkt.sip$ represents the source IPv4 address of pkt. A packet pkt is in a stream $\delta_{\mathcal{H},\Pi}$, denoted as $pkt \in \delta$, if and only if $\forall i, 1 \le i \le |\mathcal{H}|, pkt.h_i = \pi_i$.

For example, consider the following two packets:

$$pkt_1 = \langle sip = 192.168.1.2, dip = 192.168.1.3, \ldots\rangle,$$
$$pkt_2 = \langle sip = 192.168.1.2, dip = 192.168.1.4, \ldots\rangle.$$

If $\mathcal{H} = \langle sip\rangle$, pkt_1 and pkt_2 are in the same stream, specifically, $\delta_{\langle sip\rangle, \langle 192.168.1.2\rangle}$. However, if $\mathcal{H} = \langle sip, dip\rangle$, the two packets belong to different streams.

Stream Attribute. A stream attribute x_δ represents a specific internal state about the stream $\delta_{\mathcal{H},\Pi}$ in a network function. Let T_{sa} and n_{sa} denote the type and the name of this internal state, x_δ is specified as a *remote live variable*:

$$x_\delta : \langle T_{sa}, value, \emptyset, \textbf{\textit{remote}}, \emptyset, source, uuid \equiv (n_{sa}, \mathcal{H}, \Pi)\rangle. \quad (4)$$

For example, consider an intrusion detection system which exposes a `Boolean` internal state called "is_src_infected", which indicates that the source host is diagnosed as infected. Assume host 192.168.1.2 is infected, the stream attribute of $\delta_{\langle sip\rangle, \langle 192.168.1.2\rangle}$ has the following format:

$$x_\delta : \langle \texttt{Boolean}, true, \ldots, (\text{"is\_src\_infected"}, \langle sip\rangle, \langle 192.168.1.2\rangle)\rangle.$$

Stream Attribute Schema. As we can see, stream attributes representing the same internal state have some common properties. Thus, we use a *stream attribute schema* ϑ to uniquely represent these common properties. Specifically, ϑ is specified as a tuple: $\vartheta : \langle T_{sa}, n_{sa}, \mathcal{H}\rangle$, where T_{sa}, n_{sa} and $\mathcal{H}$ have the same meaning as in (4). The following illustrates the schema for the stream attribute "is_src_infected":

$$\vartheta : \langle \texttt{Boolean}, \text{"is\_src\_infected"}, \langle sip\rangle\rangle.$$

For simplicity, for a packet pkt and a stream attribute schema $\vartheta = \langle T_{sa}, n_{sa}, \mathcal{H}\rangle$, we define $pkt.\vartheta$ as follows:

$$pkt.\vartheta \triangleq \langle T_{sa}, value, \emptyset, \textbf{\textit{remote}}, \emptyset, source, (n_{sa}, \mathcal{H}, \langle pkt.h_1, \ldots, pkt.h_{|\mathcal{H}|}\rangle)\rangle$$

It is important to note that, by properly specifying the stream type $\mathcal{H}$ of each schema, the programmer can benefit from significantly reduced memory usage on the controller at runtime. For example, she can specify either $\mathcal{H}_1 = \langle sip, dip\rangle$ or $\mathcal{H}_2 = \langle sip\rangle$ for the schema "is_src_infected". Although $\mathcal{H}_1$ and $\mathcal{H}_2$ both guarantee that packets in the same stream share the same internal state, if she uses $\mathcal{H}_1$, TRIDENT needs to keep track of up to N distinct streams, where N is the number of all possible (*sip*, *dip*) combinations. However, if she uses $\mathcal{H}_2$, TRIDENT only keeps M distinct streams, where $M \ll N$ denotes the number of all possible *sip*.

5.2 Packet Selector

On top of the preceding data models, we now define the model of *packet selector*s, which are used to select packets into different streams based on their header field values and stream attribute values. Each packet selector λ : `Packet` $\mapsto \{0, 1, unknown\}$ selects packets into a stream δ such that $\delta = \{pkt | \lambda(pkt) = 1\}$, and is one of the following two types:

- a *header field packet selector*, specified as $\lambda : \langle h, op, x\rangle$, where h is a header field, x is a live variable, $op : D_h \times D_x \mapsto \{0, 1, unknown\}$, and $\lambda(pkt) \triangleq op(pkt.h, v_x)$, or
- a *stream attribute packet selector* specified as $\lambda : \langle \vartheta, op, x\rangle$, where $\vartheta = \langle T_{sa}, n_{sa}, \mathcal{H}\rangle$ is a stream attribute schema, x is a live variable, $op : D_{T_{sa}} \times D_x \mapsto \{0, 1, unknown\}$, and $\lambda(pkt) \triangleq op(pkt.\vartheta, v_x)$.

Following are examples of a header field packet selector and a stream attribute packet selector, respectively.

$$\langle sip, in, 192.168.1.0/24\rangle, \langle http\_uri, =, \text{"www.xyz.com"}\rangle.$$

At runtime, TRIDENT translates each packet selector into proper OpenFlow match conditions, as will be discussed in §7.2. Note that a programmer can avoid working with packet selectors directly; as specified in §3.2, she can specify conditions for packet selection algorithmically and let TRIDENT automatically generate the corresponding packet selectors, which will also be discussed in §7.2.

6 ROUTE ALGEBRA

TRIDENT introduces route algebra, a simple yet powerful programming abstraction to flexibly express consistent, advanced routing. We now formally specify its details, by starting with defining basic constructs of routing on top of the live variable system. We then explain *network function indicators* and *algebraic operators*.

6.1 Route Object

We first specify basic constructs of routing strategies on top of the live variable system.

Network Component. In TRIDENT, a network topology has three types of network components: a vertex u, a port p

and a link l. TRIDENT models each network component as a *remote live variable*. Live variables for u, p, and l are denoted as x_u, x_p and x_l respectively and specified as follows:

$$x_u : \langle \texttt{Vertex}, value \equiv \check{u}, \emptyset, remote, \emptyset, source, id(u)\rangle,$$
$$x_p : \langle \texttt{Port}, value \equiv \check{p}, \emptyset, remote, \emptyset, source, id(p)\rangle,$$
$$x_l : \langle \texttt{Link}, value \equiv \check{l}, \emptyset, remote, \emptyset, source, id(l)\rangle.$$

The values of these remote live variables are assigned from domain $\{0, 1\}$, indicating whether the corresponding component is physically functioning, *e.g.*, $\check{l} = 1$ means link l is up and configured correctly.

Network Attribute. A network component live variable $x \in \{x_u, x_p, x_l\}$ has multiple *attributes*, such as node label, port statistics and link capacity, which is used to help select routes in route algebra. For simplicity, we call these attributes *network attributes*. A network attribute na_x for x is a remote live variable, whose *uuid* is determined by *1)* the *uuid* of x, and *2)* the *name* of this attribute, denoted as n_{na}.

Similar to *stream attribute schema*, we specify a *network attribute schema* θ as a tuple $\langle type, comp\_type, name\rangle$ (denoted as $\langle T_\theta, C_\theta, n_\theta\rangle$). With a supporting live variable x and a schema θ where $T_x = C_\theta$, one can obtain an attribute $na = x.\theta$ where $T_{na} = T_\theta$, $n_{na} = n_\theta$. For example, for a vertex live variable $x_v : \langle \texttt{Vertex}, 1, \ldots, \textit{"openflow:1"}\rangle$ and a stream attribute schema $\theta = \langle \texttt{String}, \texttt{Vertex}, \textit{"label"}\rangle$, $x_v.\theta = \langle \texttt{String}, value, \ldots, uuid = (\textit{"openflow:1"}, \textit{"label"})\rangle$.

Route. A route r is a path in a network and consists of a sequence of links $L_r = \langle l_{r,1}, \ldots, l_{r,|L_r|}\rangle$. It has a source port and a destination port, denoted as $src_r$and dst_r. Two routes r_1 and r_2 are *1) equal*, denoted as $r_1 = r_2$, if and only if $|L_{r_1}| = |L_{r_2}|$ and $\forall 1 \le i \le |L_{r_1}|$, $l_{r_1,i} = l_{r_2,i}$, and *2) equivalent*, denoted as $r_1 \sim r_2$, if and only if $src_{r_1} = src_{r_2}$ and $dst_{r_1} = dst_{r_2}$.

We say a route r is *valid* if and only if *1)* $\forall l \in L_r$, $\check{l} = 1$, and *2)* $\forall 1 \le i < |L_r|$, the destination port of $l_{r,i}$ is the same as the source port of $l_{r,i+1}$, and we use $\check{r} \in \{0, 1\}$ to indicate whether r is valid. Since the validity of a route r depends on the physical status of network components, TRIDENT models r as a *simple live variable*. We call this live variable as a *route object*, denoted as x_r, which is specified as follows:

$$x_r : \langle type \equiv \texttt{Route}, value \equiv r, deps, operator, operands\rangle, \quad (5)$$

and can be constructed in multiple ways:

- enumerating links $l_1, \ldots, l_n$, *i.e.* $v_{x_r} = \langle l_1, \ldots, l_n\rangle$;
- concatenating two route objects x_{r_1} and x_{r_2}, denoted as $x_{r_1} + x_{r_2}$, *i.e.* $v_{x_r} = \langle l_{r_1,1}, \ldots, l_{r_1,|L_{r_1}|}, l_{r_2,1}, \ldots, l_{r_2,|L_{r_2}|}\rangle$;
- inverting a route object x_{r_1}, denoted as $\asymp r_1$, *i.e.* $v_{x_r} = \langle \asymp l_{r_1,|L_{r_1}|}, \ldots, \asymp l_{r_1,1}\rangle$ where $\asymp l$ means inverting the source and destination ports of this link.
- selecting from a *route set* Δ for a stream δ, denoted as $\psi(\Delta, \delta)$, whose details are given in §6.2.

6.2 Route Algebra

Network Function Indicator. A network function indicator (or simply an *indicator*) is a grammar that describes how a flow traverses different network functions, using *waypoints* and *patterns*. A *waypoint* represents all instances of the same network function. A *pattern* is either *1) unidirectional*, which is encoded as $\star\texttt{X}\star$ where $\texttt{X} \in \{-, >\}$ represents the traversal rules only in one direction, or *2) bidirectional*, which is encoded as $\texttt{:X:}$ where $\texttt{X} \in \{-, >, <, =\}$ represents the traversal rules for the same flow in both the forward and reversed direction. Consider two consecutive network functions in a chain, denoted as waypoints a and b. If $X \in \{<, =\}$, a flow can use different instances of b in its life cycle. If $X \in \{>, =\}$, the return flow can use a different instance of a. All patterns in an indicator are either unidirectional or bidirectional.

An indicator has the following format:

$$w_1\ rp_1\ w_2\ \ldots\ w_{|\mathcal{W}|-1}\ rp_{|\mathcal{W}|-1}\ w_{|\mathcal{W}|},$$

which interleaves a waypoint sequence $\mathcal{W} = \langle w_1, \ldots, w_{|\mathcal{W}|}\rangle$ and a sequence of $|\mathcal{W}| - 1$ patterns $\langle rp_1, \ldots, rp_{|\mathcal{W}|-1}\rangle$.

Route Set. A route set x_Δ is a live variable, where Δ is *conceptually* a set of *valid* routes, with the extensions below:

- A route r is *equivalently in* a route set Δ, denoted as $r \in_\sim \Delta$, if and only if $\exists r' \in \Delta$, $r \sim r'$ (as defined in §6.1).
- For a stream δ, a route r can be selected from a route set Δ, *i.e.*, $r = \psi(\Delta, \delta)$ only if $r \in \Delta$, $\check{r} = 1$ and $src_r = ingress_\delta$. Formally, a route set x_Δ is specified as follows:

$$x_\Delta : \langle type \equiv \texttt{RouteSet}, value \equiv \Delta, deps, operator, operands, expr\rangle \quad (6)$$

where *expr* is explained below, while the rest are from (1):

- *expr*: an expression which describes the computation process of this route set live variable.

A route set x_Δ can be constructed in multiple ways:

- enumerating route objects $x_{r_1}, \ldots, x_{r_n}$, *i.e.* $\Delta = \{r_1, \ldots, r_n\}$;
- computed by a routing function $f : Y_1 \times \cdots \mapsto \texttt{RouteSet}$ with operands $\mathcal{Y}$, *i.e.* $\Delta = f(v_{y_{x,1}}, \ldots, v_{y_{x,|\mathcal{Y}|}})$;
- using an *algebraic operator* to construct a route set, where the result is as specified in Table 1.

Note that some algebraic operators are not independent and can be equally represented using other operators. However, their use can improve performance, *e.g.*, arbitrary selection can search for one valid route, which is much faster than using optimal selection with a constant cost function.

Table 2 illustrates how to express routing requirements for some network functions and scenarios using route algebra.

7 IMPLEMENTATION

We now give details of how TRIDENT can be efficiently implemented. In particular, we introduce: *1)* how live variables are stored and managed, *2)* how to generate OpenFlow rules, and *3)* how routes are computed efficiently in TRIDENT.

7.1 Live Variable Management

As illustrated in Figure 6, TRIDENT organizes live variables in *tables*. Different live variable types are handled in slightly different ways, as we introduce below.

Union ($\cup$)/Intersection ($\cap$)/Difference ($\setminus$)

Given two route set Δ_1 and Δ_2, return the union/intersection/difference of Δ_1 and Δ_2:

$$\Delta_1 \cup \Delta_2 = \{r \mid r \in \Delta_1 \vee r \in \Delta_2\},$$
$$\Delta_1 \cap \Delta_2 = \{r \mid r \in \Delta_1 \wedge r \in \Delta_2\},$$
$$\Delta_1 \setminus \Delta_2 = \{r \mid r \in \Delta_1 \wedge r \notin \Delta_2\}.$$

Union ($\cup_\sim$)/Intersection ($\cap_\sim$)/Difference ($\setminus_\sim$) by Equivalence

Given two route set Δ_1 and Δ_2, return the union/intersection/difference of Δ_1 and Δ_2 using $\in_\sim$ instead of $\in$:

$$\Delta_1 \cup_\sim \Delta_2 = \{r \in \Delta_1 \cup \Delta_2 \mid r \in_\sim \Delta_1 \vee r \in_\sim \Delta_2\},$$
$$\Delta_1 \cap_\sim \Delta_2 = \{r \in \Delta_1 \cup \Delta_2 \mid r \in_\sim \Delta_1 \wedge r \in_\sim \Delta_2\},$$
$$\Delta_1 \setminus_\sim \Delta_2 = \{r \in \Delta_1 \cup \Delta_2 \mid r \in_\sim \Delta_1 \wedge r \notin_\sim \Delta_2\}.$$

Concatenation (+)

Given two route sets Δ_1 and Δ_2, return a new route set by concatenating all route pairs (r_1, r_2) in $\Delta_1 \times \Delta_2$ and removing the invalid ones:

$$\Delta_1 + \Delta_2 = \{r_1 + r_2 \mid r_1 \in \Delta_1, r_2 \in \Delta_2, dst_{r_1} = src_{r_2}\}.$$

Inversion ($\asymp$)

Given a route set Δ, return the inverse of $r \in \Delta$:

$$\asymp \Delta = \{\asymp r \mid r \in \Delta\}.$$

Preference ($\triangleright$)

Given two route sets Δ_1 and Δ_2, return the *preferred* route. (If there is an equivalent route in Δ_1, do not use the ones in Δ_2):

$$\Delta_1 \triangleright \Delta_2 = \{r \mid r \in \Delta_1 \vee (r \in \Delta_2 \wedge \nexists r' \in \Delta_1, r \sim r')\}.$$

Selection (σ)

Given a route set Δ and an evaluation function $f : R^* \mapsto \{0, 1\}$, return all routes in Δ that are evaluated as 1:

$$\sigma_f(\Delta) = \{r \in \Delta \mid f(r) = 1\}.$$

Optimal selection ($\diamond$)

Given one route set Δ and a routing cost function $d : R^* \mapsto \mathbb{R}$, return *any* route with the minimum value:

$$\diamond_d(\Delta) = \arg\min_{r \in \Delta} d(r).$$

Arbitrary selection ($*$)

Given one route set Δ, return a route set containing exactly one route r in Δ:

$$*\Delta = \diamond_1(\Delta).$$

Table 1: Algebraic Operators.

Description	*Expression*
Bro	I :>: DPI :−: H
Bro (in OpenNF)	(I :>: DPI :−: H) $\triangleright$ (I :=: DPI :−: H)
Shortest QoS Path	$\diamond_{QoS\text{-}f}(X + Y)$
Path Preference	$\sigma_{bw=100Gbps}(X) \triangleright \sigma_{bw=10Gbps}(X)$
"Make before break"	:= (X)
"Return to the same NF"	$x \leftarrow *(X)$, $y \leftarrow *(X \cap_\sim (\asymp x))$
"Return to another NF"	$x \leftarrow *(X)$, $y \leftarrow *(X \setminus_\sim (\asymp x))$

I - Internet, H - data center hosts, DPI - deep packet inspection, X, Y - some route sets,
x, y - temporary variables storing a route.

Table 2: Expressiveness of Route Algebra: Examples.

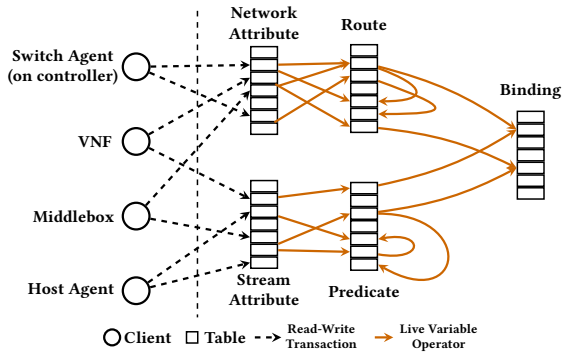


Figure 6: Tabular View of Live Variable System.

Managing Remote Live Variable. Remote live variables are updated by TRIDENT-compatible network functions. Each remote live variable x is mapped to a specific table cell, where the key is a tuple of (*table_key*, *cell_key*). Table key is the type name for a remote live variable, while cell key is the *uuid* of x. For example, the label of vertex "openflow:1" has a table key `"label"` and a cell key `"openflow:1"`, and the stream attribute "is_infected" for host *10.0.1.5* has a table key `"is_infected"` and a cell key `"10.0.1.5"`.

Managing Simple/Snapshot Live Variables. A simple or snapshot live variable is constructed using an operator. Even though Figure 6 uses an arrow from one source cell to one destination cell to represent an operator, it typically involves multiple source cells and multiple destination cells. These "internal" variables are managed using the dependency tracking and change propagation rules as defined in §4.2.

7.2 Binding Generation and Translation

We now introduce how TRIDENT automatically generates bindings from a unified SDN program and then translates the bindings into OpenFlow rules.

Automatic Binding Generation. TRIDENT automatically creates bindings as follows:

1) Before evaluating a unified SDN program for a given packet *pkt*, TRIDENT creates an execution *context*, which maintains a sequence of packet selectors Λ and a set of bindings $\mathcal{B}$ for this packet, which are both empty at the beginning.

2) Whenever a stream attribute $pkt.\vartheta$ or a packet header field $pkt.h$ is compared with a live variable x, TRIDENT appends a packet selector $\langle \vartheta, op, x \rangle$ or $\langle h, op, x \rangle$ to Λ.

3) Whenever a `bind` is invoked with a packet *pkt* and a route set Δ, TRIDENT adds a binding $\langle \Lambda, \Delta \rangle$ to $\mathcal{B}$.

For an inverted packet (created by `inv(pkt)`), the match field in each packet selector must be inverted before the translation, *e.g.*, `pkt.sip ==="10.0.1.5"` is converted to `inv(pkt).dip ==="10.0.1.5"`.

Automatic Binding Translation. We first discuss how TRIDENT generates a match for a single packet selector and then explain how it handles the rule translation.

For a given packet *pkt*, TRIDENT uses predicates to construct a match for a packet selector λ as follows: *1)* If λ is a header field packet selector, *i.e.*, $\lambda = \langle h\ op\ x \rangle$, the predicate is $h\ op\ v_x$. *2)* If λ is a stream attribute packet selector, *i.e.*, $\lambda = \langle \vartheta\ op\ x \rangle$ where $\vartheta = \langle T_{sa}, n_{sa}, \mathcal{H} \rangle$, it is converted to a sequence of predicates: $\langle (h_1, =, pkt.h_1), \ldots, (h_{|\mathcal{H}|}, =, pkt.h_{|\mathcal{H}|}) \rangle$.

For example, consider the following three packet selectors:

```
p1 = pkt.sip in "10.0.1.0/24"
p2 = pkt.http_uri === "www.xyz.com"
```

```
p3 = pkt.is_infected === true
```

If a packet with tuple ⟨*10.0.1.5*, *10.0.2.6*, 31415, 80, *tcp*⟩ satisfies all three predicates, the corresponding matches are:

Name	*sip*	*dip*	*sport*	*dport*	*ipproto*
p1	10.0.1.0/24	*	*	*	*
p2	10.0.1.5/32	10.0.2.6/32	31415	80	tcp
p3	10.0.1.5/32	*	*	*	*

The match of a stream attribute can be decomposed as the intersection of multiple header field matches. For example, p3 is translated into `pkt.sip==="10.0.1.5"`.

TRIDENT generates OpenFlow rules using this property: it creates packet traces with only header fields by replacing each stream attribute packet selector with multiple header field packet selectors. Then, it uses Maple's trace tree [10] to generate flow rules.

7.3 Efficient Change Propagation

Simple change propagation (§4.2) is inefficient for route computation, since a route set can be the result of a complex routing function, such as a shortest path algorithm and traffic engineering. During such computation, the result becomes unknown with no valid routes available. To reduce the latency without compromising consistency guarantees in §4.1, we introduce *efficient change propagation*.

Efficient change propagation works as follows. We split a route set Δ into two subsets: a known subset $\mathcal{K}$ and an unknown subset $\mathcal{U}$[1]. We process the two subsets separately when evaluating a route algebra expression, as shown in Table 3. If the unknown subset of an expression becomes empty, usually after an arbitrary selection or a snapshot, we use the known subset as the value of this expression.

Now we prove that efficient change propagation still guarantees glitch-free consistency, by showing that the result is the same as simple change propagation.

Definition 3 (Equivalent Propagation)**.** Let x be the result of a route algebra expression f which depends on N route sets, denoted as $\Delta_1, \ldots, \Delta_N$, *i.e.*, $x = f(\Delta_1, \ldots, \Delta_N)$. For a given root cause $\kappa \prec x$, two propagation processes $\psi_1(x, \kappa)$ and $\psi_2(x, \kappa)$ are equivalent, if and only if their synchronized values $v_{x_{\psi_1}} \sim v_{x_{\psi_2}}$.

Proposition 2. Efficient change propagation guarantees glitch-free consistency.

PROOF. **Sketch:** We prove it by showing that efficient change propagation and simple change propagation are equivalent. If the unknown subset is empty, it means the result does not depend on the unknown subsets so when they become known, the value does not change. □

[1]If a route set itself is unknown, its *known* subset is $\emptyset$.

Expr	*Known Subset*	*Unknown Subset*
$\Delta_1 \cup \Delta_2$	$\mathcal{K}_1 \cup \mathcal{K}_2$	$\mathcal{U}_1 \cup \mathcal{U}_2$
$\Delta_1 \cap \Delta_2$	$\mathcal{K}_1 \cap \mathcal{K}_2$	$(\mathcal{K}_1 \cap \mathcal{U}_2) \cup (\mathcal{U}_1 \cap \mathcal{K}_2) \cup (\mathcal{U}_1 \cap \mathcal{U}_2)$
$\Delta_1 \setminus \Delta_2$	$T_{\mathcal{U}_2=\emptyset}(\mathcal{K}_1 - \mathcal{K}_2)$	$(T_{\neg(\mathcal{U}_2=\emptyset)}(\mathcal{K}_1) \cup \mathcal{U}_1) - (\mathcal{K}_2 \cup \mathcal{U}_2)$
$\Delta_1 + \Delta_2$	$\mathcal{K}_1 + \mathcal{K}_2$	$(\mathcal{K}_1 + \mathcal{U}_2) \cup (\mathcal{U}_1 + \mathcal{K}_2) \cup (\mathcal{U}_1 + \mathcal{U}_2)$
$\asymp \Delta$	$\asymp \mathcal{K}$	$\asymp \mathcal{U}$
$\sigma_f(\Delta)$	$\sigma_f(\mathcal{K})$	$\sigma_f(\mathcal{U})$
$\diamond_d(\Delta)$	$T_{\mathcal{U}=\emptyset}(\diamond_d(\mathcal{K}))$	$\diamond_d(T_{\neg(\mathcal{U}=\emptyset)}(\diamond_d(\mathcal{K})) \cup \diamond_d(\mathcal{U}))$
$\Delta_1 \triangleright \Delta_2$	$\mathcal{K}_1 \cup T_{\mathcal{U}_1=\emptyset}(\mathcal{K}_2 - \mathcal{K}_1)$	$\mathcal{U}_1 \cup ((T_{\neg(\mathcal{U}_1=\emptyset)}(\mathcal{K}_2) \cup \mathcal{U}_2) \setminus (\mathcal{K}_1 \cup \mathcal{U}_1))$
$*\Delta$	$*\mathcal{K}$	$T_{\mathcal{K}=\emptyset}(*\mathcal{U})$

$T_\varepsilon(S)$ - the value is $S \cup \{\varepsilon\}$ if $\varepsilon = true$, and $\{\varepsilon\}$ otherwise.

Table 3: Known/Unknown Subsets of Route Algebra.

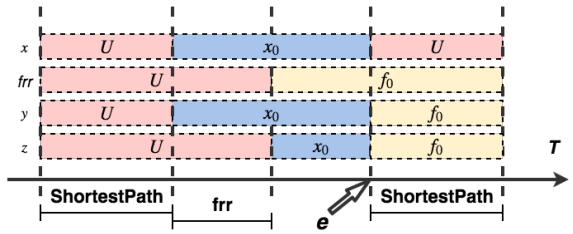


Figure 7: Results of the example in different stages. x - the value of primary path, *frr* - the value of backup path, y - the result using efficient change propagation, z - the result using simple change propagation.

Example. We use an example to demonstrate how the efficient change propagation can reduce latency. Consider the following program: use the primary path (x) if it is available, and use the backup path (*frr*) during re-computation:

```
1  val x = ShortestPath(G, s, t)
2  val xs = snapshot(x)
3  val y = any(xs >> frr(xs))
```

where `ShortestPath` and `frr` are two routing functions.

Figure 7 demonstrates how the *efficient change propagation* helps reduce the delay. As we can see, after the first execution of `ShortestPath`, there is one path x_0 between s and t, but the value of the backup route (*frr*) is still unknown (U). However, since only one valid path is demanded, the result of *frr* makes no difference as long as x_0 is available. Thus, the efficient change propagation is consistent and reduces the latency as it need not wait for **frr** to finish.

8 EVALUATION

In this section, we conduct extensive evaluation to answer the following key questions:

- How useful and expressive is TRIDENT programming framework in real network management scenarios? (§8.1)
- How robust and efficient is the TRIDENT runtime system, and what are the key factors of its performance? (§8.2)

8.1 Programming with TRIDENT

We first demonstrate the feasibility of integrating network functions into TRIDENT. We then show the expressiveness of TRIDENT through real world use cases.

Name	*Attribute*	*Language*	*LoC (f)*	*LoC (a)*	*Loc (c)*
DPI	HTTP URL	Bro	40	2	2
FreeRadius	Auth status	DSL	0	12	0

LoC - Additional lines of code, **f** - LoC to implement the library in the given framework/language, **a** - LoC in a given NF, **c** - LoC for configuration.

Table 4: Integration of Different NF(V) with TRIDENT.

Ease of Integration. We measure the complexity of integrating network functions using *additional lines of code*. Table 4 shows the results for two concrete examples.

The first row is the integration of a DPI based on Bro [35] to extract HTTP URI. Our extension requires three phases: *1)* implementing a new Bro module using the `ActiveHttp` library to send inspected results to TRIDENT (40 lines), *2)* modifying a default Bro script that exposes HTTP headers (2 lines), *3)* updating the configuration (2 lines).

The second row is the integration of FreeRadius [36], an open source RADIUS server, to extract authentication status of a host. Our extension requires 12 additional lines of its domain-specific configuration language in its `rest` module and in the site configuration file.

Expressiveness of TRIDENT. We demonstrate the expressiveness of TRIDENT with two real-world programs.

The first program implements a layer-3 routing, which delivers a packet even when the topological location of its destination host is unknown. Thus, a programmer utilizes the stream attribute `dhost` (L5), which is shared by packets with the same destination IP. If the location of a packet's destination host is already known, the packet is directly forwarded to the host using the shortest path (L13); otherwise, the packet is broadcast through a spanning tree (L10). Without TRIDENT, the programmer must manually handle complexities such as host migration or topology failures.

```
1  class L3Routing extends SDNProgram {
2    val flood: RouteSet = SpanningTree(G) // G is network topology
3    val normal: RouteSet = ShortestPath(G)
4
5    val dhost = StreamAttribute[Vertex]("dhost", DST_IPADDR)
6
7    override def onPacket(pkt: Packet) = program {
8      if (pkt.dhost?) {
9        // pkt's destination host is unknown
10       bind(pkt, flood)
11     } else {
12       val x = any(normal where { dst == pkt.dhost })
13       bind(pkt, x)
14     }
15   }
16 }
```

The second program implements a SDMZ [2], which is a network access control system for high-speed scientific data traffic. The program logic is as follows: if a packet does not belong to a scientific data set, it must go through an intrusion detection system (IDS); otherwise, the packet skips the IDS and uses high-speed scientific links as long as its destination site is allowed to accept the packet's scientific data set. To achieve this goal, the programmer utilizes two stream attributes: *1)* `dataset_id` (L7), shared by packets with the same TCP 5-tuple, and *2)* `dst_site` (L8), shared by packets with the same source IP address, as data transfers are initiated by the receiver. The program groups waypoints by label (L10-12) and constructs three route sets (L13-16): `route_ids` for non-scientific traffic going through the IDS; `sdmz` and `backup` for high-speed scientific traffic bypassing the IDS, where `sdmz` uses only links with more than 40 Gbps. If a packet's dataset ID is unknown, the program forwards the packet using `route_ids` (L20); otherwise, the program extracts the packet's destination site (L22-28). If the destination site can accept the packet's data set, the program forwards the packet using `sdmz`, or if that is not feasible, using `backup`; otherwise, the program drops the packet.

Again, without TRIDENT, the programmer must explicitly handle changes of `dataset_id`, `dst_site`, `acl` to correctly forward a packet, and also topology changes to properly install the backup route.

```
1  case class ACL(val prefix: String, val allow: Boolean)
2
3  class SDMZ(val site: String) extends SDNProgram {
4    import trident.example.Net.{capacity, label} // See Section 3.2
5    val acl = SetTable[String, ACL]("acl")
6
7    val dst_site = StreamAttribute[String]("dst_site", SRC_IPADDR)
8    val dataset_id = StreamAttribute[String]("dataset_id", TCP5TUPLE)
9
10   val H = Waypoint(V where { label === "host" })
11   val IDS = Waypoint(V where { label === "ids" })
12   val GW = Waypoint(V where { label === "gateway" })
13   val route_ids = SimplePath(G, H :-: IDS :-: GW)
14                                where (capacity <= 10 Gbps)
15   val sdmz = SimplePath(G, H :-: GW) where (capacity >= 40 Gbps)
16   val backup = SimplePath(G, H :-: GW)
17
18   override def onPacket(pkt: Packet) = program {
19     if (pkt.dataset_id?) {
20       bind(pkt, any(route_ids))
21     } else {
22       val dst_site = iff (pkt.dst_site === site) {
23         // Incoming traffic, reverse the packet
24         inv(pkt).dst_site
25       } else {
26         // Outgoing traffic or unknown, use normal destination site
27         pkt.dst_site
28       }
29       iff (dst_site.acl.first(r => pkt.dataset_id == r.prefix).allow) {
30         bind(pkt, any(sdmz >> backup))
31       } else {
32         drop(pkt)
33       }
34     }
35   }
36 }
```

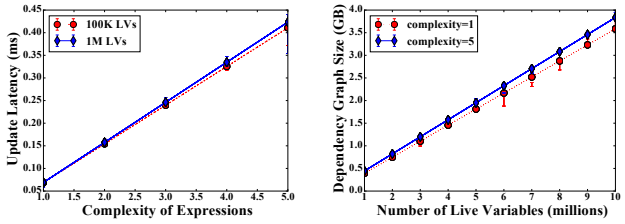
8.2 TRIDENT Runtime Performance

We evaluate TRIDENT's performance by each of our main components: live variable, stream attribute and route algebra.

Setting. We evaluate our prototype implementation on a Fedora 26 machine with an Intel Xeon CPU E5-2650 2.30GHz processor and 64G of memory, and the network is simulated with Mininet 2.3.0d1.

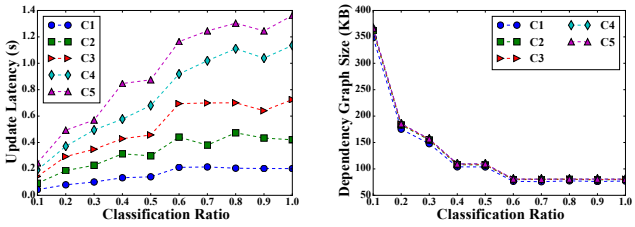
8.2.1 Live Variable. We conduct a microbenchmark test to evaluate how efficiently TRIDENT can handle dependency tracking, when it is scaled with the number of live variables and the number of directly dependent live variables.

Setup. We first generate up to 10 million live variables of integer type and assign a randomly generated arithmetic expression to each live variable as its dependency. We vary the



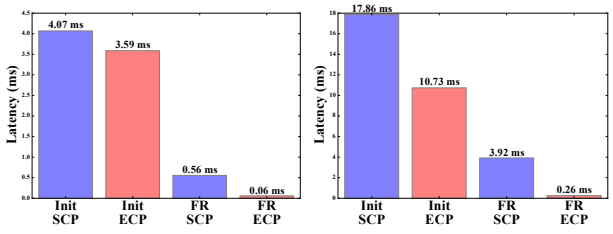
(a) Update Latency (b) Dependency Graph Size

Figure 8: Live Variable Microbenchmark

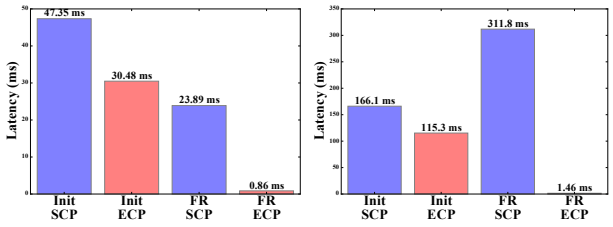


(a) Update Latency (b) Dependency Graph Size

Figure 9: Stream Attribute Benchmark



(a) Sprint (11 nodes) (b) Noel (19 nodes)



(c) Agis (25 nodes) (d) Geant (40 nodes)

Figure 10: Route Algebra Benchmark

complexity of the expressions for each experiment, so that an expression of complexity n directly depends on n other live variables. Finally, We update the value of a randomly chosen live variable every 0.01 second.

Results. Figure 8(a) shows the update latency for expressions of complexity 1 to 5, when there are 100,000 and 1,000,000 live variables. We mean the update latency by the time TRIDENT spends to correctly update all dependent variables when the value of a live variable was updated. As shown in the graph, the latency is linearly correlated with the expression complexity, while it has little relationship with the number of live variables. Figure 8(b) shows the total size of compiled dependency graphs for one to ten million live variables. Unlike the update latency, the dependency graph size is linearly correlated with the number of live variables, whereas it is relatively independent of the expression complexity. From the results, we infer that a programmer can achieve faster updates by reducing expression complexity and reduce memory usage by using fewer live variables.

8.2.2 Stream Attribute. As noted in §5.1, a programmer can enjoy reduced memory usage on the controller at runtime by specifying the proper stream type when she defines a stream attribute schema. To demonstrate this effect, we evaluate the performance of packet selection when a different fraction of packets are classified into each stream.

Setup. We first declare 100 random stream attribute schemas, all of which have source IP address as the stream type, so that we can easily control the fraction of packets classified into each stream. Based on the schemas, we randomly generate 250 packet selectors. For each experiment, we vary the complexity of packet selectors so that for complexity n, each packet selector depends on n distinct schemas. We then send 10,000 packets to the network simultaneously, where for each experiment, we vary the fraction η of all packets that share the same source IP address. We call this fraction *classification ratio*. Finally, we update a randomly chosen stream attribute every 1 *ms* and measure the performance of automatic recomputation of packet selector.

Results. Figure 9(a) and Figure 9(b) illustrate the update latency and the total dependency graph size respectively, over different classification ratios from 0.1 to 1.0. As shown in Figure 9(b), the dependency graph size decreases with a higher classification ratio, regardless of the expression complexity. This result indicates that a proper specification of the stream type for a schema can significantly reduce the memory usage on the controller at runtime. However, as shown in Figure 9(a), the update latency increases with a higher classification ratio, especially when more complex packet selectors are used. This behavior is reasonable as more number of packet selectors will need to be recomputed upon an attribute update; however, it is important for an programmer to be aware of this compromise, especially when she uses a more complex packet selector.

8.2.3 Route Algebra. The key benefit of route algebra is the efficient change propagation (§7.3). We show this benefit by comparing the performance of *efficient change propagation* (`ECP`) and *simple change propagation* (`SCP`) during the initial computation (**Init**) and failure recovery (**FR**) stages.

Setup. We use the following code segment (also listed in §7.3) to compute route sets on four network topologies from Topology zoo [37]: Sprint, Noel, Agis, and Geant.

```
1  val x = ShortestPath(G, s, t)
2  val xs := x
3  val y = any(xs >> frr(xs))
```

Specifically, we first evaluate the total time TRIDENT spends to compute route set `y` for all node pairs in each topology (initial computation). Then, we create link failure on an arbitrary link and measure the total time to recover valid `y` for all node pairs (failure recovery). We repeat this experiment 100 times for each topology to take the average.

Results. Figure 10 shows the results. For the initial computation, TRIDENT achieves consistently lower latency with the ECP (second bar) than with the SCP (first bar) across all topologies. For Agis (Figure 10(d)) as an example, TRIDENT uses 30.5 ms with the ECP, while it spends 47.4 ms with the SCP, *i.e.* 36% more efficient with the ECP. This is because, with the efficient change propagation, TRIDENT can proceed to execute the `any` operation as long as `xs` has been computed, even if `frr(xs)` is unknown.

TRIDENT can also recover from link failure extremely efficiently with the ECP (fourth bar), compared with the SCP (third bar), across all topologies. In particular, for Geant (Figure 10(d)), TRIDENT spends only 1.5 ms to recover with the ECP, which is more than 200 times faster than 311.8 ms with the SCP. This is because, with the efficient change propagation, TRIDENT can instantly update `y` as long as the already computed backup route set (`frr(xs)`) is still valid, even if `xs` became unknown due to link failure.

9 RELATED WORK

Integrating SDN and NFV. Some previous studies [7, 25, 38–42] also target the integration of SDN and NFV. However, they are leveraging the SDN technology to improve elasticity and fault tolerance of NFV systems, while TRIDENT targets the opposite scenario.

E2 [7] and FlowTags [4] encode "stream attribute" in the data plane, and support local decisions with the attributes. However, the routing capabilities in these systems are quite limited. TRIDENT can potentially leverage these techniques to improve the performance with a "backend" that translates bindings to FlowTags rules on FlowTag-compatible devices.

An alternative design is to use an event-driven system, such as Kinetic [6]. TRIDENT chooses a functional reactive programming approach for the sake of simplicity.

Traceable Data and Incremental Computation. The idea of live variables comes from traceable data [43, 44], incremental computation [45–47] and reactive programming [34, 48–50]. We are the first to find novel uses of the traceable data in the field of SDN programming.

DREAM [34] and REScala [50] target fast glitch-free consistency in general reactive programming. TRIDENT improves the performance mostly by leveraging domain knowledge.

CoVisor [51] and Wen *et. al* [52] aim to support incremental update as an add-on of SDN systems. However, they work at the level of flow rules, but cannot handle incremental update that generates these rules.

Automatic Dependency Tracking in SDN Systems. Several SDN systems [20, 53, 54] store network-related information in a distributed data store, where dependencies are managed by programmers manually. The Intent framework [22, 23] is adopted but the dependency tracking is not as flexible and efficient as TRIDENT.

NVP [28] also uses a table-based storage system and the `nlog` language provides built-in support to identify the dependencies among different tables. However, the language is optimized for special purposes and is only used internally, it is not clear how they work in the scenario that we target.

Statesman [24] and FAST [55] encapsulate the underlying data store to track dependencies but do no leverage the domain-specific semantics in networking. Statesman also has some pre-defined semantic dependencies but they are limited to the scope of network topology.

Route Specification in Networking. The use of waypoint-based route specification is motivated by previous studies [7, 14–19]. TRIDENT goes beyond basic network function chaining specifications in NFV systems such as E2 [7], and naturally supports QoS-based routing [56, 57]. Resource reservation (Merlin [15]) or traffic isolation (Genesis [16]) are not covered in this paper. They can be extended using customized properties in §4 and are left as future work.

With prior work such as Propane [14, 58] and DEFO [59], it is also an interesting open question that whether TRIDENT can be used to integrate network functions with BGP.

Many route algebra operators can be overwritten by relational algebra [60], *e.g.*, concatenation is a special *join*. However, they do have domain specific meanings. Sobrinho [56] introduces basic operators (concatenation, distance function) but we extend them to a set of routes.

10 CONCLUSION AND FUTURE WORK

In this paper, we introduce TRIDENT, a novel unified programming framework to accommodate network functions in SDN programming. Specifically, TRIDENT introduces *live Kleene variable*, *stream attribute* and *route algebra* to achieve unified automatic dependency tracking and incremental computation, functionally complete fine-grained packet selection, and flexible route construction. We demonstrate that TRIDENT is both expressive enough to work in various real world scenarios, and can efficiently handle changes.

Acknowledgment The authors thank Haitao Yu, Shiwei Chen, Xin Wang, Shenshen Chen (Tongji University), Alan Liu, Isabelle Carson, Yuki de Pourbaix, Lee Danilek, and Qiao Xiang (Yale University) for their help during the preparation of the paper. The authors also thank Ratul Mahajan (our shepherd) and the anonymous reviewers for their valuable comments. Kai thanks his PhD supervisor Jun Bi (Tsinghua University) for his strong support. The research was supported in part by NSFC grants NSFC #61672385 and #61472213, NSF grant CC-IIE #1440745, Google Research Award, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, and National Key Research and Development Plan of China (2017YFB0801701).

REFERENCES

[1] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue. Contextual, flow-based access control with scalable host-based SDN techniques. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, INFOCOM, pages 1–9, April 2016.

[2] Vasudevan Nagendra, Vinod Yegneswaran, and Phillip Porras. Securing Ultra-High-Bandwidth Science DMZ Networks with Coordinated Situational Awareness. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 22–28, New York, NY, USA, 2017. ACM.

[3] Sungmin Hong, Robert Baykov, Lei Xu, Srinath Nadimpalli, and Guofei Gu. Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security. NDSS'16. Internet Society, 2016.

[4] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546, Berkeley, CA, USA, 2014. USENIX Association.

[5] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. Practical Declarative Network Management. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 1–10, New York, NY, USA, 2009. ACM.

[6] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable Dynamic Network Control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 59–72, Oakland, CA, 2015. USENIX Association. 00053.

[7] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[8] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A Network Programming Language. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 279–291, New York, NY, USA, 2011. ACM. 00547.

[9] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing Software Defined Networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 1–13, Lombard, IL, 2013. USENIX Association.

[10] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying SDN Programming Using Algorithmic Policies. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 87–98, New York, NY, USA, 2013. ACM. 00143.

[11] Ryan Beckett, Michael Greenberg, and David Walker. Temporal NetKAT. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 386–401, New York, NY, USA, 2016. ACM.

[12] Chaithan Prakash, Ying Zhang, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, and Puneet Sharma. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. SIGCOMM'15, pages 29–42. ACM Press, 2015. 00032.

[13] R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, and T Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[14] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don'T Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 328–341, New York, NY, USA, 2016. ACM.

[15] Robert SoulÃľ, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. Merlin: A Language for Provisioning Network Resources. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 213–226, New York, NY, USA, 2014. ACM.

[16] Kausik Subramanian, Loris D'Antoni, and Aditya Akella. Genesis: Synthesizing Forwarding Tables in Multi-tenant Networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 572–585, New York, NY, USA, 2017. ACM.

[17] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic Foundations for Networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 113–126, New York, NY, USA, 2014. ACM. 00163.

[18] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. FatTire: Declarative Fault Tolerance for Software-defined Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 109–114, New York, NY, USA, 2013. ACM.

[19] Srinivas Narayana, Mina Tahmasbi, Jennifer Rexford, and David Walker. Compiling Path Queries. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 207–222, Santa Clara, CA, 2016. USENIX Association. 00029.

[20] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven SDN controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2014. 00092.

[21] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, MartÃŋn Casado, Nick McKeown, and Scott Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008. 01452.

[22] ONOS. Intent Framework, 2017. https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[23] OpenDaylight. Network Intent Composition, 2017. https://wiki.opendaylight.org/view/Network_Intent_Composition:Main.

[24] Peng Sun, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. A Network-state Management Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 563–574, New York, NY, USA, 2014. ACM. 00039.

[25] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. Pico Replication: A High Availability Framework for Middleboxes. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 1:1–1:15, New York, NY, USA, 2013. ACM.

[26] Wenxuan Zhou, Dong Jin, Jason Croft, Matthew Caesar, and P. Brighten Godfrey. Enforcing Customizable Consistency Properties in Software-Defined Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 73–85, Oakland, CA, 2015. USENIX Association.

[27] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu. Attacking the Brain: Races in the SDN Control Plane. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 451–468, Vancouver, BC, 2017. USENIX Association.

[28] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. Network Virtualization in Multi-tenant Datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Seattle, WA, 2014. USENIX Association. 00191.

[29] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: A Language for High-level Reactive Network Control. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[30] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. pages 29–43. ACM Press, 2016. 00007.

[31] Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. A distributed and robust sdn control plane for transactional network updates. In *2015 IEEE conference on computer communications (INFOCOM)*, pages 190–198. IEEE, 2015. 00043.

[32] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for Network Update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM. 00433.

[33] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent updates for software-defined networks: Change you can believe in! page 7. ACM, 2011. 00138.

[34] Alessandro Margara and Guido Salvaneschi. We Have a DREAM: Distributed Reactive Programming with Consistency Guarantees. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 142–153, New York, NY, USA, 2014. ACM.

[35] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.

[36] freeradius.org. FreeRADIUS - the open source implementation of RADIUS, 2018.

[37] Simon Knight, Hung X. Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011. 00249.

[38] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. OpenNF: Enabling Innovation in Network Function Control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174, New York, NY, USA, 2014. ACM. 00225.

[39] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. Protego: Cloud-Scale Multitenant IPsec Gateway. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 473–485, Santa Clara, CA, 2017. USENIX Association.

[40] Rohan Gandhi, Y. Charlie Hu, and Ming Zhang. Yoda: A Highly Available Layer-7 Load Balancer. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 21:1–21:16, New York, NY, USA, 2016. ACM.

[41] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM. 00433.

[42] Anat Bremler-Barr, Yotam Harchol, and David Hay. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 511–524, New York, NY, USA, 2016. ACM.

[43] Umut A. Acar, Guy Blelloch, Ruy Ley-Wild, Kanat Tangwongsan, and Duru Turkoglu. Traceable Data Types for Self-adjusting Computation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, pages 483–496, New York, NY, USA, 2010. ACM. 00028.

[44] Umut A. Acar, Amal Ahmed, and Matthias Blume. Imperative Self-adjusting Computation. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 309–322, New York, NY, USA, 2008. ACM.

[45] Matthew A. Hammer, Joshua Dunfield, Kyle Headley, Nicholas Labich, Jeffrey S. Foster, Michael Hicks, and David Van Horn. Incremental Computation with Names. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 748–766, New York, NY, USA, 2015. ACM.

[46] Matthew A. Hammer, Khoo Yit Phang, Michael Hicks, and Jeffrey S. Foster. Adapton: Composable, Demand-driven Incremental Computation. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 156–166, New York, NY, USA, 2014. ACM.

[47] Pramod Bhatotia, Alexander Wieder, \.Istemi Ekin AkkuÅ§, Rodrigo Rodrigues, and Umut A. Acar. Large-scale Incremental Data Processing with Change Propagation. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 18–18, Berkeley, CA, USA, 2011. USENIX Association.

[48] Guido Salvaneschi, Gerold Hintz, and Mira Mezini. REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications. In *Proceedings of the 13th International Conference on Modularity*, MODULARITY '14, pages 25–36, New York, NY, USA, 2014. ACM.

[49] Conal Elliott and Paul Hudak. Functional Reactive Animation. In *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming*, ICFP '97, pages 263–273, New York, NY, USA, 1997. ACM.

[50] Joscha Drechsler, Guido Salvaneschi, Ragnar Mogk, and Mira Mezini. Distributed REScala: An Update Algorithm for Distributed Reactive Programming. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 361–376, New York, NY, USA, 2014. ACM.

[51] Xin Jin, Jennifer Gossels, Jennifer Rexford, and David Walker. CoVisor: A Compositional Hypervisor for Software-Defined Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015. 00037.

[52] Xitao Wen, Chunxiao Diao, Xun Zhao, Yan Chen, Li Erran Li, Bo Yang, and Kai Bu. Compiling Minimum Incremental Update for Modular SDN Languages. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 193–198, New York, NY, USA, 2014. ACM.

[53] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and others. Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI*, volume 10, pages 1–6, 2010.

[54] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. ONOS: Towards an Open, Distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM. 00215.

[55] Kai Gao, Chen Gu, Qiao Xiang, Y Richard Yang, and Jun Bi. FAST: A Simple Programming Abstraction for Complex State-Dependent SDN Programming. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 579–580. ACM, 2016. 00000.

[56] JoÃčo LuÃŋs Sobrinho. Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet. *IEEE/ACM Transactions on Networking (TON)*, 10(4):541–550, 2002. 00328.

[57] Haijun Geng, Xingang Shi, Xia Yin, Zhiliang Wang, and Han Zhang. Algebra and algorithms for efficient and correct multipath QoS routing in link state networks. In *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pages 261–266. IEEE, 2015. 00000.

[58] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Network Configuration Synthesis with Abstract Topologies. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 437–451, New York, NY, USA, 2017. ACM.

[59] Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 15–28. ACM, 2015. 00026.

[60] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, June 1970.

Toward the First SDN Programming Capacity Theorem on Realizing High-Level Programs on Low-Level Datapaths

Christopher Leet*, Xin Wang†*‡, Y. Richard Yang*†, James Aspnes*

* Department of Computer Science, Yale University
† Department of Computer Science, Tongji University
‡ Key Laboratory of Embedded System and Service Computing, Ministry of Education, China

***Abstract*—High-level programming and programmable data paths are two key capabilities of software-defined networking (SDN). A fundamental problem linking these two capabilities is whether a given high-level SDN program can be realized onto a given low-level SDN datapath structure. Considering all high-level programs that can be realized onto a given datapath as the programming capacity of the datapath, we refer to this problem as the *SDN datapath programming capacity problem*. In this paper, we conduct the first study on the SDN datapath programming capacity problem, in the general setting of *high-level, datapath oblivious, algorithmic SDN programs* and state-of-art *multi-table SDN datapath pipelines*. In particular, considering datapath-oblivious SDN programs as computations and datapath pipelines as computation capabilities, we introduce a novel framework called *SDN characterization functions*, to map both SDN programs and datapaths into a unifying space, deriving the first rigorous result on SDN datapath programming capacity. We not only prove our results but also conduct realistic evaluations to demonstrate the tightness of our analysis.**

I. Introduction

A major research direction of SDN is programmable, efficient datapaths (*e.g.*, OF1.3 [1], OF-DPA [2], P4 [3]). Only by being programmable can a given SDN datapath support diverse, ever evolving application scenarios. At the same time, it is crucial that datapaths be efficient, to be able to satisfy demanding requirements such as achieving high throughput and being cost effective. In the last few years, multi-table pipelines have emerged as a key structure of SDN datapaths (*e.g.*, Domino [4], Forwarding Metamorphosis [5]).

One problem of efficient datapaths, however, is that they must often be programmed at an inefficiently low level. For example, TCAM, which is essential to achieve high-throughput, does not support logical negation. Hence, a second major research direction of SDN is high-level, datapath path-oblivious programming, to provide abstractions to hide low-level datapath programming. To this end, in the last few years multiple high-level SDN programming models have emerged (*e.g.*, Frenetic [6], Maple [7]).

As both directions progress, a basic problem emerges: whether a given high-level program can be realized on a given low-level datapath. A good understanding of this problem can benefit both the design of high-level SDN programming and the design of datapaths. Given a fixed datapath (*e.g.*, a fixed pipeline architecture such as OF-DPA), the vendor of the datapath can provide guidelines on the high-level programs that can be realized. Given a set of high-level programs to be supported, one could use this understanding to design the most compact datapath supporting these programs. Even for reconfigurable datapaths (*e.g.*, P4), as reconfiguration can be expensive and time consuming, one can use this understanding to guide the design of a more robust datapath. Considering all high-level programs that can be realized onto a given programmable datapath as the capacity of the datapath, we define the basic problem as the *SDN datapath programming capacity problem*.

Solving the datapath capacity problem, however, is not trivial. Consider a simple datapath, named Simple-DP, shown in Fig. 1. It is among the simplest datapaths, consisting of three tables forming a pipeline, where the first table (`t1`) matches on source IP and may jump to one of the two following tables, which both match on destination IP.

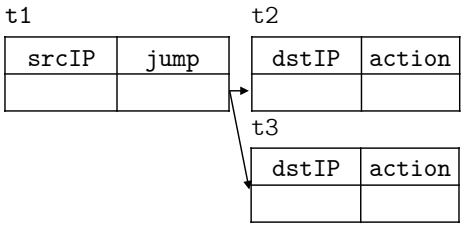


Fig. 1. A simple example datapath: Simple-DP.

Consider two simple high-level SDN programs below, both specified in the algorithmic, event-driven programming style to handle packet misses; see Sec. II for more details on the programming model. An interested reader can try to verify that the first program can be realized by Simple-DP, but the second cannot.

```
//  Routing Function: secureL3Route
L0: secureL3Route(Addr srcIP, Addr dstIP):
L1:   if srcIP == 10.0.0.1:
L2:     return Forward(port=shortestPath(dstIP))
```

Christopher Leet and Xin Wang are co-first authors

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

```
L3:   else:
L4:     return Drop();

//  Program: twoHostL3Route
L0: def twoHostL3Route(Addr srcIP, Addr dstIP):
L1:   if srcIP == 10.0.0.1:
L2:     return Forward(port=shortestPath(dstIP))
L3:   elif srcIP == 10.0.0.2:
L4:     return Forward(port=securePath(dstIP))
L5:   else:
L6:     return Drop();
```

Although the preceding datapath and high-level programs are among the simplest, they may already appear to be non-trivial for a reader to analyze. General datapath and high-level programs can be much more complex as multiple services need to be implemented and hence they can pose severe challenges in analysis. The goal of this paper is to develop the first systematic methodology to solve the SDN datapath programming capacity problem.

The contributions of this paper can be summarized as follows. First, we propose a unifying characteristic functional space to unify and extract the essence of programs and pipelines, removing complexities such as program structures and pipeline layouts. Second, we define a comparator in this functional space, which can be used to check whether a high-level program can be realized on a given pipeline.

The rest of the paper is organized as follows. We define our model precisely in Sec. II. The main results are given in Sec. III. Sec. IV shows our evaluation results. Finally, related work is provided in Sec. V.

II. Models

We start by specifying the high-level SDN programs and low-level datapath models. Since the main focus of SDN is routing, we refer to a high-level SDN program as a routing function. Since multi-table pipelines are the state-of-art for SDN datapaths, we focus on pipelines as datapaths.

A. Routing Function Model

Routing function: We denote a routing function as f, and assume that it is a logically centralized, deterministic function written in a high level language logically executed by an SDN controller on every packet [7] entering that controller's network to determine network-wide routing for that packet.

Each execution of f on a packet reads a set of the packet's attributes (called match fields) $\mathcal{M} = \langle m_1, ..., m_n \rangle$ (*e.g.*, `<srcIP, dstIP, ...>`). We use M to denote a subset of packet match fields included in $\mathcal{M}$. Moreover, we denote $dom(M)$ as the domain of a set of match fields M. The execution of f returns a routing action from a set of valid actions $\mathcal{R}$ (*e.g.*, `Drop, Forward(port=2)`):

$$f : dom(\mathcal{M}) \rightarrow \mathcal{R}.$$

The space of such functions is denoted $\mathcal{F}$.

Example: We use the routing function `onPkt` below to illustrate key features of our routing function model.

```
\\  Routing function: onPkt
    Map hostTbl[key: dstIP, value: switch]
    Map condTbl[key: (dstIP, port), value: cond]
    Map routeTbl[key: (switch, cond), value: outPort]
L0: onPkt(Type ethType, Addr srcIP, Port srcPort, \
          Addr dstIP, Port dstPort):
L1: if (ethType != IPv4):
L2:   return Drop()
L3: if (verify(srcPort, srcIP)):
L4:   dstCond = condTbl[dstIP, dstPort]
L5:   dstSw = hostTbl[dstIP]
L6:   return Forward(port = routeTbl[dstCond, dstSw])
L7: return Drop()
```

Specifically, `onPkt` reads the match fields $\mathcal{M} = \langle \texttt{ethType}, \texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}, \texttt{dstPort} \rangle$ and maps each value in the domain of $\mathcal{M}$ to a routing action in $\mathcal{R} = \{\texttt{Drop()}, \texttt{Forward(port=x)}\}$. While we write `onPkt` as an imperative function, we emphasize that our model is fully generic and does not specify a programming paradigm.

Elaborating, `onPkt`'s first three lines declare key-value tables. Specifically, `hostTable` and `condTable` associate each IP address with an attachment switch and host condition (*e.g.*, authentication status) respectively, while `routeTable` maps a switch, condition pair to its forwarding port. Moving on to `onPkt`'s body, `L1` and `L2` detect and drop non-IPv4 traffic, while `L7` drops traffic from unverified endpoints. For verified packets, `L4` to `L6` further set `dstCond` and `dstSw` variables, and then return a routing action from `routeTbl` based on the two variables.

Routing function DFG: Since a generic routing function can have arbitrary, complex control structure, we transform a routing function into a dataflow graph (DFG) to better represent its structure. We denote an f's DFG as G_f.

Specifically, to compute G_f for f, we must remove all of f control follow dependencies. These dependencies are removed by the following transformations:

- We remove assignment statement order dependencies by converting f to single static assignment form (SSA).
- We remove branches by assigning their conditionals' values to guards, and appending dependencies on these guards to all statements in their `if` and `else` blocks.
- We remove program loops by converting them to black box functions which read all variables read by the loop and write all variables written by them.

For example, our example routing function `onPkt` is transformed as follows:

```
L0: onPkt(...):
L1: g0 = (ethType != IPv4)
L2: if  g0: return Drop()
L3: g1 = verify(srcPort, srcIP)):
L4: if  g1: dstCond = condTbl[dstIP, dstPort]
L5: if  g1: dstSw = hostTbl[dstIP]
L6: if  g1: return Forward(port = routeTbl[...])
L7: if !g1: return Drop()
```

Note that `onPkt`'s `if` statement at `L1`'s has been replaced by an assignment from its conditional to the guard `g0`. This

guard is appended to L2, which was formally in the if statement's if block.

Given this transformation, we define G_f for f:

Definition 1. *A routing function f's' dataflow graph DFG $G_f = (V_f, E_f)$ is a vertex weighted dag generated from a transformed f such that:*

- *Each vertex v_f in V_f is a variable in f.*
- *A v_f's weight is its domain size.*
- *There is a directed edge in E_f between two variables if the source variable appears in the target variable's assignment.*

As an example, we give onPkt's DFG below:

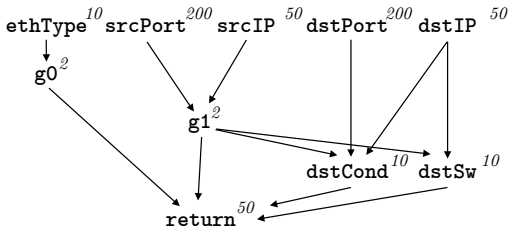


Fig. 2. The routing function onPkt's DFG $G_{\texttt{onPkt}}$.

Observe that the vertex dstSw is descended from the two variables in its assignment, g1 and dstIP. The vertex's weight, 10, indicates dstSw's domain.

B. Pipeline Model

We focus on state-of-the-art datapaths: multi-table pipelines. We first model a table t in a pipeline p and then we give a clear definition for the pipeline.

Pipeline table: Each pipeline table $t \in p$ is a exact match match-action table. Each of t's actions is a routing action output, or a write to t's output register $r(t)$ followed by a hop to a subsequent table in p, or a simple jump action to a subsequent table in p. Not all t output routing actions, and we denote the t that do as an egress table.

Each t matches on a set of inputs $I(t)$ that contains packet match fields $m_i \in \mathcal{M}$ and preceding tables' output registers $r(t)$. Key limitations on a t are the maximum number of rules it can contain and $r(t)$'s bit length, which we denote $maxrules(t)$ and $bits(r(t))$ respectively.

Pipeline: A pipeline p is a singly rooted dag (directed acyclic graph) of tables $\{t_i\}$. An edge (t_i, t_j) in a p indicates that a packet arriving at t_i can jump to t_j.

Each packet passing through p starts at p's root and proceeds through p to an egress table. Therefore, each packet passing through p can map to a path in the p, along with a routing action for that packet from $\mathcal{R}$.

A packet's path through p and the action its egress table outputs are determined by the set of packet match fields $\mathcal{M}$ each $t_i \in p$ matches on. Given this, p may also be summarized as a mapping from $dom(\mathcal{M})$ to $\mathcal{R}$, which depends on p's contents.

We denote the space of all pipelines p as P.

Symbol	Definition
Routing function symbols	
f	Routing function
$\mathcal{F}$	Routing function space
m_i	Packet match field
$\mathcal{M}$	Set of $\forall$ m_i
$dom(M)$	Domain of valid values of M
$\mathcal{R}$	Set of $\forall$ valid routing actions
Pipeline symbols	
p	Pipeline
$\mathcal{P}$	Pipeline function space
t_i	Pipeline table
$r(t_i)$	t_i's output register
$bits(r(t_i))$	t_i's output register bit length
$I(t_i)$	t_i's table inputs
$maxrules(t_i)$	Maximum # of rules t_i can contain

TABLE I
SYMBOL TABLE LISTING NOTATION IN OUR MAIN RESULTS.

Example: We now give an example pipeline ExampleDP, shown in Fig 3, to illustrate our pipeline model. Note that in the example, a table matches on fields on its left-hand side, writes to a register on its right-hand side, and the field output of a table indicates the table contains output routing actions.

Narrowing our focus, consider t2 ∈ ExampleDP. t2 is an exact match table whose inputs I(t2) are srcIP and srcPort, and whose output register is r(t2).

Significant computation limits on t2 are its maximum number of rules $maxrules$(t2)) and the size of its output register $bits$(r(t2)).

III. Main Results

Given the function and pipeline models, we now present our main results, on whether a function f can be realized by a pipeline p.

To simplify the reading of our results, we put only the definitions and main results in the main text. The proofs of the results are in the appendix. To make it easier to follow the symbols, we collect key symbols in Table I for reference.

A. Overview

A main challenge in developing a systemic method to verify whether a routing function f can be realized by a pipeline p, which we denote as $f \Rightarrow p$, is that routing functions and pipelines are represented differently and both types of representations can have substantial complexities and variations. Consider each routing function f as a point in a functional space $\mathcal{F}$, and each pipeline p as a point in functional space $\mathcal{P}$.

Our main contribution is the introduction of a novel, unifying, normalization functional space $\mathcal{C}$ called the characteristic functions space. Each routing function f is mapped by the mapping τ to a characteristic function $\tau(f) \in \mathcal{C}$, characterizing the computational load of f. Each pipeline p, on the

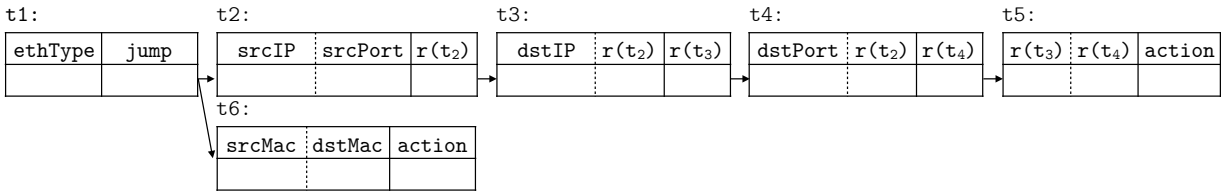


Fig. 3. Example datapath, `ExampleDP`

other hand, is mapped to a set $\kappa(p) \subset \mathcal{C}$ of characteristic functions, representing the set of computational capabilities of the pipeline. Fig 4 illustrates the mapping structure.

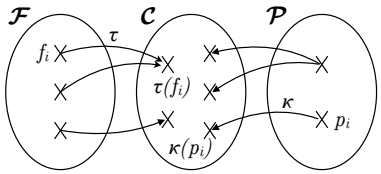


Fig. 4. The spaces $\mathcal{F}$, $\mathcal{P}$ and $\mathcal{C}$ and the mappings between them.

Since $\tau(f)$ and $\kappa(p)$ are defined in the the same space $\mathcal{C}$, as a point and as a set of points respectively, one can compare $\tau(f)$ with each element in $\kappa(p)$, to see if the load can be "covered" by a capability, resulting in our basic capacity theorem: that if $\exists\ c \in \kappa(p) \geqslant \tau(f), f \Rightarrow p$.

B. Characteristic Functions

We begin by defining a generic characteristic function c.

Definition 2. *A* characteristic function c *is a mapping from each subset M of a packet's match fields to a vector consisting of two components:*

$$c(M) \triangleq < scope(M),\ ec(M) > .$$

We refer to the two components of $c(M)$'s vector as $c(M)[scope]$ and $c(M)[ec]$ respectively.

Given two characteristic functions, one can compare them.

Definition 3. *We define c_i* dominates *c_j, denoted as $c_i \geqslant c_j$ as follows:*

$$c_i \geqslant c_j \triangleq\ \forall\ n \in \{scope, ec\},$$
$$\forall\ M \in 2^{\mathcal{M}},\ c_i(M)[n] \geqslant c_j(M)[n].$$

To verify our capacity theorem, we need to compare a set of characteristic functions with a single characteristic function.

Definition 4. *A set of characteristic functions C_i* dominates *a characteristic function c_j, denoted as $C_i \trianglerighteq c_j$, if a $c_i \in C_i$ dominates c_j:*

$$C_i \trianglerighteq c_j \triangleq \exists\ c_i \in C_i : c_i \geqslant c_j.$$

C. Characterization of a Routing Function

Given the concept of characteristic functions, we now derive the characteristic function, denoted as $\tau(f)$, of a routing function f.

Definition 5. *The* scope *of the characteristic function of a routing function for a subset of packet match fields M is the size of the domain of valid values of M:*

$$\tau(f)(M)[scope] \triangleq dom(M)$$

$\tau(f)(M)[ec]$ is a property that we build from the concept of f-equivalence:

Definition 6. *We define* f-equivalence*, denoted as $\sim_f$, as a relationship between two values of M, which we write as $v_i(M)$ and $v_j(M)$, which denotes that these values cannot be distinguished by f:*

$$v_i(M) \sim_f v_j(M) \triangleq \forall\ v_k(\mathcal{M} - M) \in dom(\mathcal{M} - M),$$
$$f(v_i(M), v_k(\mathcal{M} - M)) = f(v_j(M), v_k(\mathcal{M} - M)).$$

Our definition of f-equivalence leads naturally to our definition of an f-equivalence class.

Definition 7. *An* f-equivalence class*, denoted as $[v_i(M)]_f$, is the set of all values f-equivalent to a given M's value $v_i(M)$:*

$$[v_i(M)]_f \triangleq \{v_j(M) \in dom(M) : v_i(M) \sim_f v_j(M)\}.$$

Counting equivalence classes gives us the concept of f-equivalence class number.

Definition 8. *The* f-equivalence class number *of M, denoted as $|dom(M)/\ \sim_f\ |$, is the cardinality of M's set of f-equivalence classes.*

We now arrive at our definition of $\tau(f)(M)[ec]$.

Definition 9. *The* ec *of a routing function's characteristic function for an M is the cardinality of M's set of f-equivalence classes:*

$$\tau(f)(M)[ec] \triangleq |dom(M)/\sim_f|$$

Definition 10. The characteristic function $\tau(f)$ of a routing function *characterizes f's computational load:*

$$\tau(f)(M) \triangleq (dom(M),\ |dom(M)/\sim_f|).$$

While $\tau(f)$ is powerful it is impractical because f-equivalence class number is costly to directly calculate. We therefore bound a $\tau(f)$ by defining the bounding characteristic function of a routing function $\tau_G(f)$ which is easily derivable from f's DFG. This function characterizes an upper bound on f's computation load: $\tau_G(f)$ dominates $\tau(f)$.

We find $\tau_G(f)[scope]$ as before. Instead of calculating $\tau_G(f)[ec]$, however, we determine an upper bound for with

the value of specific vertex cut in G_f, f's DFG. We now construct this cut.

Definition 11. *Let $V_f(M)$ be the vertices of $m_i \in M$ in G_f, and $D_f(M)$ be the vertices in G_f descended from $V_f(M)$.*

The vertex-min-cut of M, $G_f.vertexMinCut(M)$, *is the product of the weights of the vertices in the minimum weight vertex cut severing $V_f(M)$ from $D_f(\mathcal{M} - M)$.*

Given this cut, we define $\tau_G(f)$ follows:

Definition 12. The characteristic function $\tau_G(f)$ of a routing function *characterizes an upper bound on f's computational load; $\tau_G(f)$ dominates $\tau(f)$:*

$$\tau_G(f)(M) \triangleq (dom(M),\ G_f.vertexMinCut(M)).$$

Example: We illustrate these concepts with our example routing function `onPkt`.

Consider `onPkt`'s match fields `srcIP` and `srcPort`. Each are only read once: on `L3`, by the boolean function `isVerified`. Thus, while `srcIP` and `srcPort` may have many f-equivalence classes individually, (`srcIP`, `srcPort`) only has two: values that `isVerified` evaluates to 0, and values it evaluates as 1.

Suppose `onPkt` is a routing function for a small commercial network fronted by a NAT with 50 hosts each running a limited set of applications that only use 200 standard ports. Given this, $dom(\texttt{srcIP}, \texttt{srcPort}) = 10000$, and thus $\tau(\texttt{onPkt})(\texttt{srcIP}, \texttt{srcPort}) = (10000, 2)$.

While the equivalence class number of (`srcIP`, `srcPort`) was straightforwards, the equivalence class number of most other subsets of `onPkt`'s inputs is not so obvious. We therefore bound $\tau(\texttt{onPkt})$ with $\tau_G(\texttt{onPkt})$, which we calculate using `onPkt`'s DFG $G_{\texttt{onPkt}}$, shown in Fig. 5.

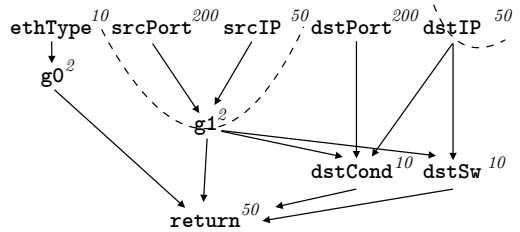


Fig. 5. The routing function `onPkt`'s DFG $G_{\texttt{onPkt}}$ and the cut (`srcIP`, `srcPort`, `dstPort`).

To bound, for example, the equivalence class number of `onPkt`'s inputs (`srcIP`, `srcPort`, `dstIP`) we take the vertex-min-cut in $G_{\texttt{onPkt}}$ between their vertices and every vertex descended from `onPkt`'s other inputs: (`ethType`, `dstPort`,`g0`, `dstCond`, `return`). This vertex-min-cut is indicated in Fig. 5 by a dotted line.

The vertices in this cut, (`g1`, `dstIP`) have weight 50 and 2, and thus $\tau_G(\texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}) = (50000,\ 100)$.

D. Characterization of a Pipeline

We now define $\kappa(p)$, the set of characteristic functions of a pipeline p. We start by defining a path ρ through a pipeline p.

Definition 13. *A path, ρ, in a p is a path through p's dag $\langle t_1, ..., t_n \rangle$ such that t_1 is a root table and t_n an egress table in the p.*

As an example, `ExampleDP` contains two paths: $\langle$`t1,t2,t3,t4,t5`$\rangle$, and $\langle$`t1,t6`$\rangle$, which we denote as ρ_{L2} and ρ_{L3} respectively.

We define, $\forall\ \rho \in p,\ \kappa_\rho(p)$ as the characteristic function of the a path through a pipeline.

Definition 14. The characteristic function set $\kappa(p)$ *of a p is the union of $\forall\ \rho \in p$'s characteristic functions:*

$$\kappa(p)(M) \triangleq \{c \in \mathcal{C} : c = \kappa_\rho(\rho)\ \forall\ \rho \in p\}.$$

We now construct the characteristic function of a path ρ by introducing the following definitions:

Definition 15. *The* input closure $\bar{M}_\rho(t_i)$ *of a table $t_i \in \rho$ is the set of inputs that t_i can obtain information about:*

$$\bar{M}_\rho(t_i) \triangleq \{m_i \in \mathcal{M} : m_i \in I(t_i)\ \vee \\ m_i \in \bar{M}_\rho(t_j)\ s.t.\ r(t_j) \in I(t_i)\}.$$

Definition 16. *The* closure set, $\bar{\bar{M}}_\rho(M)$ *of a ρ's M is the set of $t_i \in \rho$ with input closure M.*

$$\bar{\bar{M}}_\rho(M) \triangleq \{t_i \in \rho : \bar{M}_\rho(t_i) = M\}.$$

Using these definitions, we define the characteristic function of a ρ as:

Definition 17. The characteristic function $\kappa_\rho(\rho)$ of a ρ *characterizes the computational capacity of a ρ.*

$\kappa_\rho(\rho)[scope]$ is the maximum number of values of M that ρ can read and $\kappa_\rho(\rho)[ec]$ is the maximum number of equivalence classes of M that ρ can distinguish.

$$\kappa_\rho(\rho)(M) \triangleq$$
$$\begin{cases} \bar{\bar{M}}_\rho(M) \neq \varnothing & (min[maxrules(t_i) : t_i \in \bar{\bar{M}}_\rho(M)], \\ & \quad min[2^{bits(r(t_i))} : t_i \in \bar{\bar{M}}_\rho(M)]) \\ \bar{\bar{M}}_\rho(M) = \varnothing \wedge \exists\ m_i \in M : & \\ m_i \notin \bigcup_{t_i \in \rho} \bar{M}(t_i, \rho) & (1, 1) \\ otherwise, & (\texttt{T}, \texttt{T}). \end{cases}$$

Example: As before, we provide intuition into the characteristic functions of pipelines using our example pipeline `ExampleDP`.

Recall from our model that `ExampleDP` contains two ρ: $\rho_{\texttt{L2}}$ and $\rho_{\texttt{L3}}$. Consider the table `t4`, only contained by $\rho_{\texttt{L3}}$. The input closure $\bar{M}_{\rho_{\texttt{L3}}}(\texttt{t4})$ is (`srcIP`, `srcPort`, `dstIP`) since `t4` reads `dstIP` and `r(t2)`, and `t2` in turn reads `srcIP` and `srcPort`. The closure set, $\bar{\bar{M}}_{\rho_{\texttt{L3}}}(\texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP})$, of `t4`'s inputs in $\rho_{\texttt{L3}}$ is {`t4`}: `t4`'s input closure is unique.

Thus, $\kappa_{\rho_{\texttt{L3}}}(\bar{M}_{\rho_{\texttt{L3}}}) = \kappa_{\rho_{\texttt{L3}}}(\texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}) = (maxRules(\texttt{t4}),\ 2^{bits(r(\texttt{t4}))})$. In the case that `t4` has 2^{20} rules and a 16 bit output register, $\kappa_{\rho_{\texttt{L3}}}(\bar{M}_{\rho_{\texttt{L3}}}) = (2^{20}, 2^{16})$.

Further, consider the subset of ExampleDP's match fields (srcMac, dstMac). ρ_{L3} does not contain the inputs srcMac or dstMac and thus it can only realize functions that contain them in the unlikely event that all are constants. Constants have domain 1 and 1 equivalence class. Thus the value of $\kappa_{\rho_{\text{L3}}}$ for any set of outputs containing srcMac is $(1, 1)$.

Finally, consider the subset of ExampleDP's match fields (srcIP, srcPort). srcIP and srcPort are both read by ρ_{L3}, but (srcIP, srcPort) is not an input closure of any $t_i \in \rho_{\text{L3}}$. In this case, it is not necessary to consider (srcIP, srcPort) to verify realizability, and thus $\kappa_{\rho_{\text{L3}}}(\texttt{srcIP}, \texttt{srcPort}) = (\mathsf{T}, \mathsf{T})$, indicating that we can skip this field during comparison with a routing funciton's τ.

E. Datapath Programming Capacity Theorems

Combining the preceding definitions to characterize both routing functions and pipelines, we finally arrive at our central result: a sufficient condition for whether a given f can be realized in a given p.

Theorem 1 (Pipeline Realization Theorem)**.** *A routing function f can be realized by a pipeline p if $\kappa(p)$, the set of characteristic functions of p dominates $\tau(f)$, the characteristic function of f. Formally, we have:*

$$\kappa(p) \trianglerighteq \tau(f) \Rightarrow f \Rightarrow p.$$

As a corollary, because $\tau_G(f) > \tau(f)$, the Pipeline Realization Theorem extends to $\tau_G(f)$.

Example: We illustrate our Pipeline Realization Theorem using onPkt and ExampleDP. Specifically, our Pipeline Realization Theorem states that $\kappa(\texttt{ExampleDP}) \trianglerighteq \tau(\texttt{onPkt}) \Rightarrow \texttt{ExampleDP} \Rightarrow \texttt{onPkt}$.

Further, $\kappa(\texttt{ExampleDP}) \trianglerighteq \tau(\texttt{onPkt})$ is true if $\kappa_\rho(\rho_{\text{L2}}) > \tau_G(\texttt{onPkt})$ or $\kappa_\rho(\rho_{\text{L3}}) > \tau_G(\texttt{onPkt})$ We verify each conditional by comparing each component of each vector given by each pair of characteristic functions. For example, $\tau(\texttt{onPkt})(\texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}) = (50000, 100)$, $\kappa_\rho(\rho_{\text{L3}})(\texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}) = (2^{20}, 2^{16})$, and thus the input set (srcIP, srcPort, dstIP) does not prevent onPkt from being realized in ρ_{L3}.

Tightness: Though the theorem provides only a sufficient condition, tighter results, in particular sufficient and necessary conditions, can be established in multiple settings. In particular, we have the following result:

Definition 18. *A branchless pipeline p is a p whose dag is a path from its root to its output node.*

Theorem 2. *If p is a branchless pipeline, p's table size is large, and each match field $m_i \in \mathcal{M}$ appears in exactly one of p's tables, $\kappa(p) \trianglerighteq \tau_G(f) \Leftrightarrow f \Rightarrow p$.*

In Sec. IV-A, we evaluate our realization theorem's tightness on more general pipelines through experiments.

IV. Evaluation

We now evaluate the tightness, time complexity and output size of routing function and pipeline characterization numerically. All experiments are conducted on a 1.6 GHz Intel Core i5 with 4 GB RAM.

A. Routing Function Characterization Tightness

We demonstrate the tightness of our characterization of a routing function by comparing the number of equivalence classes of a M for both τ and τ_G for the following set of simple routing functions:

```
\\  Routing function: simpleRoute
L0: def simpleRoute(Addr srcIP, Addr dstIP):
L1:   srcSw = hostTbl[srcIP]
L2:   dstSw = hostTbl[dstIP]
L3:   route = routeTbl[srcSw, dstSw]
L4:   return route
```

Our first function, simpleRoute maps a packet's srcIP and dstIP to their host packet switches dstSw and srcSw and then looks up the route between them.

```
//  Routing Function: condRoute
L0: condRoute(srcIP, dstIP):
L1:   srcSw = hostTbl[srcIP]
L2:   dstSw = hostTbl[dstIP]
L3:   routeCond = condTbl[srcIP, dstIP]
L4:   route = routeTbl[srcSw, dstSw, routeCond]
L5:   return route
```

Our second function condRoute extends simpleRoute by introducing a route condition variable which modulates the function's route look up.

```
//  Routing Function: secureRoute
L0: secureRoute(Addr srcIP, Addr dstIP):
L1:   if (isFiltered(srcIP)):
L2:     return Drop()
L3:   else:
L4:     route = fwdTbl[dstIP]
L5:     return route
```

Our final function secureRoute drops all traffic from srcIPs on a filter list and forwards remaining traffic.

Results: We present our results in Table II. Specifically, in Table II, column 2 defines the domain of srcIP, dstIP, columns 3-6 give the output ranges $O(tbls)$ of each table, and columns 7-10 give the values of selected fields in each function's $\tau(f)$ and $\tau_G(f)$.

Note that the row b1(scR) and b2(scR) represent the two branches of secureRoute. We record N/A when a value is not applicable to a given function, and null for values of $\tau(f)$ where computation failed to halt.

In our evaluation of simpleRoute and condRoute, the results of τ and τ_G are almost identical in every case barring an extreme one where $O(\texttt{condTbl}) = 1$. Notably, our values of $\tau(f)$ and $\tau_G(f)$ are not influenced by the range of $routeTbl$ $O(\texttt{routeTbl})$. This implies there is no pattern between the allocation of routes to (srcIP, dstIP) pairs $\tau(f)$ can exploit to reduce its number of equivalence classes.

Further notice that our functions with control statements: `secureRoute` and `onPkt` have a large gap between τ and τ_G, suggesting τ_G's bound is loose on heavily branching programs. However, as rows b1(scR) and b2(scR) show, we can ameliorate this problem by calculating each route through a program characteristic function separately.

B. Routing Function Characterization Time Complexity

We now compare the time required to compute τ and τ_G for given routing functions. We run our tests using `simpleRoute` where $O($`hostTbl`$)$ = 100 and $O($`routeTbl`$)$ = 30.

Results: Fig. 6 shows the scalability of τ_G as input length grows. As `srcIP` and `dstIP`) bit length increase, τ_G's computation time remains constant while τ's computation time grows rapidly.

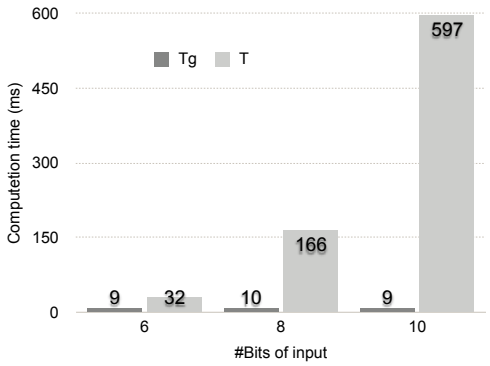


Fig. 6. Computation time required to generate $\tau($`simpleRoute`$)$ and $\tau_G($`simpleRoute`$)$ as input bit length varies.

C. Characterization of a Pipeline

We now examine the memory utilization and computation time of pipeline characterization. We evaluate the following pipelines:

1) **The OF-DPA Abstract Switch 2.0:** The OpenFlow Data Plane Abstraction Abstract Switch 2.0 (OF-DPA) is an abstract switch model based on the Open Flow 1.3.4 protocol designed to allow the programming of Broadcom-based switches under the OpenFlow protocol. We examine two OF-DPA flow table configurations: bridging and routing (BR), and data center overlay tunnel (OT), which contain 7, and 3 tables in 5, 3 stages respectively. [2]
2) **PicOS:** PicOS is a network operating system for white box switches that provides OF programmability across HP, Edgecore and Pica switches. We examine two fixed pipelines offered by PicOS as table type patterns: PicOS's IP routing pipeline (IPR) and Policy routing pipeline (PR), which contain 4 and 5 tables in 4 and 5 stages respectively. [8]

Results: Table III our characterization results for our evaluated pipelines. The results show that despite the theoretically large number of subsets of $\mathcal{M}$ across evaluated pipelines, memory utilization and computation time are small.

V. Related Work

High level SDN Program Compilers: Multiple systems that allow programmers to write SDN programs in high level languages and then compile such programs to flow table pipelines have been proposed over the last several years. Such systems are related to our work in that they examine the transformation of policy programs into switch flow tables. We group these systems into two categories: *tier-less* and *split-level*.

Tier-less systems (*e.g.* SNAP [9], FML [10], FlowLog [11], Maple [7]), require programmers to specify forwarding behaviors as packet handling functions which are then used by the SDN controller to configure and update network state. Such systems pioneer our pipeline capacity theorem's notion of a *program function* and are able to compile such functions to single pipelines. These systems, however, are unable to verify that submitted functions can be written to a given pipeline without physically carrying out the time consuming process of compilation, and cannot write programs to multi-pipeline networks.

Split-level systems such as the Frentic family (*e.g.* Frenetic [6], Pyretic [12]) provide a two tiered programming model in which controller programs specify events of interest and then respond to these events when they occur by calculating new network policies. Again, such systems cannot verify that a given controller program's output can be written to the controller's switches' pipelines, although this paradigm falls outside of our pipeline capacity theorem's model as well.

Pipeline specification languages: There are some superficial similarities between pipeline specification languages (*e.g.* P4 [3], PISCES [13], Concurrent NetCore [14]) and our pipeline capacity theorem, such as the analysis and guarantees that such languages provide about pipeline behavior. For example, Concurrent NetCore's type system ensures that any program used to populate a pipeline has certain properties, such as determinism, whilst PISCES's switch specification allows compilers to analyze pipelines and optimize their performance. We contend, however, that our capacity theorem attacks an entirely different space in pipeline analysis - guaranteeing pipeline properties or improving performance is qualitatively different to verifying whether compilation is possible.

Pipeline design: Pipeline design schemes such as Jose et al.'s "Compiling Packet Programs to Reconfigurable Switches" [15], Sun et al.'s "Software-Defined Flow Table Pipeline" [16], FlowAdapter [17], and Domino [4] are clearly related to our pipeline capacity theorem in that they examine pipeline layout design under hardware constraints. Jose et al., Sun et al., and FlowAdapter however, focus on mapping logical lookup tables/flow table pipelines to physical tables whilst our pipeline capacity theorem focuses on generic programs, while Domino considers weaker hardware constraints (*e.g.* limits on stateful operations at line-rate) than our work does.

f	$bits(IP)$	O(hostTbl)	O(routeTbl)	O(condTbl)	O(fwdTbl)	τ(f)(srcIP)	τ(f)(dstIP)	τ_G(f)(srcIP)	τ_G(f)(dstIP)
smplR	10	100	2	N/A	N/A	100	100	100	100
smplR	10	100	30	N/A	N/A	100	100	100	100
smplR	10	100	5000	N/A	N/A	100	100	100	100
smplR	12	100	30	N/A	N/A	100	100	100	100
smplR	10	200	30	N/A	N/A	200	200	200	200
condR	10	100	30	50	N/A	1024	1024	1024	1024
condR	10	100	30	5	N/A	1024	1024	1024	1024
condR	10	100	30	1	N/A	100	100	1024	1024
scR	10	N/A	N/A	N/A	100	2	100	2	1024
b1(scR)	10	N/A	N/A	N/A	N/A	1	N/A	1	N/A
b2(scR)	10	N/A	N/A	N/A	100	1	100	1	100
onPkt	32	100	30	50	N/A	null	null	2^{32}	2^{32}

TABLE II
CHARACTERIZATION RESULTS OF ROUTING FUNCTIONS WITH DIFFERENT STATISTICS.

Pipeline	#Paths	Time (ms)	#Valid M	#M
ExampleDP	3	8	6	22
Broadcom BR	4	13	19	$3 * (2^{24}) + 2^7$
Broadcom OT	2	7	5	$2^{24} + 16$
PicOS IPR	1	7	4	2^7
PicOd PR	3	9	14	$2 * (2^{24}) + 2^7$

TABLE III
CHARACTERIZATION RESULTS OF PIPELINES.

VI. ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments. Shenshen Chen helped with initial discussions. The research was supported in part by NSFC grants NSFC #61672385, NSFC #61702373; Shanghai Key Project Grant #16511100900; NSF grants CC-IIE 1440745, CCF-1637385 and CCF-1650596; Google Research Award; and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

[1] "OpenFlow Switch Specification Version 1.3.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf, ONF.

[2] (2014) OpenFlow Data Plane Abstraction (OF-DPA): Abstract Switch Specification Version 2.0. Broadcom. [Online]. Available: www.broadcom.com

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[4] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 15–28.

[5] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[6] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291. [Online]. Available: http://doi.acm.org/10.1145/2034773.2034812

[7] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. ACM, 2013, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486030

[8] (2015) Scaling up SDNs using TTPs (Table Type Patterns). Pica 8. [Online]. Available: www.pica8.com

[9] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934892

[10] T. Hinrichs, J. Mitchell, N. Gude, S. Shenker, and M. Casado, "Practical declarative network management," in *in ACM Workshop: Research on Enterprise Networking*, 2009.

[11] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi, "Tierless programming and reasoning for software-defined networks," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 519–531. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616496

[12] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482629

[13] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 525–538.

[14] C. Schlesinger, M. Greenberg, and D. Walker, "Concurrent netcore: From policies to pipelines," in *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '14. New York, NY, USA: ACM, 2014, pp. 11–24. [Online]. Available: http://doi.acm.org/10.1145/2628136.2628157

[15] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 103–115. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/jose

[16] X. Sun, T. E. Ng, and G. Wang, "Software-Defined Flow Table Pipeline," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 2015, pp. 335–340.

[17] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, and G. Xie, "The FlowAdapter: Enable flexible multi-table processing on legacy hardware," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 85–90.

APPENDIX

We now present proofs to verify our main results. The structure of these proofs will be as follows. First, we define a mechanism to encode sufficient information about a given

$M \in \mathcal{M}$ to fully execute a given f. Second, we show that a pipeline transmitting information internally using our encoding can realize an f in p given that $\kappa(p) \trianglerighteq \tau_G(f)$. Finally, we show that $\tau_G(f) > \tau(f)$, proving by extension that if $\kappa(p) \trianglerighteq \tau(f), f \Rightarrow p$. We omit a proof of Theorem 2 and the proofs of certain corollaries and lemmas due to space constraints. We will give these proofs in an extended report in an upcoming technical journal.

We base our summary on the vertices in the $G_f.vertexMinCut(M)$ of a f's G_f.

Definition 19. *The* min cut vertices $\mu_f(M)$ *are the vertices in an f's G_f cut by $G_f.vertexMinCut(M)$.*

Let a given value of $\mu_f(M)$ be $v_i(\mu_f(M))$ and the domain of values of $\mu_f(M)$ be $dom(\mu_f(M))$.

Lemma 1. *Given a $G_f.vertexMinCut(M)$, we can calculate f without knowing $v_i(M)$ given $v_i(\mu_f(M))$.*

While $\mu_f(M)$ acts as an effective representation of values of M $v_i(M)$, we can compress it by introducing the concept of codewords, allowing us to maximize transmission through a pipeline.

Definition 20. *The* codewords $\chi_f(M)$ *of inputs M of a f are a set of integers that correspond to the f-equivalence classes of M.*

Receiving a codeword $\in \chi_f(M)$ is equivalent to receiving a value for M $v_i(M)$, since the codeword can be deterministically mapped back into a value from $v_i(M)$'s equivalence class. We now define the shorthand '*compute the codewords of M*' which we will use in our proofs:

Definition 21. *If we can* compute the codewords $\chi_f(M)$ *of M, $\forall\ v_i(M) \in dom(M)$ we can compute the codeword associated with the equivalence class of $v_i(M)$.*

Our codewords give us a bound on the transmission requirements of an M, given in Lemma 2.

Lemma 2. *A table t_i only requires $log_2(\lceil |dom(\mu_f(M))| - 1 \rceil)$ bits of information about M to execute f correctly.*

Proof. We can encode the value of any $v_i(M) \in dom(M)$ as a codeword in $\chi_f(M)$ and still convey sufficient information to compute f. If $\mu_f(M)$ can take $|dom(\mu_f(M))|$ distinct values, we can assign each value a unique codeword from the set [0, ..., $|dom(\mu_f(M))| - 1$], which take at most $log_2(\lceil |dom(\mu_f(M))| - 1 \rceil)$ bits to represent. □

Proving the realization theorem: Given our characterization of function transmission requirements, we can now embark on our proof of our realization theorem. First, we will give our key underlying lemma, lemma 3, from which our realization theorem follows naturally.

Lemma 3. *If $\forall\ t_i \in \rho = \langle t_1, ..., t_n \rangle$ have $maxRules(t_i) > \tau_G(f)(\bar{M}_\rho(t_i))[dom]$, and $2^{r(t_i)} > \tau_G(f)(\bar{M}_\rho(t_i))[ec]$, then $\forall t_i \in \rho = \langle t_1, ..., t_n \rangle$ can output $\chi_f(\bar{M}_\rho(t_i))$ to $r(t_i)$.*

Given Lemma 3, we are now equipped to prove the realization theorem.

Proof. Given an f and p, we will prove that if $\kappa(p) \trianglerighteq tau(f)$, $f \Rightarrow p$. Consider a $\kappa_\rho(\rho) \in \kappa(p)$.

$\forall\ M \in \mathcal{M} : m_i \in M \rightarrow m_i \notin \bigcup_{t_i \in \rho} \bar{M}_\rho(t_i),\ \kappa_\rho(\rho)(M) = (1,\ 1)$. Therefore, if $\kappa_\rho(\rho) > \tau_G(f) \Rightarrow$ all m_i not read by ρ are treated as constants or not read at all by f, and thus f is effectively a mapping from $\bigcup_{t_i \in \rho} \bar{M}_\rho(t_i) \rightarrow \mathcal{R}$.

Further, given $\kappa_\rho(\rho) > \tau_G(f)\ \forall\ t_i \in \rho,\ maxRules(t_i) > \tau_G(f)(\bar{M}_\rho(t_i))[dom]$, and $2^{r(t_i)} > \tau_G(f)(\bar{M}_\rho(t_i))[ec]$, and thus by Lemma 3 t_n can calculate $\chi_f(\bar{M}_\rho(t_n))$.

Finally, consider that if a t_i can calculate $\chi_f(M_i)$, and an f is a mapping $dom(M_i) \rightarrow \mathcal{R}$, t_i can compute f's output $\forall\ v_j(M_i) \in dom(M_i)$ by mapping each codeword in $\chi_f(M_i)$ to the output of f it corresponds to.

Since t_n is ρ's only output, $\bar{M}_\rho(t_n) = \bigcup_{t_i \in \rho} \bar{M}_\rho(t_i)$. Thus, t_n can compute f's output. Further, since t_n is an egress table it can always pass this output back to the switch.

Therefore, if $\kappa_\rho(\rho) > \tau_G(f), f \Rightarrow \rho$. Since $\kappa_\rho(\rho) \in k(p)$ and $\rho \in p$, we have proved that if $\kappa(p) \trianglerighteq \tau_G(f)$, $f \Rightarrow p$. □

The last step required to prove our realization theorem is to show that $\tau_G(f) > \tau(f)$ and thus that $\kappa(p) \trianglerighteq \tau(f) \Rightarrow f \Rightarrow p$. The crux of this step is given in Lemma 4, below.

Lemma 4. *The number of equivalence classes of M is bounded by $dom(\mu_f(M))$.*

Proof. Suppose, by way of contradiction, $\exists\ (f,\ M) : |dom(M)/\ \sim_f\ | > dom(\mu_f(M)$. Each $v_i(M)$ in one of M's equivalence classes must generate a $v_i(\mu_f(M))$. By the pigeonhole principle, if M has more equivalence classes than $\mu_f(M)$, two values of M from different equivalence classes must generate the same value of $\mu_f(M)$. However, by Lemma 1, $\mu_f(M)$ contains sufficient information about M to fix f's outputs value, and thus these two bindings of M must be in the same equivalence class, which is a contradiction. □

Corollary 1. *The number of f-equivalence classes of any M is bounded by $G_f.vertexMinCut(M)$.*

Corollary 2. *The characteristic function $\tau_G(f)$ dominates the characteristic function $\tau(f)$.*

We have therefore proven our realization theorem: that $\kappa(p) \trianglerighteq \tau_G(f) \Rightarrow f \Rightarrow p$.

Prophet: Fast Accurate Model-based Throughput Prediction for Reactive Flow in DC Networks

Kai Gao*§, Jingxuan Zhang†‡, Y. Richard Yang†‡[1], Jun Bi*§[1]

* Institute of Network Science and Cyberspace, Tsinghua University
† Department of Computer Science, Yale University
‡ Department of Computer Science, Tongji University
§ Tsinghua National Laboratory for Information Science and Technology

***Abstract*—As modern network applications (*e.g.*, large data analytics) become more distributed and can conduct application-layer traffic adaptation, they demand better network visibility to better orchestrate their data flows. As a result, the ability to predict the available bandwidth for a set of flows has become a fundamental requirement of today's networking systems. While there are previous studies addressing the case of non-reactive flows, the prediction for *reactive flows*, *e.g.*, flows managed by TCP congestion control algorithms, still remains an open problem. In this paper, we identify three challenges in providing throughput prediction for reactive flows: *throughput dynamics*, *heterogeneous reactive control mechanisms*, and *source-constrained flows*. Based on a previous theoretical model, we introduce a novel learning-based prediction system with a key component named *fast factor learning* (FFL) model. We adopt novel techniques to overcome practical concerns such as scalability, convergence and unknown system parameters. A system, Prophet, is proposed leveraging the emerging technologies of *Software Defined Networking* (SDN) to realize the model. Evaluations demonstrate that our solution achieves significant accuracy in a wide range of settings.**

I. Introduction

The last decade has witnessed the rapid development of infrastructures for distributed applications such as public Cloud platforms [1]–[3]. For many of these distributed applications, such as Content Delivery Networks (CDN) [4] and large-scale data analytics systems [5], [6], knowing the network performance in advance helps them to make scheduling decisions for better performance. Hence, predicting network performance has become a fundamental functionality for today's high performance distributed applications.

While many existing studies [7]–[11] work on predicting network performance, they tend to focus on reserved resources and not consider the case of *best-effort, reactive* flows (*e.g.*, flows that are managed by TCP congestion control). Best-effort, reactive flows can have multiple benefits, including full resource utilization with fairness guarantees, and hence contribute a large portion of total traffic in many networks [12]. As a result, predicting network performance involving such flows is an important problem but remains open.

Despite the importance, the problem is difficult due to several challenges.

- First, *the prediction system must handle the dynamics of reactive flows*. A fundamental difference between predicting the throughput of flows with bandwidth reservations and reactive flows is that reactive flows are *adaptive*. In a system with reactive flows, throughput of existing flows will change when flows arrive and die. Thus, the available resources cannot be computed by simply observing current state.
- Second, *the prediction system must handle heterogeneous reactive mechanisms*. In particular, the widely used reactive flow control mechanism, TCP, is host-based congestion control, which allows different hosts to use different implementations. For example, Linux kernel has a kernel option `tcp_congestion_control` and a socket option `TCP_CONGESTION` to set congestion algorithms system-wide and for each flow [13]. Such heterogeneity complicates the throughput prediction of TCP flows [14], [15].
- Third, *the prediction system must handle source constraints*. Throughput of a reactive flow is constrained by not only network resources but also application-level factors such as flow preferences or data availability. Thus, bandwidth estimation methods operating at the transport layer, as proposed in many previous TCP designs [16]–[18], does not suffice the need.

To handle both dynamics and heterogeneous reactive mechanisms, we take advantage of a previously proposed unifying theoretical model for heterogeneous reactive mechanisms [19]–[21].

Although the model provides a solid starting point, it leaves two key issues unaddressed. First, the theoretical model has a key parameter which is unknown in advance. We call this parameter a *scaling factor* as discussed in Section III-A and refer to the issue of computing the scaling factor as the scaling-factor computation challenge. Second, the theoretical model does not include source constraints which are important in real settings but introduce substantial complexity, as the constraints of background flows are not even known.

In this paper, we solve the scaling-factor computation challenge by deriving a novel method called *Fast Factor Learning* (FFL). While a naive approach which requires a unique variable for each individual flow can introduce scalability and convergence issues in practice, FFL uses 1) a novel technique based on equivalence classes, to substantially reduce the num-

[1]Prof. Yang (yry@cs.yale.edu) and Prof. Bi (junbi@tsinghua.edu.cn) are the corresponding authors of this paper.

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

ber of variables, improving scalability, and 2) a novel method based on gradient descent using sampling guidance, to achieve accelerated convergence. To address the source constraint challenge, we use learning to infer source constraints and extend our design to compute accurate throughput prediction.

Our **main contributions** in this paper are three-fold:

- We formally define the problem of *predicting throughput for reactive flows with source constraints*. A simple solution is first proposed for the most basic setting and we introduce two novel approaches to extend the solution to more general settings. In particular, we have adopted a novel learning component based on historical traffic samples to provide *accurate (relative errors less than 10%), fast (responses in less than 0.05s)* throughput prediction.
- We propose Prophet, a novel system to provide the service of throughput predictions for reactive flows, based on our theoretical findings.
- We implement a prototype and evaluate it extensively. It is demonstrated that Prophet can provide accurate estimation results even in multiple scenarios.

The rest of the paper is organized as follows. In Section II we formally define the problem of throughput prediction for reactive flows. Theoretical analysis is conducted and we design novel approaches to solve the prediction problem in Section III. In Section IV, we propose Prophet, a system to provide the flow query service based on the theoretical results and advanced monitoring. The evaluation results are presented in Section V. Finally, we compare with related work in Section VI and summarize our paper in Section VII.

II. Problem Statement

A. Motivation

Unlike predicting throughput for non-reactive flows which can be achieved by checking the available reservation, predicting throughput for reactive flows is much more difficult. Consider the example in Fig. 1, there are two flows in the original network, denoted as *background flows* f_1 and f_2. The two flows are *fully utilized*, meaning they do not have any source constraints and consume all available bandwidth.

Consider the prediction for a new flow q_1 arriving at the network. If the flow is not source-constrained, it can get 1/3 of the total bandwidth, *i.e.* 200 Mbps as in Fig. 1(b). In the general case, the new flow can be source-constrained, and the preceding result is only an upper bound. Assume the source limit is 50 Mbps, q_1 will only consume 50 Mbps as in Fig. 1(c).

The ability to predict the throughput is more important for a set of flows. Specifically, consider two simultaneous flows q_1 and q_2. If the two new flows are not source-constrained, they will both achieve 150 Mbps. However, if one flow, say q_1, is source-constrained, the result becomes more complex. As shown in Fig. 1(d), assume that q_1 is source-limited to 60 Mbps, one may compute that q_2 will achieve 180 Mbps (=(600-60)/3). If q_1 is limited to 180 Mbps, on the other hand, both q_1 and q_2 will receive 150 Mbps. In general settings with complex networks and complex flow source constraints, the prediction service must be able to respect source constraints.

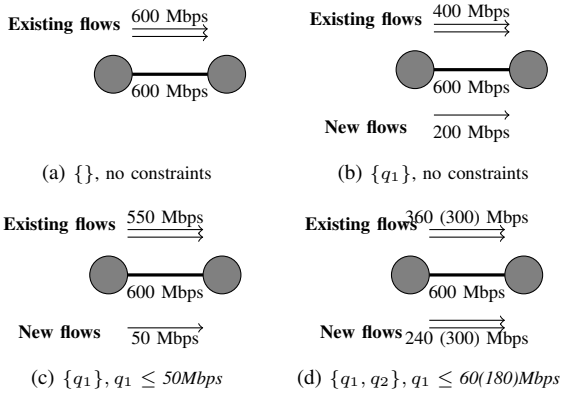


Fig. 1: Throughput for Different Flows and Source Constraints.

B. Problem Definition

Network: We consider a data center network with N full duplex edge connections where network congestion only happens at these edge connections [22]. The network core can be seen as a non-blocking switch and each connection can be seen as two separate links. The links are numbered from 1 to $2N$ and the set of links is denoted by L. The k-th link is represented as l_k, and we use c_k to represent its capacity. The capacity vector $\boldsymbol{c}$ is defined as a column vector that $\boldsymbol{c} = \langle c_1, \ldots, c_{(2N)} \rangle^T$.

Background flows: There are K running reactive flows in the network. We number these "background" flows from 1 to K where the i-th flow is denoted by f_i whose source is denoted by s_i and destination by d_i. F is a set consisting of all the background flows. The throughput of the i-th flow is denoted as x_i and the throughput vector $\boldsymbol{x} = \langle x_1, \ldots, x_K \rangle^T$. The i-th flow f_i may have a source rate limit of τ_i, where $\boldsymbol{\tau} = \langle \tau_1, \ldots, \tau_K \rangle$. **We also abuse the symbols F and f_i in the general case when there is no need to distinguish between background flows and queried flows**.

Flow query: We consider a flow query Q which consists of M reactive flows numbered from 1 to M. We refer to the i-th flow as q_i, and denote its throughput by y_i. The throughput vector $\boldsymbol{y}$ is a column vector defined as $\boldsymbol{y} = \langle y_1, \ldots, y_M \rangle^T$. The i-th flow q_i may have a source rate limit of π_i, where $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_M \rangle$.

Routing: We consider the case that routing is already known. The routing matrices A and B are binary matrices of size $2N \times K$ and $2N \times M$ respectively, which represents how flow f_i and q_i traverses the network. In particular, $a_{ki} = 1$ if and only if f_i traverses l_k and $b_{ki} = 1$ if and only if q_i traverses l_k.

Based on the network system model, we define the problem of predicting throughput for reactive flows.

Problem 1 (Throughput Prediction for Reactive Flows)**.** Given a network with reactive flows and a flow query Q with source constraints $\boldsymbol{\pi}$, return the predicted throughput $\hat{\boldsymbol{y}}$.

One can prove that when there are non-reactive flows in the network, *i.e.* a set of flows S where any flow $f \in S$ has a fixed

TABLE I: Symbols

Scope	*Symbol*	*Meaning*
Network	N	# of access ports/links
	L	Set of links where $\|L\| = 2N$
	l_k	The k-th link
	c_k	Link capacity of l_k
Background Flows	K	# of running flows in a network
	F	Set of running flows where $\|F\| = K$
	f_i	The i-th running flow
	s_i/d_i	Source/destination host of f_i
	x_i	Throughput of flow f_i
	τ_i	Source constraint for f_i
Flow Query	M	# of flows in a flow query
	Q	Set of flows in a query where $\|Q\| = M$
	q_i	The i-th flow
	y_i	Throughput of flow q_i
	π_i	Source constraint for q_i
Routing	a_{ki}	$a_{ki} = 1$ indicates f_i traverses l_k
	b_{ki}	$b_{ki} = 1$ indicates q_i traverses l_k
Monitoring & Learning†	ρ_i	Utility scaling factor for f_i or q_i
	p_i	Equivalent class index for f_i or q_i
	$\bar{\rho}_j$	$\rho_i = \bar{\rho}_{p_i}$
	α_i	Utility parameter for f_i or q_i
	x_i^*	Equilibrium throughput for f_i or q_i
	$s\hat{y}m$	Estimated result of *sym* $(x_i, y_i, \dots)$
	$s\tilde{y}m$	*sym* $(x_i, y_i, \dots)$ for the sampled flows

† Unless explicitly stated in the context of flow queries, $\rho_i, p_i, \dots$ and x_i^* are associated with f_i.

bandwidth guarantee, throughput estimation for the reactive flows can be converted to the problem of *throughput prediction for reactive flows* by subtracting the reserved bandwidth from c_k. Thus, we only consider reactive flows in this paper.

III. Design Foundation

In this section, we start with the most basic setting (*i.e.*, homogeneous reactive mechanism and no source constraints) to form a fundamental understanding of how to predict throughput for reactive flows. We then increase the complexities by *considering heterogeneous TCP implementations* in Section III-B and *handling source-constrained flows* in Section III-C, as demonstrated in Fig. 2.

A. Starting Point

Our throughput prediction solution is based on the *Network Utility Maximization* (NUM) [20] which is the theoretical foundation for many TCP designs. Let $U_i(x_i)$ denote the utility when the throughput of f_i is x_i, the allocation for each flow is a solution to the following optimization problem:

$$\textit{System} \qquad \max \sum_{f_i \in F} U_i(x_i) \tag{1}$$

$$\textit{Subject to} \qquad A\boldsymbol{x} \leq \boldsymbol{c}.$$

The problem of *throughput prediction for reactive flows* can then be solved by finding the optimal solution of the following optimization problem:

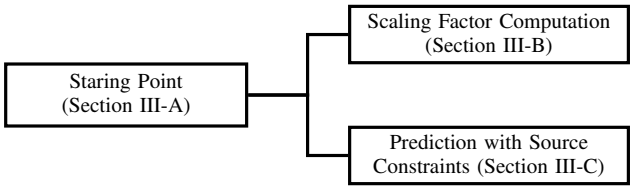


Fig. 2: Design Road Map.

$$\textit{System} \qquad \hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} \bar{U}_i(y_i) \right) \tag{2}$$

$$\textit{Subject to} \qquad \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \boldsymbol{c}.$$

To solve this optimization problem, it is essential to know the utility function. In particular, our method is based on the unifying model introduced by Srikant [19] where utility function of a TCP flow is modeled as:

$$U(x) = \rho \cdot \frac{x^{1-\alpha}}{1-\alpha}. \tag{3}$$

The parameter α is determined by the TCP congestion control algorithm. For example, the values of α for TCP Vegas and Reno [19] are 1 and 2 respectively. Existing works [23]–[25] have proposed sophisticated solutions on how to identify the TCP implementations, which means that we can assume that the α of each flow is already known. Thus, given x, the value of a utility function is proportional to the unknown parameter ρ, which we refer to as the *scaling factor*.

B. Scalable, Fast Scaling Factor Computation

In this section, we describe how to reduce the scale of the estimation problem and propose the theoretical foundation of estimation the scaling factors $\boldsymbol{\rho} = \langle \rho_1, \dots, \rho_K \rangle$ where ρ_i is the scaling factor for flow f_i.

For a given $\boldsymbol{\rho}$, the optimization problem (1) has a unique optimal solution since the utility functions are concave. Thus, we consider $\boldsymbol{x}$ as a function of $\boldsymbol{\rho}$, as in Lemma 1.

Lemma 1 (Optimality of Equilibrium Bandwidth)**.** Consider a network system where each running TCP flow f_i has its scaling factor ρ_i. If the equilibrium bandwidth of all running flows $\boldsymbol{x}^*$ fits the condition $A\boldsymbol{x}^* = \boldsymbol{c}$, it should be the maximal point of the following *Lagrange function*:

$$\mathcal{L}(A, \boldsymbol{c}, \boldsymbol{\rho}; \boldsymbol{x}, \boldsymbol{\lambda}) = \sum_i \rho_i \frac{x_i^{1-\alpha_i}}{1-\alpha_i} - \boldsymbol{\lambda}^{\mathrm{T}}(A\boldsymbol{x} - \boldsymbol{c}).$$

Proof. This lemma can be trivially proved by substituting $U_i(x_i)$ with $\rho_i \frac{x_i^{1-\alpha_i}}{1-\alpha_i}$ in the *Lagrangian* of the optimization problem (1). □

The best estimation of $\boldsymbol{\rho}$ maximizes the likelihood between the actual bandwidth allocation ($\boldsymbol{x}$) and the one ($\hat{\boldsymbol{x}}$) estimated

from $\boldsymbol{\rho}$. In particular, we use the *least square* to measure the error of $\hat{\boldsymbol{x}}(\boldsymbol{\rho})$, *i.e.*,

$$E(\boldsymbol{\rho}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|^2. \tag{4}$$

In this paper, we make a mild assumption that a host uses the same TCP implementation throughout our prediction. This is a rational assumption since hosts typically do not change the TCP algorithms very frequently.

1) Improving scalability with equivalent classes: A first issue to be resolved is to *reduce the number of variables to be estimated*. In an arbitrary network, each flow potentially has a unique link price (dual variables in the NUM problem) so that its scaling factor ρ is also unique. For a network with K flows, a brute-force approach has to solve a K-variable estimation problem, which can be large in real world cases.

A key observation is that *many flows have similar scaling factors*. In a classic Clos topology [26], any connection can only have 1 out of 3 different hop count values, and respectively 3 different propagation delays. Since scaling factors usually depend only on TCP implementation and round trip time, scaling factors of flows with the same source can be approximately classified into three different groups.

For a network where there are P different scaling factors, the scaling factor of each flow belongs to one of the P equivalent classes. Let p_i denote the equivalent class index of flow f_i, which is uniquely determined by the source host and the routing matrix. Let $\bar{\rho}_i$ denote the scaling factor value of the i-th equivalent class, we have $\rho_i = \bar{\rho}_{p_i}$. *The problem of estimating $\boldsymbol{\rho}$ is now reduced to estimating the scaling factors of all equivalent classes $\bar{\boldsymbol{\rho}}$.*

By grouping flows with similar scaling factors, we substantially reduce the number of variables from $N(N-1)$ to $3N$. In a Clos topology where $k = 4$, our reduced model only has 48 variables while the brute-force method has 240.

2) Achieving fast convergence with gradient descent method: The second issue is *convergence*. We derive the gradient of $\bar{\boldsymbol{\rho}}$ to speed up the computation.

Proposition 1. Given a routing matrix A, a capacity vector $\boldsymbol{c}$ and the equilibrium flow rates $\boldsymbol{x}$, there is no neighbourhood $U(\bar{\boldsymbol{\rho}}_{\mathbf{0}}, \delta)$ of the scaling factor $\bar{\boldsymbol{\rho}}$ which makes the following formula always true:

$$\forall \bar{\boldsymbol{\rho}} \in U(\bar{\boldsymbol{\rho}}_{\mathbf{0}}, \delta), \quad \det(A\Lambda^{-1}A^{\mathrm{T}}) \equiv 0,$$

where $\Lambda = \mathrm{diag}(\rho_{p_i}\alpha_i x_i^{-\alpha_i - 1})$.

Proof. Since $\det(A\Lambda^{-1}A^{\mathrm{T}})$ is a polynomial of $\bar{\boldsymbol{\rho}}$, if it is always zero in the neighbourhood $U(\bar{\boldsymbol{\rho}}_{\mathbf{0}})$, it must be zero in $\Re^P$. But we can always find a diagonal matrix Λ which makes $\det(A\Lambda^{-1}A^{\mathrm{T}})$ non-zero. So the assumption is not valid. □

Now we show that the gradient of the error estimation function $E(\bar{\boldsymbol{\rho}})$ can be computed very efficiently.

Substituting $\boldsymbol{\rho}$ with $\bar{\boldsymbol{\rho}}$ in Lemma 1, we have

$$\mathcal{L}(A, \boldsymbol{c}, \bar{\boldsymbol{\rho}}; \boldsymbol{x}, \boldsymbol{\lambda}) = \sum_i \bar{\rho}_{p_i} \frac{x_i^{1-\alpha_i}}{1-\alpha_i} - \boldsymbol{\lambda}^{\mathrm{T}}(A\boldsymbol{x} - \boldsymbol{c}),$$

where $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ subject to the following equations:

$$\nabla_{\boldsymbol{x}}\mathcal{L} = 0 \quad \Rightarrow \qquad \forall j, \quad \sum_k a_{kj}\lambda_k = \bar{\rho}_{p_j} x_j^{-\alpha_j}, \tag{5}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = 0 \quad \Rightarrow \qquad \forall k, \quad \sum_j a_{kj} x_j = c_k. \tag{6}$$

The solution of the bandwidth allocation problem for a given $\bar{\boldsymbol{\rho}}$ is a function of $\bar{\boldsymbol{\rho}}$. We denote it as $\hat{\boldsymbol{x}}$ and use the following symbols for simplicity:

$$\delta x_{ji} = \frac{\partial \hat{x_j}}{\partial \bar{\rho}_i}, \; \delta\lambda_{ki} = \frac{\partial \hat{\lambda_k}}{\partial \bar{\rho}_i}.$$

Now consider the partial derivatives for (5) and (6) at $\boldsymbol{\rho}$, we have

$$\begin{cases} \bar{\rho}_{p_j}\alpha_j x_j^{-\alpha_j - 1}\delta x_{ji} + \sum_k a_{kj}\delta\lambda_{ki} = 0 & \forall i, \forall j \text{ s.t. } p_j \neq i, \\ \bar{\rho}_i\alpha_j x_j^{-\alpha_j - 1}\delta x_{ji} + \sum_k a_{kj}\delta\lambda_{ki} = x_j^{-\alpha_j} & \forall i, \forall j \text{ s.t. } p_j = i, \\ \sum_j a_{kj}\delta x_{ji} = c_k & \forall i, \forall k. \end{cases}$$

It is equivalent to the following linear equation:

$$\begin{pmatrix} \Lambda & A^{\mathrm{T}} \\ A & O \end{pmatrix} \begin{pmatrix} \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \\ \mathbf{J}_{\boldsymbol{\lambda}}(\bar{\boldsymbol{\rho}}) \end{pmatrix} = \begin{pmatrix} \Gamma \\ \boldsymbol{c}J_{1,P} \end{pmatrix}, \tag{7}$$

where

$$\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) = (\delta x_{ji}), \mathbf{J}_{\boldsymbol{\lambda}}(\bar{\boldsymbol{\rho}}) = (\delta\lambda_{ki}),$$

$$\Gamma_{j,i} = \begin{cases} x_j^{-\alpha_j} & \text{if } p_j = i, \\ 0 & \text{if } p_j \neq i. \end{cases}$$

If the coefficient matrix of (7) is reversible, $\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})$ can be computed very efficiently. From Proposition 1, we know there is no continous $\bar{\boldsymbol{\rho}}$ making the determinant of this coefficient matrix $\det(O - A\Lambda^{-1}A^T)$ always zero, which means we can always generate the next $\bar{\boldsymbol{\rho}}$ iteratively or make a small disturbance to get a reversible coefficient matrix.

Now we can use the gradient descent method to minimize the error. To eliminate the drawback of using a fixed step with Cartesian coordinates, we choose spherical coordinates and the gradient is calculated as follows.

Let $\hat{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})$ denote the estimated value of $\boldsymbol{x}$ by $\bar{\boldsymbol{\rho}}$ and $(r, \boldsymbol{\phi})$ be the spherical coordinate transform of $\bar{\boldsymbol{\rho}}$, $\hat{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) = \hat{\boldsymbol{x}}(\boldsymbol{\phi})$. Let $\boldsymbol{e}$ be the unit vector of $\bar{\boldsymbol{\rho}}$, *i.e.* $\boldsymbol{e} = \frac{\bar{\boldsymbol{\rho}}}{\|\bar{\boldsymbol{\rho}}\|}$. $\boldsymbol{e}$ is also a function of $\boldsymbol{\phi}$. Let $\mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})$ be the Jacobian matrix of $\boldsymbol{e}(\boldsymbol{\phi})$. We can get:

$$\nabla_{\boldsymbol{\phi}} E = 2(\hat{\boldsymbol{x}} - \boldsymbol{x})^{\mathrm{T}}\mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}})\mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})^{\mathrm{T}}. \tag{8}$$

C. Prediction with Source-Constrained Background Flows

We now consider the more general problem of throughput prediction for source-constrained flows. Two major challenges are raised: 1) how to compute the scaling factors with source constraints, and 2) how to obtain the source constraints for background flows.

For the first challenge where $\boldsymbol{\tau}$ is already given, we extend the optimization problem in (2) to include source constraints in

our prediction framework. The throughput prediction problem is equivalent to solving the following optimization problem:

$$\textit{System} \quad \max \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} U_i(y_i) \right) \tag{9}$$

$$\textit{Subject to} \quad \begin{pmatrix} A & B \\ I & O \\ O & I \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{\tau} \\ \boldsymbol{\pi} \end{pmatrix}.$$

Thus, the throughput estimation in the FFL model is tranformed to a new format using the augmented routing matrix and constraint vector.

For the second challenge where $\boldsymbol{\tau}$ is also unknown, the problem is further complicated. Instead of taking the naive approach by augmenting the original FFL model with $\boldsymbol{\tau}$ as variables, we use a much simpler method. Our solution is based on the following observation.

A fundamental difference between source-constrained flows and non-reactive flows, even though they may appear similar because flows of both types may have a fixed transmission rate, is that *source-constrained flows only transmit at a fixed rate when its throughput is less than the equilibrium throughput*.

Proposition 2 (Hard Constraint)**.** Let $\langle \boldsymbol{x'}, \boldsymbol{y'} \rangle^T$ be the optimal solution of (2) and $\langle \boldsymbol{x''}, \boldsymbol{y''} \rangle^T$ be the optimal solution of (9). We have the following conclusion:

$$x''_i = \tau_i, \qquad \forall i, x'_i > \tau_i$$
$$y''_j = \pi_j, \qquad \forall j, y'_j > \pi_j.$$

Proof. Consider the Lagrangian of optimization problems (2) and (9) (denoted as $\mathcal{L}_{(2)}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}; \boldsymbol{\lambda})$ and $\mathcal{L}_{(9)}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}; \boldsymbol{\lambda}, \boldsymbol{\mu})$ respectively). Without loss of generality, we consider a specific i such that $x'_i > \tau_i$. First, consider the derivatives of x_i, we have

$$\nabla_{x_i} \mathcal{L}_{(2)} = \frac{\partial U_i}{\partial x_i}(\cdot) - \sum_k a_{ki} \lambda_k \tag{10}$$

$$\nabla_{x_i} \mathcal{L}_{(9)} = \frac{\partial U_i}{\partial x_i}(\cdot) - \sum_k a_{ki} \lambda_k - \mu_i. \tag{11}$$

Since $\boldsymbol{x''}$ also satisfies the constraints of (2), consider the gradient of x_i at $\boldsymbol{x''}$ and the corresponding $\boldsymbol{\lambda''}$. Since $x'_i > \tau_i \geq x''_i$, $\nabla_{x_i} \mathcal{L}_{(2)}(\boldsymbol{x''}, \boldsymbol{\lambda''}) > 0$ but $\nabla_{x_i} \mathcal{L}_{(9)}(\boldsymbol{x''}, \boldsymbol{\lambda''}) = 0$. Thus, $\mu_i > 0$ and according to the *Karush-Kuhn-Tucker* conditions, $x''_i = \tau_i$.

Similarly we can prove $\forall j, y'_j > \pi_j$, $y''_j = \pi_j$. □

Our identification method on $\boldsymbol{\tau}$ is based on Proposition 2 but uses it in a reversed way. If the estimated equilibrium rate $\hat{x}_i$ is "significantly larger"[1] than the actual throughput x_i, we consider this flow being source-constrained with a rate limit of x_i and use it in future computations. In a series of samples, the source constraint of a given flow f_i cannot be "significantly smaller" than the average actual throughput, $\bar{x}_i$.

[1]We allow a 10% relative error, *i.e.*, if $x > (1 + 10\%)y$, we say x is significantly larger than y.

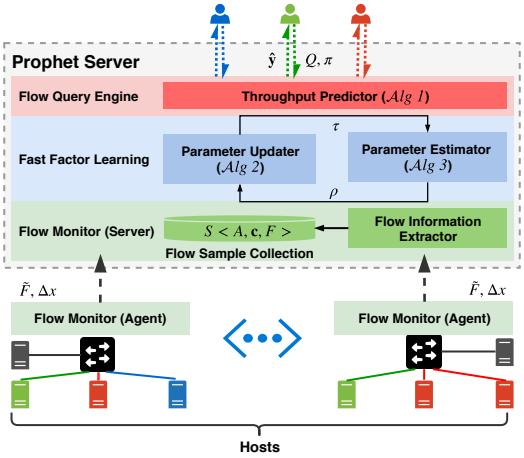


Fig. 3: The Prophet System Overview

IV. The Prophet System

From the observations and analysis in Section III, we have derived what kind of information is necessary to solve the problem of throughput prediction for reactive flows. In this section, we introduce the details of the Prophet system based on the results.

A. System Overview

As demonstrated in the figure, Prophet consists of the following components:

- **Flow query engine**: A flow query engine receives flow queries from users. It uses the estimated parameters learned by the FFL component and calculates the estimated throughput prediction for flow queries based on (9).
- **Fast factor learning**: The *fast factor learning* (FFL) component uses flow samples collected by flow monitors to train a TCP utility function estimation model and learn the parameters for each flow continuously. The component iteratively updates the parameters once the information of new flows is reported. Using the estimated parameters, the FFL component can construct utility functions.
- **Flow monitor**: A flow monitor monitors the traffic for a set of given hosts, extracts the header information which is further processed by TCP classifiers. It is also responsible for monitoring real-time traffic at the access switch.

B. Throughput Prediction for Reactive Flow Queries

For a flow query Q, consider the throughput prediction problem in (9). Background flows F and the routing matrix A are provided by the network controller, the routing matrix of queried flows B is computed from Q, source constraints of queried flows $\boldsymbol{\pi}$ are (optionally) provided along with Q, and finally source constraints for background flows $\boldsymbol{\tau}$, parameters of utility functions $\boldsymbol{\alpha}$ and $\bar{\rho}$ are provided by the FFL component.

Thus, we can conduct throughput prediction for Q by solving the problem of (9), as in Algorithm 1. The algorithm constructs utility functions for background flows (Line 2-3)

Alg. 1: Throughput Prediction for Reactive Flows

Input: $F, A, \boldsymbol{c}, \boldsymbol{\tau}, \boldsymbol{\rho}; Q, \boldsymbol{\pi}$
Output: $\hat{\boldsymbol{y}}$ - estimated equilibrium bandwidth

1 **Function** PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \boldsymbol{\rho}; Q, \boldsymbol{\pi}$)
2 $\quad \boldsymbol{\alpha}, \boldsymbol{p} \leftarrow$ IDENTIFYTCP(F)
3 $\quad$ **foreach** $f_i \in F$ **do**
4 $\quad\quad U_i(x) \leftarrow \bar{\rho}_{p_i} \frac{x^{1-\alpha_i}}{1-\alpha_i}$
5 $\quad \boldsymbol{\alpha'}, \boldsymbol{p'} \leftarrow$ IDENTIFYTCP(Q)
6 $\quad$ **foreach** $q_i \in Q$ **do**
7 $\quad\quad \bar{U}_i(x) \leftarrow \bar{\rho}_{p'_i} \frac{x^{1-\alpha'_i}}{1-\alpha'_i}$
8 $\quad B \leftarrow$ GETROUTINGMATRIX(Q)
9 $\quad U(\boldsymbol{x}, \boldsymbol{y}) \leftarrow \left(\sum_{f_i \in F} U_i(x_i) + \sum_{q_i \in Q} \bar{U}_i(y_i)\right)$
10 $\quad A' \leftarrow \begin{pmatrix} A & B \\ I & O \\ O & I \end{pmatrix}, \boldsymbol{c'} \leftarrow \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{\tau} \\ \boldsymbol{\pi} \end{pmatrix}$
11 $\quad \hat{\boldsymbol{y}} \leftarrow \arg\max U(\boldsymbol{x}, \boldsymbol{y})$ subject to $A' \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \boldsymbol{c'}$
12 $\quad$ **return** $\boldsymbol{y}$

Alg. 2: Parameter Updates based on Flow Samples

1 **Function** *ModelLearningDaemon*
2 $\boldsymbol{\alpha} \leftarrow \emptyset$, $\boldsymbol{\tau} \leftarrow \emptyset$, $\bar{\boldsymbol{\rho}} \leftarrow \mathbf{1}$, $S \leftarrow \emptyset$
3 **for** $k = 1, \ldots, +\infty$ **do**
4 $\quad$ **for** $i = 1, \ldots, |\tilde{F}^{(k)}|$ **do**
5 $\quad\quad$ **if** $\tilde{f}_i \notin F$ **then**
6 $\quad\quad\quad \tau_i \leftarrow +\infty$,
7 $\quad \boldsymbol{\alpha}, \boldsymbol{p} \leftarrow$ IDENTIFYTCP($\tilde{F}^{(k)}$), $\boldsymbol{c} \leftarrow \boldsymbol{c_0} - \boldsymbol{\Delta x}^{(k)}$
8 $\quad F \leftarrow \tilde{F}^{(k)}$, $A \leftarrow$ GETROUTINGMATRIX(F)
9 $\quad S \leftarrow \langle A, \boldsymbol{c}, F \rangle$
10 $\quad \bar{\boldsymbol{\rho}} \leftarrow$ ESTIMATEPARAMETERS($\boldsymbol{\tau}, \bar{\boldsymbol{\rho}}, S$)
11 $\quad \hat{\boldsymbol{x}} \leftarrow$ PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \bar{\boldsymbol{\rho}}; \emptyset, \emptyset$)
12 $\quad \boldsymbol{\tau} \leftarrow$ UPDATE($\hat{\boldsymbol{x}}, \tilde{\boldsymbol{x}}$)

Alg. 3: TCP Parameter Estimation

Input: $\boldsymbol{\tau}$, $\bar{\boldsymbol{\rho}}$, S
Output: $\hat{\bar{\boldsymbol{\rho}}}$ - updated estimated parameters

1 **Function** ESTIMATEPARAMETERS($\boldsymbol{\tau}, \bar{\boldsymbol{\rho}}, S$)
2 $\quad \boldsymbol{\phi} \leftarrow$ CARTESIANTOSPHERICAL($\bar{\boldsymbol{\rho}}$)
3 $\quad \varepsilon \leftarrow \varepsilon_0$
4 $\quad$ **while** $\varepsilon > tolerance$ **do**
5 $\quad\quad E \leftarrow 0, \nabla_{\boldsymbol{\phi}} E \leftarrow \mathbf{0}$
6 $\quad\quad$ **foreach** $< A, \boldsymbol{c}, F > \in S$ **do**
7 $\quad\quad\quad \bar{\boldsymbol{\rho}} \leftarrow$ SPHERICALTOCARTESIAN($\boldsymbol{\phi}$)
8 $\quad\quad\quad \hat{\boldsymbol{x}} \leftarrow$ PREDICTTHROUGHPUT($F, A, \boldsymbol{c}, \boldsymbol{\tau}, \bar{\boldsymbol{\rho}}; \emptyset, \emptyset$)
9 $\quad\quad\quad E \leftarrow E + \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|^2$
10 $\quad\quad\quad \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \leftarrow$ SOLVE($Equation\ 7$)
11 $\quad\quad\quad \nabla_{\boldsymbol{\phi}} E \leftarrow \nabla_{\boldsymbol{\phi}} E + 2(\hat{\boldsymbol{x}} - \boldsymbol{x})^{\mathrm{T}} \mathbf{J}_{\boldsymbol{x}}(\bar{\boldsymbol{\rho}}) \mathbf{J}_{\bar{\boldsymbol{\rho}}}(\boldsymbol{\phi})$
12 $\quad\quad \boldsymbol{\phi} \leftarrow$ SGD($\boldsymbol{\phi}, E, \nabla_{\boldsymbol{\phi}} E$)
13 $\quad\quad \varepsilon \leftarrow \frac{E}{\sum_{\boldsymbol{x}} \dim \boldsymbol{x}}$
14 $\quad \hat{\bar{\boldsymbol{\rho}}} \leftarrow$ SPHERICALTOCARTESIAN($\boldsymbol{\phi}$)
15 $\quad$ **return** $\hat{\bar{\boldsymbol{\rho}}}$

and queried flows (Line 4-6) respectively. It then builds the routing matrix for the queried flows in Line 7 and constructs the optimization problem in (9). The estimated throughput is calculated by solving the optimization problem.

C. Computing Scaling Factors and Source Constraints

In this section, we describe how the information required by throughput prediction are learned from flow samples.

Let $\tilde{F}$ denote the sampled flows and $\boldsymbol{\Delta x}$ as the aggregated throughput of source-constrained small flows that are considered as non-reactive flows as in Proposition 2. Given the sampled flows $\tilde{F}$ and $\boldsymbol{\Delta x}$, we need to update five parameters that are essential to predict throughput: active background flow set F, its routing matrix A, source constraints $\boldsymbol{\tau}$, link capacities $\boldsymbol{c}$, and utility function parameters $\bar{\boldsymbol{\rho}}$.

We number the flow samples received from 1 incrementally. Let $\tilde{F}^{(k)}$ and $\boldsymbol{\Delta x}^{(k)}$ denote the k-th samples. Algorithm 2 describes how we process the samples.

For the i-th flow $\tilde{f}_i \in \tilde{F}^{(k)}$, $\tilde{\alpha}_i$ is determined by the TCP algorithm used by its source $\tilde{s}_i$. We use mechanisms from previous studies [23]–[25] to identify TCP implementations and set the α for $\tilde{f}_i$. The equivalence class index p_i is identified as a combination of $\tilde{s}_i$ (TCP implementation) and the distance between $\tilde{s}_i$ and $\tilde{d}_i$ (link price). These steps are represented using a function called IDENTIFYTCP.

Source constraint $\tilde{\tau}_i$ is set to $+\infty$ at the beginning (Line 6) and is continuously updated throughout the life cycle of f_i in our prediction model. We use $\tau_i^{(k)}$ to denote the values after k samples. Further processing on $\boldsymbol{\tau}^{(k)}$ is discussed later.

The link capacities are calculated by subtracting the total throughput of source-constrained flows $\boldsymbol{\Delta x}$ from the physical link capacity $\boldsymbol{c_0}$ (Line 7). The active background flow set $F^{(k)}$ is set to $\tilde{F}^{(k)}$ directly (Line 8). We get the routing matrix A from the network controller (Line 8).

The values of $\tau_i^{(k)}$ is updated using the methods introduced in Section III-C. The steps are denoted as function UPDATE (Line 11-12). In particular, we use the sampled throughput $\tilde{x}_i$ as an estimation of actual x_i.

The input of Algorithm 3 is a set of flow samples. Each flow sample is formulated as a 3-tuple $< A, \boldsymbol{c}, F >$ where A and $\boldsymbol{c}$ are as defined in Table I and F is a set of hash code. Each hash code in F indexes a record of flow f_i, which contains the TCP identification result α_i and p_i, the measured equilibrium bandwidth x_i and the estimated source constraint τ_i.

Algorithm 3 first transforms initial scaling factor $\bar{\boldsymbol{\rho}}$ to spherical coordinates $\boldsymbol{\phi}$. It then updates $\boldsymbol{\phi}$ by the sample set S iteratively and estimates the error ε until ε is less than a given threshold. In each loop, the algorithm passes the current $\bar{\boldsymbol{\rho}}$ to Algorithm 1 to get a bandwidth estimation for each sample in S. Then it calculates the value of E and $\nabla_{\boldsymbol{\phi}} E$ which are defined in (4) and (8). With this information, the algorithm applies *Stochastic Gradient Descent* to update $\boldsymbol{\phi}$. After the result is converged, the algorithm transforms $\boldsymbol{\phi}$ back to the Cartesian coordinates $\hat{\bar{\boldsymbol{\rho}}}$, which minimizes the error of bandwidth estimation E, and return it.

D. Extracting Flow Information

In this section, we describe how flow samples are collected at flow monitors.

From the analysis in Section III, Prophet requires the actual flow throughput to be collected. However, in a real world scenario, some adjustment must be taken into consideration.

We use sampling to gather the information on the access switches. Each flow is mapped to the TCP 5-tuple. The throughput of a flow is estimated as the total bytes in a sampling period divided by the sampling time. We construct $\tilde{F}$ by selecting the flows with throughput larger than 5% of the total bandwidth as *sampled flows* and the rest are considered *source-constrained* with a small rate limit, whose total throughput of the source-constrained flows Δx_k on link l_k is measured.

V. Evaluation

We have developed a prototype of the Prophet system. In this section, we conduct extensive experiments and evaluate the accuracy of throughput prediction for reactive flows.

A. Settings

System configuration: We use Mininet 2.2.1 and Network Simulator ns-2.35 to simulate the network. The prototype is implemented in Python and evaluations are conducted on a 64-bit Ubuntu 14.04 LTS server with 4-core Intel(R) Xeon(R) E5-2609 v2 (@ 2.50GHz) CPU, 32G memory.

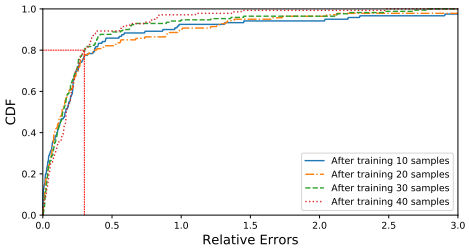
Network setting: The network used in our evaluations is a Clos topology with $k = 4$, *i.e.* 16 hosts. The link capacities between *core* and *aggregation*, and between *aggregation* and *edge* are set to 1 Gbps to ensure no congestion in the network core. The link capacity between an edge switch and a host is 1 Mbps. All links have the same propagation delay of $2us$.

Methodology: For each evaluation, we initiate some random background flows and capture their real time throughput. Then a random flow query is made for which we predict the throughput. The flows in the query are then added to the network. We capture their equilibrium throughput and compare the results with our estimations.

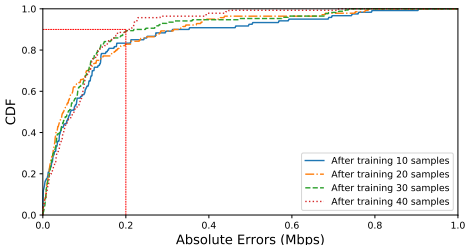
Metrics: We measure the following metrics: 1) **throughput error** between an estimation $\hat{\boldsymbol{x}}$ and the actual throughput $\boldsymbol{x}$, which represents the accuracy of a prediction and is calculated as $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|$ (absolute error) and $\frac{\|\hat{\boldsymbol{x}}-\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$ (relative error), 2) **scaling factor error**, which indicates the convergence speed of the training process, and 3) **execution time** of both the training and the prediction stages.

B. Throughput Prediction for A Single TCP Implementation

We evaluate the performance of Prophet in the setting with only one TCP implementation. In particular, we use TCP Vegas in a Clos topology with $K = 4$, with the number of flows in a query randomly picked in $[10, 20]$. We conduct 10 simulations on NS2. For each execution, an initial set of flows is launched first and 50 queries are made consecutively. For each query, we record the predicted throughput and add the flows to the network. After the network has reached the equilibrium, we record the throughput for the queried flows and compare with the prediction.



(a) CDF of Relative Errors



(b) CDF of Absolute Errors

Fig. 4: Errors for Throughput Prediction.

Fig. 4a demonstrates the cumulative distribution of relative errors of the queried reactive flows. As we can see, more than 80% of the flow queries have a relative error of less than 30% and more than 40% flows are less than 10%, which demonstrates *that our throughput prediction system can achieve high accuracy from 70% up to 90%.*

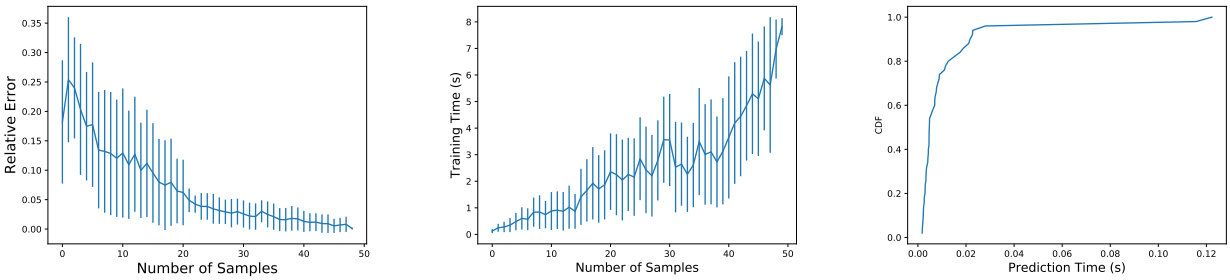
The curve in Fig. 5(a) demonstrates the relative errors between estimated scaling factors $\bar{\boldsymbol{\rho}}$ after training with k samples and the final scaling factors. As we can see, *the scaling factors converge very quickly* with a relative error of approximately 5% after around 20 samples.

The training time is presented in Fig. 5(b). Not surprisingly, the training time increases as the number of samples increase. However, the relatively large standard deviations and the jitters indicate that the execution time depends heavily on the actual flow distribution. From the previous analysis on the convergence of scaling factors, a reasonable cut-off value is 20, where the training time is mostly less than 4 seconds.

The prediction time is demonstrated in Fig. 5(c). Since the prediction is independent of sampling numbers, we are more interested in their distributions. As we can see from Fig. 5(c), the prediction for 10-20 flows is pretty fast. All queries are responded in less than 0.15s and about 90% of the queries only takes less than 0.05s. We conclude that *Prophet can efficiently predict throughput for reactive flows.*

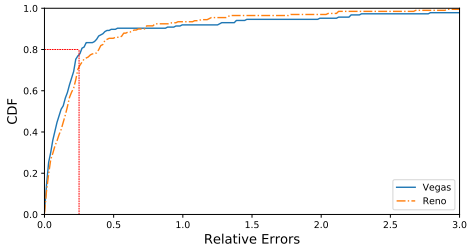
C. Throughput Prediction for Multiple TCP Implementations

We use the same methodologies as in the single TCP setting, with the exception that now multiple TCP implementations are

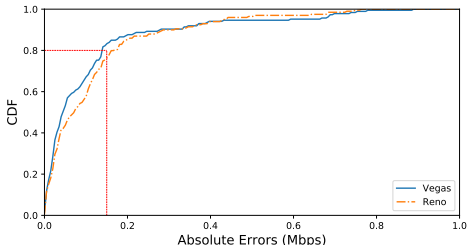


(a) Relative Errors of $\bar{\rho}$ (b) Training Time (c) Query Response Time

Fig. 5: Performance of a Clos Topology with $K = 4$, 50 Queries with $10 - 20$ Flows.



(a) CDF of Relative Errors



(b) CDF of Absolute Errors

Fig. 6: Errors for Multiple TCP Implementations.

used in the experiments. We randomly select TCP implementation algorithms for all the hosts in the network and make all flows from the same source use the same TCP implementation.

Again we demonstrate the throughput prediction results in Fig. 6. As we can see, the prediction results are similar to the single TCP implementation case that for 80% of the flows, the relative errors are less than 30%. We can also see that for different TCP implementations, Reno and Vegas in this case, the prediction accuracy may vary slightly.

D. Summary

In our evaluations, we have demonstrated that Prophet achieves accurate throughput predictions for reactive flows for networks of either single or multiple TCP implementations. We also demonstrate that the training model can converge with a small number of samples (10-20), and provide fast responses (90% responses in less than 0.05s).

VI. Related Work

TCP throughput estimation. Throughput estimation, or bandwidth estimation, is a widely-studied topic and is the foundation of many TCP implementations. TCP Vegas [27] estimates the throughput as the *expected rate*, defined by *cwnd*/*propagation delay*. TCP Westwood [17] takes a series of throughput samples and uses different low-pass filters for throughput estimation. BBR [28] estimates the throughput by finding the optimal operating point where throughput stops increasing alongside RTT. These throughput estimation methods are usually deployed on end hosts and can only provide the estimation after a flow is instantiated. More importantly, they cannot predict throughput for *a set of flows*. Instead, Prophet predicts throughput both *before* flows are launched and for *a set of reactive flows*.

Network performance prediction. Centralized network management systems, such as SDN, have enabled the ability to provide network views to applications, which makes it possible to predict network performance [7]–[11], [29]. However, existing solutions provide either a deterministic allocation or simple and static predictions for reserved bandwidth allocation. Prophet is the first approach to predict the dynamic network performance for reactive flows to the best of our knowledge, especially with source constraint considerations.

The general bandwidth allocation framework for TCP has been introduced in prior studies (*e.g.*, [19]–[21]). While they have laid down the theoretical foundation of our work, Prophet takes one more step on designing an approximate but practical prediction system with the advanced monitoring capabilities.

Modeling coexisting TCP flows. Many researchers have studied the equilibrium of TCP flows when multiple different congestion avoidance algorithms coexist. Vojnovic *et al.* [30] have studied the fairness of TCP implementations based on *additive-increase and multiplicative-decrease*. Tang *et al.* [15] have proved that the equilibrium of mixed TCP flows still exists. They also give the sufficient conditions for global uniqueness of network equilibrium. Instead of developing an accurate model for coexisted TCP flows, Prophet estimates equilibrium rates for heterogeneous TCP flows based on model learning and real-time monitoring.

VII. Conclusion

In this paper, we systematically study the problem of predicting throughput for reactive flows. We propose a novel learning-based method to accurately predict the throughput for reactive flow queries, based on both theoretical models and advanced monitoring capabilities provided by SDN and NFV. A system, namely Prophet, is proposed to provide reactive flow query service for applications hosted in a data center network. Evaluations have demonstrated that our prediction method yields accurate throughput prediction for well-known TCP implementations with fast convergence and quick responses.

VIII. Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments. We have made some changes accordingly and include further discussions in an extended technical report [31] due to the limited space.

This research is sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

This work is also supported by the *National Science Foundation* (CC-IIE 1440745), *Google Research Award*, *National Key Research and Development Plan of China* (2017YFB0801701), and the *National Natural Science Foundation of China* (No. 61672385, No. 61472213 and No. 61502267).

References

[1] Amazon, "Amazon elastic compute cloud (Amazon EC2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[2] Google, "Google Cloud Computing, Hosting Services & APIs," 2017, https://cloud.google.com/.

[3] Microsoft, "Microsoft Azure Cloud Computing Platform & Services," 2017, https://azure.microsoft.com/en-us/.

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[6] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, M. Girone, C. Grandi, and others, "LHC Computing Grid," *Technical design report*, p. 8, 2005.

[7] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. O. Guedes, "Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks." in *WIOV*, 2011.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 242–253.

[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *NSDI*, vol. 11, 2011, pp. 22–22.

[10] R. Alimi, Y. Yang, and R. Penno, "RFC 7285, Application-layer traffic optimization (ALTO) protocol," 2014.

[11] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "NOVA: Towards on-demand equivalent network view abstraction for network optimization," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, Jun. 2017, pp. 1–10.

[12] CAIDA, "Analyzing UDP usage in Internet traffic," http://www.caida.org/research/traffic-analysis/tcpudpratio/index.xml.

[13] Linux, "Manual tcp (7)," 2017, http://man7.org/linux/man-pages/man7/tcp.7.html.

[14] . Budzisz, R. Stanojevic, A. Schlote, F. Baker, and R. Shorten, "On the Fair Coexistence of Loss- and Delay-Based TCP," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1811–1824, Dec. 2011.

[15] A. Tang, J. Wang, S. H. Low, and M. Chiang, "Equilibrium of Heterogeneous Congestion Control: Existence and Uniqueness," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 824–837, Aug. 2007.

[16] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Adaptive bandwidth share estimation in TCP Westwood," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, Nov. 2002, pp. 2604–2608 vol.3.

[17] S. Mascolo, C. Casetti, M. Gerla, S. Lee, and M. Sanadidi, "TCP Westwood: congestion control with faster recovery," *Univ. California, Los Angeles, Tech. Rep. CSD TR*, vol. 200017, 2000.

[18] A. Capone, L. Fratta, and F. Martignon, "Bandwidth estimation schemes for TCP over wireless networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 2, pp. 129–143, Apr. 2004.

[19] R. Srikant, *The Mathematics of Internet Congestion Control*. Springer Science & Business Media, 2012.

[20] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, pp. 237–252, 1998.

[21] S. H. Low, L. L. Peterson, and L. Wang, "Understanding TCP Vegas: A Duality Model," *J. ACM*, vol. 49, no. 2, pp. 207–235, 2002.

[22] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62.

[23] J. Oshio, S. Ata, and I. Oka, "Real-Time Identification of Different TCP Versions," in *Managing Next Generation Networks and Services: 10th Asia-Pacific Network Operations and Management Symposium, APNOMS 2007, Sapporo, Japan, October 10-12, 2007. Proceedings*, S. Ata and C. S. Hong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 215–224.

[24] ——, "Identification of Different TCP Versions Based on Cluster Analysis," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, Aug. 2009, pp. 1–6.

[25] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP Congestion Avoidance Algorithm Identification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1311–1324, Aug. 2014.

[26] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.

[27] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[28] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[29] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.

[30] M. Vojnovic, J.-Y. Le Boudec, and C. Boutremans, "Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2000, pp. 1303–1312.

[31] K. Gao, J. Zhang, Y. R. Yang, and J. Bi, "Prophet Technical Report," 2017, https://www.dropbox.com/s/tvgqx4v830c2nb0/prophet-tr.pdf?dl=0.

NOVA: Towards On-Demand Equivalent Network View Abstraction for Network Optimization

Kai Gao[†‡], Qiao Xiang[§‡], Xin Wang[§‡], Yang Richard Yang[§‡] and Jun Bi[†*]
[§] Department of Computer Science, Tongji University
[†] Institute for Network Sciences and Cyberspace, Tsinghua University
[‡] Department of Computer Science, Yale University

***Abstract*—As many applications today migrate to distributed computing and cloud platforms, their user experience depends heavily on network performance. Software Defined Networking (SDN) makes it possible to obtain a global view of the network, introducing the new paradigm of developing adaptive applications with network views. A naive approach of realizing the paradigm, such as distributing the whole network view to applications, is not practical due to scalability and privacy concerns. Existing approaches providing network abstractions are limited to special cases, such as bottlenecks exist only at networks edges, resulting in potentially suboptimal or infeasible decisions. In this paper, we introduce a novel, on-demand network abstraction service that provides an abstract network view supporting not only accurate *end-to-end QoS metrics*, which satisfy the requirements of many peer-to-peer applications, but also *multi-flow correlation*, which is essential for bandwidth-sensitive applications containing many flows to conduct global network optimization. We prove that our abstract view is *equivalent* to the original network view, in the sense that applications can make the same optimal decision as with the complete information. Our evaluations demonstrate that the abstraction guarantees feasibility and optimality for network optimizations and protects the network service providers' privacy. Our evaluations also show that the service can be implemented efficiently; for example, for an extreme large network with 30,000 links and abstraction requests containing 3,000 flows, an abstract network view can be computed in less than one second.**

I. Introduction

Software Defined Networking (SDN) is a new emerging technique. It enables a network to collect information from all the devices and construct a global view. This global view makes it possible to share with applications essential quality of service (QoS) metrics such as available bandwidth, loss rate and routing cost values, which are critical to performance of network-based services.

While northbound APIs for "apps" (management programs) to access the global view have been provided by many SDN controllers (*e.g.*, [1]–[4]), they are not open to non-administrative parties such as content providers, neighbor domains and VPN tenants, because of privacy, security and consistency concerns. Therefore, a new network abstraction providing global network view for non-administrative parties is needed.

A naive design is to only return a *slice* of the network to these non-administrative parties, or as we refer to as network *consumers*. Specifically, the *slice* can be calculated by an SDN controller which receives a query containing the flows of interest from a consumer, calculates the forwarding paths for the flows, and returns only links on the paths with associated attributes. However, this simple approach can have drawbacks. First, a slice can reveal sensitive information like network topology, leading to privacy leaks. Second, a slice may contain redundant information and introduce unnecessary communication overhead.

Thus, a problem arises on how to provide an abstract network view which can both eliminate these drawbacks and still provide high-quality information. It is non-trivial because of the following challenges:

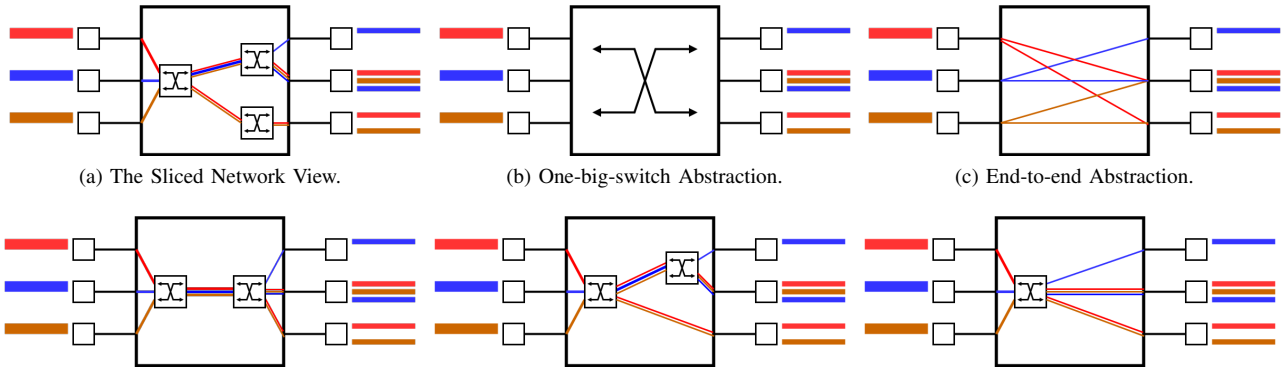
- *Feasibility*: A decision made with the abstract view should also be feasible in the original network. Infeasible solutions such as flow rate scheduling can lead to congestion and significantly affect the total throughput and also the user experience.
- *Generality*: The abstraction should be general enough to provide fine-grained information and suffice the demands of applications with heterogeneous objectives.
- *Optimality*: A decision made with the abstract view should be as optimal as with the original network information. A suboptimal solution will affect the quality of service and cannot fully utilize the network resources.
- *Privacy*: The abstraction must be able to protect the privacy of the network service provider, making it difficult for malicious consumers to infer the original information.
- *Efficiency*: The abstraction should not introduce too much computation/communication overhead, even with moderately large networks and workloads.

Existing abstractions [5]–[14] usually target at a certain type of scenario and cannot support applications which require fine-grained QoS metrics. For example, many of these abstractions do not have the ability to accurately represent bottlenecks shared by multiple correlated flows in an arbitrary network, which is critical in emerging use cases such as geo-distributed data centers [15]–[17] and scientific computing platforms [18].

In this paper, we formally define the problem of providing high-quality on-demand abstract network views and make the first step by introducing the *equivalent network view* which guarantees *feasibility*, *generality* and *optimality*. The *equivalent network view* is based on the observation that these network information are eventually used by network

* The corresponding author is Prof. Jun Bi. junbi@tsinghua.edu.cn

978-1-5386-2704-4/17/$31.00 ©2017 IEEE



(a) The Sliced Network View. (b) One-big-switch Abstraction. (c) End-to-end Abstraction.

(d) Incorrect Topology Aggregation. (e) Simple Equivalent Aggregation. (f) Advanced Equivalent Transformation.

There are 6 flows: 2 red, 2 blue and 2 brown. The flows are labeled as b_1, r_1, y_1, b_2, r_2 and y_2 from top to bottom.

Fig. 1: Comparison between Different Network View Abstractions.

consumers to conduct network optimizations.

Based on the concept of equivalent network view, we propose NOVA, the *Network Optimization View Abstraction*, which can effectively find such *equivalent network views*. Demonstrated by analysis and evaluations, NOVA can also achieve good *privacy preservation* and *efficiency*.

Our main contributions in this paper can be summarized as follows:

- We systematically investigate the problem of providing on-demand view abstraction for arbitrary application-layer network optimization, which is a missing functionality from current SDN northbound API design. We discuss the potential benefits for such a new functionality and identify its challenges.
- We propose NOVA to address the challenges, the *Network Optimization View Abstraction*, which is based on the concept of *equivalent network view*. We prove our equivalent transformation algorithms can effectively generate *equivalent network views* which can achieve feasibility, generality and optimality, while protecting the network privacy.
- We implement a prototype of NOVA and evaluate its performance using real topologies. Evaluations show that NOVA guarantees both feasibility and optimality, improves privacy by reducing information leak, and reduces the communication overhead by a factor of 1.25 to 5 even for large networks, for example, real ISP network topologies with up 10000 nodes and 30000 links, and large workloads (more than 3000 flow requests).

The rest of the paper is organized as follows. We summarize the demands for fine-grained network views, existing abstractions and their limitations in Section II. A formal description of the abstraction problem and the *equivalent network view* are then given in Section III, followed up by the algorithms in Section IV. We evaluate the prototype and analyze the results in Section V. Finally, we discuss the related work and give conclusions in Section VI and Section VII respectively.

II. Motivation

In this section we discuss the motivations that drive our research. See the network in Figure 1a, assume on the left-hand side, a web service provider has three services colored in red, blue and brown respectively while on the right-hand side there are six clients using different services. Assume the red service is live streaming, blue is video subscribing and brown is large file downloading. All three services require bandwidth so it's important to know about the bottlenecks in the network and the content provider sends a request on the bandwidth correlation of the six flows to the network. Meanwhile, the content provider does not want the network to know about how it would manage the services so it does not provide any further information.

The naive approach returns the slice containing all the links on the flow paths with the associated bandwidth information and how the links are shared by the flows, denoted by Figure 1a in this case. However, this can lead to information leaks so that malicious network consumers may leverage this service to infer the network information, which jeopardizes the privacy of the network provider. Also as the network size increases, the topology cannot be effectively represented.

The hose model, also known as the *one-big-switch* abstraction, returns the network as a single big switch as demonstrated in Figure 1b. However, the content provider would only know about the available bandwidth on the ingress/egress "port". If the bottleneck is the upper middle link, it is not propagated to the content provider. Thus, the content provider may incorrectly increase the traffic for the blue flows without knowing that they are correlated with r_1, which leads to congestion. Because of the TCP congestion control mechanism, congestion would not only affect the r_1, jeopardizing the overall objective to maximize the weighted quality of service, but also the throughput of the other flows sharing the same link, leading to an overall performance degradation.

The end-to-end abstraction as demonstrated in Figure 1c is barely useless in this case. It has the same problem as the *one-*

big-switch abstraction that information about the bottleneck within the network cannot be accurately provided to the consumer.

Topology aggregation [13] is a common technique in reducing the topology size. However, it also suffers from the incapability of providing accurate information about the flow correlations. What is worse, if not aggregated correctly as in Figure 1d, it may introduce unnecessary bottlenecks that lead to suboptimal decisions right in the beginning.

A simple observation is that the lower middle link and the lower right link are both shared by r_2 and y_2 only. Thus, we can aggregate them together as a new virtual link, as demonstrated in Figure 1e. One may think we can just delete one of them. However, if the content provider asks about the end-to-end routing metrics such as hop count at the same time, deletion would return incorrect values for the two flows.

At the same time, if we already know that the upper middle link would not be the bottleneck, the network view can be further reduced as demonstrated in Figure 1f. It is worth noticing that just like the case with the simple equivalent aggregation, we cannot just delete it if end-to-end metrics are also requested.

Thus, the question arises on how we can determine what kind of links can be reduced and how to reduce them correctly. In order to answer this question, we introduce the concept of *equivalent network view* and propose NOVA, the *Network Optimization View Abstraction*, with efficient algorithms to compute equivalent network view.

III. Equivalent Network View

In this section, we formally define *equivalent network view* after introducing its theoretical foundations – the variant routing metric algebra, and the unified network element. We prove its properties and demonstrate that it guarantees *generality*, *feasibility* and *optimality*.

A. The variant routing metric algebra

The routing metric algebra is introduced by Sobrinho to compute QoS-based routing [19]. The routing metric algebra is based on *path concatenation*. The routing metric algebra consists of the following operations: the *weight* function w, the concatenation operator $\circ$, the "plus" operator $\oplus$ and a binary relation $\preceq$.

The original routing metric algebra system can be represented as $(P, S, w, \circ, \oplus, \preceq)$. P is the set of paths and S is a closed set of metrics. The weight function w maps a path from P to a given metric in S. The concatenation operator $\circ$ is a binary operator on P, which takes two paths and returns a new one. Operator $\oplus$ is a binary operator on S and $\preceq$ is a binary relation on S.

In this paper, we introduce a variant of the routing metric algebra. First, to better formulate the constraints on *flow-independent* metrics, we introduce a new $\otimes : \mathbb{R} \times S \mapsto S$ operator to linearize the metric calculation. Second, we relax the constraint on path concatenation in [19], by extending the meaning of P from the set of paths to the set of *unified network elements* as defined in Section III-B. The new metric algebra can be described as $(P, S, w, \circ, \oplus, \preceq, \otimes)$. Concrete examples of some common routing metrics are demonstrated in Table I.

$(S, \oplus)$ is a semigroup so that $\oplus$ is *commutative* and *associative*. We also require that the $\otimes$ operator is *distributive* over $\oplus$. It can be easily proved that all the examples listed in Table I satisfy these requirements.

B. Unified network elements

Traditional graph representations of a network would treat links and nodes differently because the routing capability is only provided on nodes (switches/routers/middleboxes). However, since routing is not a mandatory functionality in our network view definition, we can generalize the nodes and links as *unified network elements* to simplify the representation.

For example, a deep packet inspection (DPI) middlebox may have a maximum processing speed, which yields the constraint on the total throughput passing through this DPI node. From the consumers' perspective, it has no difference as a shared bottleneck link. To guarantee that these unified network elements would not affect the results of routing metric algebra, we must alter the weight function w as:

$$w^*(p) = \begin{cases} w(p) & \text{if } w(p) \text{ exists} \\ e & \text{otherwise} \end{cases}$$

where e is the *identity* of w and some concrete examples are given in Table I.

Unified network elements have several benefits. First, this unified representation of links/nodes greatly simplifies our analysis. Second, it provides a high-level abstraction which focuses on the routing metric semantics and allows consumers to ignore how the metrics are computed/constrained.

C. Network view abstraction

We identify two kinds of routing metrics that are essential for network optimization:

- *Flow-independent* metrics represent those metrics whose value is *independent* of the flow correlations, in the sense that the existence of a flow would not affect the value of another flow's flow-independent metric sharing the same network element. Common flow-independent metrics used in QoS routing [20] include hop count, local link preference, delay, jitters and loss rate[1].
- The *flow-correlated* metrics represents the network resources that are shared among flows. Bandwidth is the most common and also the most important *flow-correlated* metric. Other shared resources such as flow entries or middlebox-related metrics may also exist in certain scenarios.

Flow-independent metrics are formulated as *equations* according to the routing metric algebra introduced in Section III-A. For example, the hop count between two end hosts is equal to the sum of the all hop count on each link in the

[1]Delay, jitters and loss rate are sensitive to traffic volumes so their real time values are not flow independent. However, they are considered flow independent when measured *statistically*.

TABLE I: The Variant Routing Metric Algebra.

S	Weight function (w)	$w(p_1)$	$w(p_2)$	$w(p_1 \circ p_2) = w(p1) \oplus w(p_2)$	$N \otimes w(p_1)$	Identity (e)	Zero ($\mathbf{0}$)
$\mathbb{N}^+$	Hopcount	h_1	h_2	$h_1 + h_2$	$N \cdot h_1$	0	$+\infty$
$\mathbb{R}^+$	Bandwidth	b_1	b_2	$\min(b_1, b_2)$	b_1	$+\infty$	0
$\mathbb{R}^+$	Delay	d_1	d_2	$d_1 + d_2$	$N \cdot d_1$	0	$+\infty$
$[0, 1]$	Loss rate	r_1	r_2	$1 - (1 - r_1)(1 - r_2)$	$1 - (1 - r_1)^N$	0	1

TABLE II: Symbols for Network View Abstraction.

Symbol	Meaning
V	Network view represented by 4-tuple
$V_i(\boldsymbol{u}_i)$	The i-the component (and its key matrix) of V
r_{ij}	Indicator of whether element j appears in flow i's path
$p_j^k(\hat{p}_i^k)$	k-th flow-independent metric of element j (flow i)
a_{ji}^k	Proportion of f_i's correlated metric k on element j
$q_j^k(\hat{q}_i^k)$	k-th flow-correlated metric of element j (flow i)
$\mathcal{E}$	Objective function of the consumer
$\mathcal{A}$	Abstraction function
V'	Abstract view computed by $\mathcal{A}(V)$

path. Let $\boldsymbol{p}_j$ represent the metrics on element j, $\hat{\boldsymbol{p}}_i$ represent the metrics for flow i, and $r_{ij}(r_{ij} \in \{0, 1\})$ represent whether element j appears in the path of flow i, we have:

$$\hat{\boldsymbol{p}}_i = \left(\bigoplus_j r_{ij} \otimes p_j^1, \ \ldots \ , \bigoplus_j r_{ij} \otimes p_j^s \right) = \boldsymbol{R}_i \times \boldsymbol{P}$$

If the k-th *flow-correlated* metric (resource) flow i consumes on element j is proportional to the total amount of resources consumed by this flow and the proportion is independent of flow correlations, the *flow-correlated* metrics are formulated as *linear constraints*. The sum of the resource consumption of all the flows on a single element must not exceed the available amount. Let $\boldsymbol{q}_j$ represent the available resources on element j, $\hat{\boldsymbol{q}}_i$ represent the resource consumed by flow i, and $a_{ji}^k (\geq 0)$ represent the proportion of the k-the kind of resource consumed by flow i on element j, we have:

$$\sum_i a_{ji}^k \hat{q}_i^k \leq q_j^k \Leftrightarrow \boldsymbol{A}^k \hat{\boldsymbol{q}}^k \leq \boldsymbol{q}^k$$

Thus, the network view can be formulated as a tuple with four elements: $V = (\boldsymbol{R}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{Q})$. Based on this network view model, an *abstraction* can be defined as below:

Definition 1 (Network View Abstraction)**.** The network view abstraction is a transform function $\mathcal{A}$ which takes the original network view V and returns the *abstract* view V', i.e.,

$$V'(\boldsymbol{R'}, \boldsymbol{P'}, \boldsymbol{A'}, \boldsymbol{Q'}) = \mathcal{A}\left(V(\boldsymbol{R}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{Q})\right)$$

D. Equivalent network view

A key insight of the network view abstraction is that the returned information is usually the input parameters of a specific algorithm, whose result can help applications make decisions. If the applications can make the same optimized decision with the abstract network view, we can say the abstract network view is *equivalent*.

Consider a consumer of the network view whose optimization problem consists of a set of flows $\{f_1, \ldots, f_n\}$. For flow i, the consumer queries its *flow-independent* metrics $\hat{\boldsymbol{p}}_i = (\hat{p}_i^1, \ldots, \hat{p}_i^s)$, and its *flow-correlated* metrics $\hat{\boldsymbol{q}}_i = (q_i^1, \ldots, q_i^t)$. The optimization problem can be formulated as

$$\begin{aligned} & \min \mathcal{E}\left(\{\hat{\boldsymbol{p}}_i\}, \{\hat{\boldsymbol{q}}_i\}\right) \\ & \hat{\boldsymbol{P}} = \boldsymbol{R} \times \boldsymbol{P} \\ \text{s.t.} \quad & \boldsymbol{A}^k \hat{\boldsymbol{q}}^k \leq \boldsymbol{q}^k \\ & \boldsymbol{R} \geq 0, \boldsymbol{A}^k \geq 0, \boldsymbol{q}^k \geq 0, \hat{\boldsymbol{q}}^k \geq 0 \end{aligned}$$

We use the symbols in Table II and define the equivalent network view as the following:

Definition 2 (Network View Equivalence)**.** Two network views V_1 and V_2, V_1 is *equivalent* to V_2 if and only if for any consumer with flows $\{f_i\}$, its objective function $\min \mathcal{E}\left(\{\hat{\boldsymbol{p}}_i\}, \{\hat{\boldsymbol{q}}_i\}\right)$ achieves the same optimal objective.

We use the symbol "$\sim$" to represent the network view equivalence. It can be easily proved that the network view equivalence is an *equivalence relation*. We also propose the criterion in Theorem 1 to simplify the verification.

Theorem 1 (Network View Equivalence Criterion)**.** Two network views V_1 and V_2, V_1 is *equivalent* to V_2 if and only if for any consumer with flows $\{f_i\}$ and metrics $\{\hat{\boldsymbol{p}}_i\}, \{\hat{\boldsymbol{q}}_i\}$:

$$\boldsymbol{R_1} \times \boldsymbol{P_1} = \boldsymbol{R_2} \times \boldsymbol{P_2} \tag{1a}$$

$$F_1^k = \left\{\boldsymbol{x} \mid \boldsymbol{A_1^k x} \leq \boldsymbol{q_1^k}\right\} = \left\{\boldsymbol{x} \mid \boldsymbol{A_2^k x} \leq \boldsymbol{q_2^k}\right\} = F_2^k \tag{1b}$$

Proof. **Sketch**: For two network views V_1 and V_2, we use $V_1 \sim^* V_2$ and $V_1 \sim V_2$ to represent that V_1 and V_2 are equivalent by the criterion and are equivalent by definition respectively. It can be easily proved that if $V_1 \sim^* V_2$, $V_1 \sim V_2$. For the other direction, we prove it by contradiction.

We first assume that there exists $V_1 \sim V_2$ but $V_1 \nsim^* V_2$, i.e., either Equation 1a or Equation 1b does not hold.

Assume Equation 1a does not hold. For any $\hat{\boldsymbol{P}}_\mathbf{1} = \boldsymbol{R_1} \times \boldsymbol{P_1} \neq \boldsymbol{R_2} \times \boldsymbol{P_2} = \hat{\boldsymbol{P}}_\mathbf{2}$, we find one entry that is not equal in $\hat{\boldsymbol{P}}_\mathbf{1}$ and $\hat{\boldsymbol{P}}_\mathbf{2}$, say $\hat{p_1}_i^k \neq \hat{p_2}_i^k$ and construct an objective function which use the $\hat{p}_i^k$ as the objective value. Thus, for the objective function the two network views yield different optimal objectives and they are not equivalent by definition, which contradicts with our assumption.

Assume Equation 1b does not hold, $\exists \boldsymbol{x_0} \in (F_1 \setminus F_2) \cup (F_2 \setminus F_1)$. Without loss of generality, let $\boldsymbol{x_0} \in F_1 \setminus F_2$. Since $\boldsymbol{x_0} \notin F_2$, there exist j, k such that $\boldsymbol{A_2}_j^k \boldsymbol{x_0} = y_0 > q_j^k$. Now

we construct a linear objective function in the standard form $\min -\boldsymbol{A_2}_j^k \boldsymbol{x}$, and assume the optimal objective is y_1 and y_2 respectively. We have $y_1 \geq y_0 > q_j^k = y_2$ which means the objective function yields different optimal objective values for the two network views. Again, we get a contradiction.

Thus, we can conclude that if $V_1 \sim V_2$, $V_1 \sim^* V_2$ and the criterion is both sufficient and necessary. □

We have proved that the network views satisfying this criterion also satisfy Definition 2 and can provide the *equivalent network view* for consumers with arbitrary objective functions and arbitrary fine-grained routing metrics as long as they fit in the *variant routing metric algebra*.

IV. Network Optimization View Abstraction

In this section, we introduce NOVA, the *Network Optimization View Abstraction* which conducts equivalent transformations to obtain the equivalent network view. NOVA consists of two algorithms, namely the *equivalent element aggregation* and the *equivalent element decomposition*. We prove both algorithms guarantee the *equivalence* condition, and analyze how they can improve *efficiency* and *privacy* as well.

To help simplify the analysis, we assume the request contains n flows with s flow-independent metrics and t flow-correlated metrics, while the corresponding original network view contains m elements.

A. Equivalent element aggregation

In this section, we introduce the *equivalence aggregation*. The intuition is that, as demonstrated with Figure 1e, the elements shared by the same set of flows and with the same coefficients in the network constraints can be combined into a single element. The algorithm is given in Algorithm 1 and we analyze its efficiency and prove its correctness.

Algorithm 1: NOVA Equivalent Element Aggregation

Input: $V(\boldsymbol{R}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{S})$
Output: $V'(\boldsymbol{R'}, \boldsymbol{P'}, \boldsymbol{A'}, \boldsymbol{S'})$
1 **Function** EquivAggregation*(V)*
2 $\quad \mathcal{V} \leftarrow \left\{V_i \mid V_i \leftarrow \left(\boldsymbol{R^T}_i, \boldsymbol{P}_i, \{\boldsymbol{A}_i^k\}, \{\boldsymbol{Q}_i^k\}\right), 1 \leq i \leq m\right\}$
3 $\quad \mathcal{G} \leftarrow$ GroupBy$(\mathcal{V}, V_i \Rightarrow \left(k_i \leftarrow (\boldsymbol{R^T}_i, \{\boldsymbol{A}_i^k\}), V_i\right))$
4 $\quad$ **for** $G_i \in \mathcal{G}$ **do**
5 $\quad\quad V_i' \leftarrow$ Aggregate$(k_i, \{V_{a_j^i} \in G_i\})$
6 $\quad V' \leftarrow$
$$\left(\left[\boldsymbol{R'^T}_1 \cdots \boldsymbol{R'^T}_{m'}\right], \begin{bmatrix} \boldsymbol{p'}_1 \\ \vdots \\ \boldsymbol{p'}_{m'} \end{bmatrix}, \begin{Bmatrix} \boldsymbol{A'}_1^k \\ \vdots \\ \boldsymbol{A}_{m'}^k \end{Bmatrix}, \begin{Bmatrix} \boldsymbol{q'}_1^k \\ \vdots \\ \boldsymbol{q'}_{m'}^k \end{Bmatrix}\right)$$
$\quad$ **return** V'
7 **Function** Aggregate*(*$k_i, \{V_{a_j^i}\}$*)*
8 $\quad (\boldsymbol{R'^T}_i, \{\boldsymbol{A'}_i^k\}) \leftarrow k_i$
9 $\quad \boldsymbol{p'}_i \leftarrow [\bigoplus p_{a_j^i}^1, \ldots, \bigoplus p_{a_j^i}^s]$
10 $\quad \boldsymbol{q'}_i \leftarrow \left\{q_i'^k \mid q_i'^k \leftarrow \min\{q_{a_1^i}^k, \ldots, q_{a_t^i}^k\}, \forall k\right\}$
11 $\quad$ **return** $\left(\boldsymbol{R'^T}_i, \boldsymbol{p'}_i, \{\boldsymbol{A'}_i^k\}, \{\boldsymbol{q'}_i^k\}\right)$

The network view is represented as row vectors (components), as demonstrated in Line 2. Line 3 groups the i-th *component* $V_i(\boldsymbol{R}_i^T, \boldsymbol{P}_i, \{\boldsymbol{A}_i^k\}, \{\boldsymbol{Q}_i^k\})$ using a unified row vector $\boldsymbol{u}_i = \left[\boldsymbol{R^T}_i^T, \boldsymbol{A}_i^1, \ldots, \boldsymbol{A}_i^t\right]_{1\times(n+nt)}$ as the key. Line 5 computes the aggregation of the components in each group. Finally Line 6 constructs the new network view by merging all the aggregated components. For each component V_i, the time complexity for the grouping and the aggregation is $O(n(s+t))$ and $O(n(s+t))$ respectively while the Merge process is totally logical, which yields a total time of $O(mn(s+t))$.

Now we prove the element aggregation algorithm is correct, in the sense that it maintains the equivalence condition.

Theorem 2. $V' \leftarrow$ EquivAggregation(V), $V' \sim V$.

Proof. **Sketch**: Assume $\boldsymbol{a_i}$ represents the index of the components in G_i, and let $b_i \leftarrow \min a_{ij}$ and $c_i^k \leftarrow \arg\min_{j\in\boldsymbol{a_i}} q_j^k$.

First we check Equation 1a is met. Let $\boldsymbol{M} = \boldsymbol{R} \times \boldsymbol{P}$, we have

$$m_{ij} = \bigoplus_k r_{ik} \otimes p_k^j = \bigoplus_{1\leq k\leq m'} \left\{\bigoplus_{u\in\boldsymbol{a_k}} r_{iu} \otimes p_u^j\right\}$$
$$= \bigoplus_{1\leq k\leq m'} r_{ib_k} \otimes \left\{\bigoplus_{u\in\boldsymbol{a_k}} p_u^j\right\} = \bigoplus_{1\leq k\leq m} r_{ib_k} \otimes p'^j_u = m'_{ij}$$

The key steps are based on that $\oplus$ is transitive and commutative, $\otimes$ is distributive over $\oplus$, and $\forall u \in \boldsymbol{b}_k, r_{iu} = r_{ia_k}$.

Now we check Equation 1b. For any k, we have

$$F^k = \left\{\boldsymbol{x} \mid \boldsymbol{A^k}_i \boldsymbol{x} \leq \boldsymbol{q^k}_i, \forall i\right\}$$
$$F'^k = \left\{\boldsymbol{x} \mid \boldsymbol{A'^k}_i \boldsymbol{x} \leq \boldsymbol{q'^k}_i, \forall i\right\} = \left\{\boldsymbol{x} \mid \boldsymbol{A^k}_{c_i} \boldsymbol{x} \leq \boldsymbol{q}_{c_i}^k, \forall i\right\}$$

Since the constraints of F'^k is a subset of F^k, $F^k \subseteq F'^k$. If $F'^k \neq F^k$, $\exists \boldsymbol{x_0} \in F'^k \setminus F^k$, meaning $\boldsymbol{x_0}$ must at least violates one constraint in F^k, say $\boldsymbol{A^k}_{d_i}$ where $d_i \in \boldsymbol{a_i}$. Thus, we have

$$\boldsymbol{A^k}_{d_i} \boldsymbol{x_0} > q_{d_i}^k \geq \min_{j\in\boldsymbol{a_i}} q_j^k = q_{c_i}^k$$

which means $\boldsymbol{x_0}$ also violates one constraint in F'^k and leads to contradiction with our assumption. So we have $F^k = F'^k$.

By Theorem 1, $V' \sim V$. □

B. Equivalent element decomposition

In this section, we introduce the motivations for *equivalent element decomposition* which can substantially improve the performance of *equivalent element aggregation*.

Algorithm 1 guarantees the equivalence condition which is important to prove the correctness of NOVA, however, the condition to aggregate components is not easy to be satisfied without further processing. Thus, in practice we need to conduct another equivalent transformation, namely *equivalent element decomposition*. The intuition of this algorithm can be demonstrated using the simple example below:

$$a : \textit{routingcost} = 1, \textit{bandwidth} = 100\text{Mbps}$$
$$b : \textit{routingcost} = 2, \textit{bandwidth} = 100\text{Mbps}$$
$$c : \textit{routingcost} = 3, \textit{bandwidth} = 200\text{Mbps}$$
$$\boldsymbol{R}_a^T = \boldsymbol{A}_a = \begin{bmatrix}1 & 0\end{bmatrix}^T, \boldsymbol{R}_b^T = \boldsymbol{A}_b = \begin{bmatrix}0 & 1\end{bmatrix}^T, \boldsymbol{R}_c^T = \boldsymbol{A}_c = \begin{bmatrix}1 & 1\end{bmatrix}^T$$

According to grouping condition, there will be three different groups. But we can make the observation that since the constraint for c: $bw_1 + bw_2 \leq 200$ is *redundant*, we can decompose c as two unified network elements c_1 and c_2 where

$$c_1 : \textit{routingcost} = 3, \textit{bandwidth} = 200\text{Mbps}$$
$$c_2 : \textit{routingcost} = 3, \textit{bandwidth} = 200\text{Mbps}$$
$$\boldsymbol{R}_{c_1}^T = \boldsymbol{A}_{c_1} = \begin{bmatrix}1 & 0\end{bmatrix}^T, \boldsymbol{R}_{c_2}^T = \boldsymbol{A}_{c_2} = \begin{bmatrix}0 & 1\end{bmatrix}^T$$

After c is decomposed, we can invoke EQUIVAGGREGATION (Algorithm 1) and c_1 and c_2 can be aggregated with a and b respectively. We introduce the definition of *constraint redundancy* by Telgen [21] as Definition 3 and further prove that the decomposition guarantees equivalence.

Definition 3 (Redundant linear constraint – Telgen [21])**.** For a linear system whose feasible region $F = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, the k-the constraint $\boldsymbol{A}_k\boldsymbol{x} \leq b_k$ is redundant if and only if the feasible region $F_k = \{\boldsymbol{x} \mid \boldsymbol{A}_i\boldsymbol{x} \leq b_i, i \neq k\}$ is equal to F, i.e. $F_k = F$.

Theorem 3. For $V_i(\boldsymbol{R}_i^T, \boldsymbol{P}_i, \{\boldsymbol{A}_i^k\}, \{\boldsymbol{Q}_i^k\})$, we say V_i is redundant if and only if $\boldsymbol{A}_i^k\boldsymbol{x} \leq \boldsymbol{q}_i^k$ is redundant for all k. If and only if V_i is redundant, we can construct an equivalent network view $V' = V \setminus V_i \cup \{V_i^j\}$ where V_i is decomposed as $V_i^{(j)}(\boldsymbol{R}_i^{Tj}, \boldsymbol{P}_i^{(j)}, \emptyset, \emptyset)$ with $\boldsymbol{R}_i^T = \sum_j \boldsymbol{R}_i^{T(j)}$ and $\boldsymbol{P}_i^{(j)} = \boldsymbol{P}_i$.

Proof. **Sketch**: We still consider the criteria Equation 1a and Equation 1b and use the same symbols in Theorem 2.

First we can prove criterion Equation 1a holds whether V_i is redundant or not.

$$\begin{aligned} m_{uv} &= \bigoplus_k r_{uk} \otimes p_k^v = \bigoplus_{k \neq i} r_{uk} \otimes p_k^v + r_{ui} \otimes p_k^v \\ &= \bigoplus_{k \neq i} r_{uk} \otimes p_k^v + \left(\sum_j r_{ui}^{(j)}\right) \otimes p_k^v \\ &= \bigoplus_{k \neq i} r_{uk} \otimes p_k^v + \bigoplus r_{ui}^{(j)} \otimes p_k^{v(j)} = m'_{uv} \end{aligned}$$

For Equation 1b, first we consider the case when V_i is redundant but $V \nsim V'$. V_i is redundant so that $\forall k$, $\boldsymbol{A}_i^k\boldsymbol{x} \leq \boldsymbol{q}_i^k$ is redundant. According to Definition 3, we have the feasible regions $F^k = F_i^k = F'^k$ for all k. Since we have already proved that Equation 1a holds, according to Theorem 1, $V \sim V'$ which leads contradiction.

If V_i is not redundant but $V \sim V'$, we can similarly construct a contradiction between the definition of redundancy and the equivalency criterion.

Thus, we have proved that V_i can be equivalently decomposed if and only if V_i is redundant. □

The efficiency and privacy of equivalent decomposition depend on 1) how to identify redundant components, and 2) how to find the basis. In this paper, we use a heuristic approach which aims to simplify the selection of basis, as introduced in Algorithm 2.

Line 2 identifies the set of decomposable components V_d according to Theorem 3, i.e. $\forall v_i \in V_d$, v_i is redundant. The algorithm then decomposes these redundant components to a unit basis $\{V_j^{(i)} \mid r_{ji} \neq 0\}$ in Line 4-11. According to Theorem 3, V' after each iteration is *equivalent* to the original network view V. Finally we invoke EQUIVAGGREGATION(V') to aggregate the V_j^i with the same $\boldsymbol{R}_i^T$, which is also proved to maintain the equivalence condition as in Theorem 2. Thus, Algorithm 2 returns the equivalent network view.

For each iteration, the decomposition takes $O(n)$ time to check the condition in Line 8 and $O(ns)$ to construct the corresponding basis. The algorithm would at most have m iterations so the total execution time for decomposition is $O(mns)$ and the total execution time with EQUIVAGGREGATION is $O(mn(s+t))$.

Different algorithms exist to find the decomposable components, based on Theorem 3. For example, one can find all elements with non-redundant linear constraints, which is a well-studied problem, and get the decomposable set by calculating its inverse.

C. Privacy preservation

The equivalent aggregation and equivalent decomposition are equal to matrix factorization. While the consumer can only infer the network elements which cannot be decomposed without jeopardizing feasibility or optimality, it is impossible to infer the complete original network state without knowing the exact value of the transform matrix. Thus, Algorithm 2 can improve the privacy preservation and reduce information leak. It is noticeable that compared with some other optimization frameworks, NOVA does not require consumers to specify private information, e.g. private constraints and objective functions, which also protects the privacy of the consumers.

V. EVALUATION

In this section, we evaluate NOVA to demonstrate its efficiency and efficacy in providing accurate on-demand network views in comparison to some other abstraction models.

Algorithm 2: NOVA Equivalent Element Decomposition

Input: $V(\boldsymbol{R}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{Q})$, F
Output: $V'(\boldsymbol{R'}, \boldsymbol{P'}, \boldsymbol{A'}, \boldsymbol{Q'})$
1 **Function** EQUIVDECOMPOSITION(V, F)
2 $V_d \leftarrow$ FINDEQUIVDECOMPOSABLE(V)
3 $V' \leftarrow V$
4 **foreach** $V_j \in V_d$ **do**
5 $S \leftarrow \emptyset$
6 **foreach** $f_i \in F$ **do**
7 $\boldsymbol{R}_i^T \leftarrow \left[\underbrace{0, \ldots, 0}_{i-1}, r_{ji}, 0, \ldots\right]$
8 **if** $r_{ij} = 1$ **then**
9 $V_j^{(i)} \leftarrow \left(\boldsymbol{R}_i^T, \boldsymbol{P_j}_i, \emptyset, \emptyset\right)$
10 $S \leftarrow S \cup \left\{V_j^{(i)}\right\}$
11 $V' \leftarrow (V' \setminus V_j) \cup \{S\}$
12 $V' \leftarrow$ EQUIVAGGREGATION(V')
13 **return** V'

A. Methodology

Performance metrics. We evaluate the performance of NOVA using the following metrics:

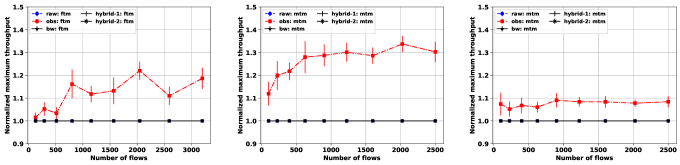
- *Optimality* and *feasibility*: To demonstrate optimality and feasibility, we generate random linear objective functions maximizing the weighted throughput [22] to demonstrate its generality and compare the results.
- *Communication overhead reduction*: We measure the communication overhead by the number of (*flow, element*) pairs contained in an network view so that smaller numbers represents better reduction.
- *Privacy preservation*: We measure the privacy preservation by the ratio of network elements in the original network view that still appear in the abstract network view. Smaller numbers represents less information leak and better privacy. As being said in Section IV-C, the non-redundant network elements cannot be reduced, we specifically evaluate the preservation of redundant elements.
- *Computation overhead*: We measure the computation overhead by the execution time of the abstraction process so that smaller numbers represents better performance.

Topology. Three topologies are used in this evaluation: Kdl (752 nodes, 1790 links), AS4755 (531 nodes, 1428 links) and AS2914 (10820 nodes, 32844 links) from two data sets: the topology zoo [23] and rocketfuel [24]. If the topology already has bandwidth information, we use it directly. Otherwise, we generate stepped values for links from edge to core. We allocate the routingcost randomly following the standard distribution around the reciprocal of bandwidth multiplied by a given constant to avoid precision issues.

Redundancy check algorithms. We use two redundancy check algorithms in our evaluation. The first, denoted as *strict redundancy check*, follows Definition 3 and can find the minimal set of non-redundant linear constraints. The second, denoted as *relaxed*, identifies redundancy by randomly selecting basis and comparing the bound with the sum of the basis and may lead to *false negative* in identifying redundancies.

Flow requests. We use two traffic patterns: *few-to-many* (*ftm*) and *many-to-many* (*mtm*). The first represents the server-client traffic pattern while the second represents the peer-to-peer traffic pattern. For each pattern, we have 9 groups with different number of flows and for each group we randomly generate 3 flow requests for each topology. The flow requests are computed with *bandwidth-only* (*bw*), *routingcost-only* (*rc*) and *hybrid* (two variants *hybrid-1* and *hybrid-2*) respectively. For *bandwidth-only* requests, we use the *strict* redundancy check. The other requests uses Algorithm 2 with different redundancy check algorithms: no check for *routingcost-only*, *strict* redundancy check for *hybrid-1* and *relaxed* redundancy check for *hybrid-2* to demonstrate the effect of how redundancy check algorithms on the performance of NOVA.

Environment and data collection. The prototype system is built with Python and uses the PuLP framework and *COIN Branch and Cut* (CBC) solver to solve linear programming.



(a) Kdl, *ftm*, normalized. (b) Kdl, *mtm*, normalized. (c) 2914, *mtm*, normalized.

Fig. 2: Normalized Maximum Weighted Throughput.

The evaluations are conducted on a laptop with Linux kernel 4.9.6, 4 quad-core Intel(R) Core(TM) i7-4700MQ @2.40GHz CPU and 16 GB memory. For each topology, we generate three different routing cost distributions and three different flow requests of the same flow size. The data are collected from the same execution.

B. Optimality and feasibility

The normalized results for using different views to solve the same random linear programming problems are compared with the original network view (*raw*) and the *one-big-switch* (*obs*). Since queries with only routing cost would not generate any linear constraints, it is omitted in this evaluation.

As demonstrated in Figure 2, we can see that NOVA *always* achieves the same optimal solution as with the original network view while the *one-big-switch* abstraction results in infeasible solutions. The reason is that *one-big-switch* abstraction does not identify the bottleneck links inside the network.

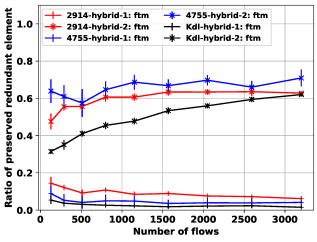
C. Privacy preservation

To demonstrate how much information consumers can learn about the original network, we use the number of preserved links to measure privacy preservation as specified in Section V-A. As demonstrated in Figure 3, we can see that despite the non-redundant links which cannot be hidden without jeopardizing equivalence, NOVA with strict redundancy checks can hide almost all the redundant links.

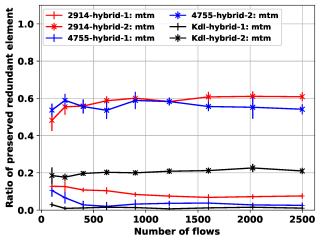
We have identified three effective factors on the privacy preservation of NOVA: 1) the redundancy check criterion, 2) flow patterns, and 3) the number of flows.

As demonstrated in Figure 3, we can see that the effect of privacy preservation is mostly determined by the redundancy check algorithm. In both traffic patterns, *hybrid-1* using strict redundancy check preserves very few links in the abstract network view (less than 10% in all three topologies) that it almost overlaps with the theoretical optimal ratio denoted by the *bandwidth-only*, while *hybrid-2* generally preserves more redundant links. The reason is that the relaxed redundancy check used in *hybrid-2* has false negative results and those redundant links are not decomposed.

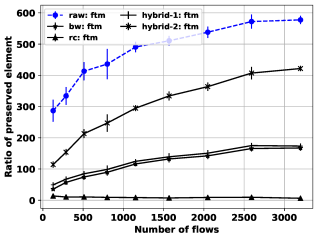
Traffic patterns have slightly less impact on the privacy preservation but we can still observe that for traffic with the *few-to-many* pattern, more links are preserved by *hybrid-2*. This is because in the *few-to-many* traffic pattern, the flows sharing a link on the "*many*" side would diverge to different paths and have no further correlation. Thus, even the link is



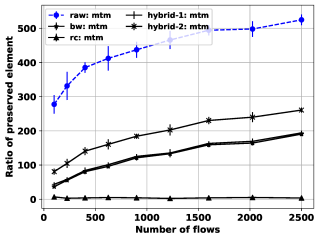
(a) *few-to-many*, normalized.



(b) *many-to-many*, normalized.

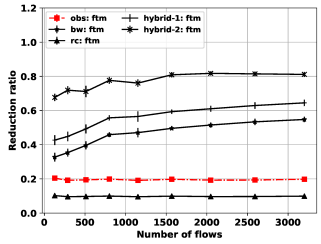


(c) Kdl, *many-to-many*, absolute.

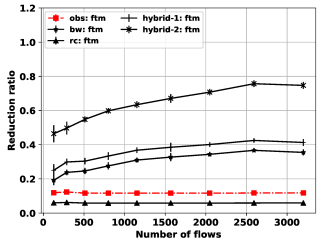


(d) Kdl, *many-to-many*, absolute.

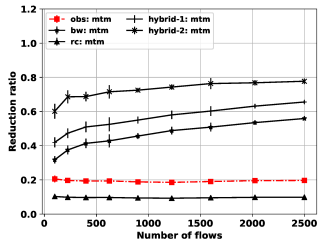
Fig. 3: Preserved links in abstract network view.



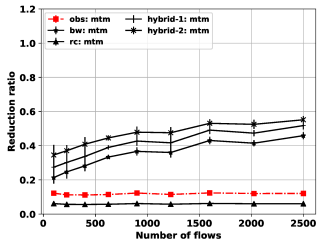
(a) 2914, *few-to-many*, normalized.



(b) Kdl, *few-to-many*, normalized.



(c) 2914, *many-to-many*, normalized.



(d) Kdl, *many-to-many*, normalized.

Fig. 4: Communication overhead reduction.

unlikely to become the bottleneck, it still cannot be identified by simple redundancy check algorithms.

The number of flows would affect the privacy preservation, as demonstrated in Figure 3c and Figure 3d. It is intuitive since more flows can generate more combinations of correlation, and reveal more information about the network. However, even with relatively large flow requests (more than 2500 flows), NOVA can protect as much as 60% of the original sliced network view in both traffic patterns using the strict redundancy check. It is also worth pointing out that when the request only contains *flow-independent* metrics, NOVA will fall back to the *end-to-end* abstraction.

D. Communication overhead reduction.

As analyzed in Section IV-A and Section IV-B, while NOVA guarantees feasibility, optimality and protects privacy, it can also reduce the communication overhead. The communication overhead is measured by the number of (*flow, element*) pairs in the network view, and the results are normalized by the value of the original network view. As demonstrated in Figure 4, we can see that depending on the effective factors, NOVA can shrink the communication overhead by a factor of 1.25 to 5.

We can see that the communication overhead reduction is affected by the same effective factors – the redundancy check algorithm, the traffic pattern and the number of flows – in a similar way as privacy preservation. The reason is that the communication overhead is mostly reduced by aggregating and decomposing redundant links. As the ratio of non-redundant links grows, the reduction is less obvious.

We can see that the theoretical lower bound (result of *bandwidth-only*) of communication overhead reduction is larger than that of privacy preservation. The reason is that most bottlenecks are usually shared by more flows, so the average number of flows on the preserved links is larger than the average in the whole slice. The gap between *hybrid-1* and *bandwidth-only* in communication overhead reduction is larger than that in privacy preservation, because flows are sent to multiple receivers from a host so links containing a single flow will not affect the privacy but still has an impact on the communication overhead.

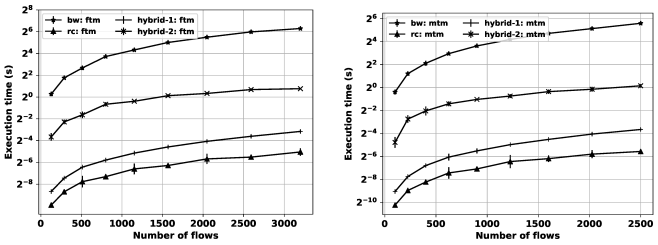
E. Computation overhead

As we can see from Figure 5, the most time-consuming part of our evaluation is to find the minimal non-redundant linear constraints, which is a well-studied problem in optimization theory and network providers can reduce this cost significantly by introducing more efficient algorithms. Even our naive solution can return an equivalent network view within 5 seconds for up to 400 flows in a large network (AS2914), which is often fast enough for many network optimization problems.

Meanwhile, the *equivalent decomposition* (represented as *hybrid-1* in Figure 5) is very efficient, which takes less than 250ms even for the largest test case with 3200 flows and a network with more than 10000 nodes (AS2914). The relaxed redundancy check algorithm takes approximately 1 second, which makes it useful in certain scenarios.

F. Summary

In this section, we evaluate the performance of NOVA thoroughly. We demonstrate that *one-big-switch* can lead to infeasible solutions while NOVA guarantees both feasibility and optimality, which enables consumers to fully utilize the network resources. Privacy is achieved by decomposing the redundant network elements. With strict redundancy check algorithms, we can reduce 60% to nearly 100% of unnecessary information leaks. Depending on the number of flow requests and traffic pattern, NOVA can improve the communication



(a) 2914, *few-to-many*. (b) 2914, *many-to-many*.

Fig. 5: Computation overhead.

time by a factor of 1.25 to 5. The computation overhead is within a reasonable range for typical uses where elephant flows usually takes tens to hundreds of seconds to finish. Thus, NOVA is useful for network providers to achieve collaborative optimization with non-administrative parties to build QoS-aware applications.

VI. Related Work

A. Demands for network views

The demands for being network-aware are quite common. For services built on top of the Internet, user experience depends heavily on the quality of networking service [25]. Previous studies [26] have already shown that obtaining end-to-end metrics can significantly improve the user experience of peer-to-peer services and content delivery networks.

Meanwhile, several studies (*e.g.*, [27]–[29]) have also addressed the need to conduct flow scheduling over the network, suggesting the importance of obtaining the correlations between different data transfers. Such demands are usually associated with traffic with large volumes, such as inter-data center communication, e.g, Google's globally-deployed B4 [15] system and global data intensive science networks [18]. Feeding these applications with more accurate network information allows them to make more intelligent operating decisions.

Another example where being aware of the network performance can be beneficial is fine-grained routing. Latest approaches such as the Software Defined Internet Exchange point (SDX) [30] have enabled Autonomous Systems to set up fine-grained forwarding rules. With the ability to query the expected network performance, an AS would be able to make routing decisions based not only on the cost, but also on the real-time quality of service. Meanwhile, such information can also be provided to QoS-based routing protocols [19], [20].

SOL [31] and CoFlow [27] are SDN-based network optimization frameworks which provide abstractions to simplify the modeling of network optimization problems. However, it would require the optimizer to provide all the information to the network, which jeopardizes the privacy. General collaborative optimization [32]–[34] typically protects the privacy by multiplying a monomial matrix. NOVA enables collaborative optimization by providing the network views to the optimizer, while conducting equivalent transformations to reduce the communication overhead as well as protect the privacy.

B. Providing network view

The most straight-forward way of providing network views is to use its graph representation. Several routing protocols [8]–[11] including OSPF and IS-IS conceptually provide such an abstraction of the network and it is also adopted by the I2RS (Interface to Routing System) IETF Working group [35]. Modern SDN controllers [1]–[4] also provide the global view using the annotated graph model.

The hose model [6] is first introduced for VPN provisioning in 1999. Each endpoint is associated with a *hose* in this model and the details of the actual VPN tunnels are hidden. It is sometimes referred to as the *one-big-switch* in the context of SDN because the network is abstracted as a single logical switch in this model. Because of its simplicity, the hose model is widely used, for example, by many network programming languages [36], [37]. SDX also uses this model to encapsulate the underlying network topology. Data center fabrics are highly customized for scalability [7] and can be modeled as a non-blocking switch where congestion only occurs on access links [14], thus, the *one-big-switch* abstraction is also widely used for data center flow scheduling and tenant resource provisioning [27]–[29].

The pipe model is mostly used by web-based applications or measurement frameworks, which have no control over the network. The pipe model consists of several flows (host pairs) and provides a single pipe for each flow (pair) with the associated metrics. PerfSONAR [38], Meridian[39] and ClosestNode [40] are some concrete examples which provide such end-to-end network views based on measurement, while P4P [26] and the ALTO (Application-Layer Traffic Optimization) protocol [12] are leveraging the network providers' information.

NOVA is similar to ALTO in the sense that in both cases information is provided by the network to non-administrative consumers, which is likely to achieve better accuracy. Meanwhile, we overcome the limitations of ALTO by adopting the equivalence abstraction to provide fine-grained metrics, in particular the *flow correlations*, which makes it possible to suffice the demands from a broader range of applications. This underlying philosophy also distinguishes NOVA from other (especially QoS related) routing protocols and network views based on topological aggregation [10].

VII. Conclusion

In this paper, we systematically study the problem of providing an accurate on-demand network view which is general enough to suffice the requirement of heterogeneous optimization problems. Our abstraction is based on the principle of *equivalence* which guarantees generality, feasibility and optimality. We design the NOVA framework to construct equivalent network views with enhanced privacy preservation and evaluate its performance compared with some well-known network view abstractions. While we have established the theoretical foundations and guidelines of constructing on-demand network optimization view abstractions, essential functionalities such as communication protocols, easy-to-use API design and commercial models are still not fully discovered. We plan

to explore these missing functionalities in the future, along with the implementation and deployment in real networks.

VIII. Acknowledgment

The authors would like to thank Chen Gu, Jingxuan Zhang, Shenshen Chen, Xiao Lin, Haoran Wang and Haizhou Du for their help during the preparation of the paper. We would also like to thank the reviewers for their valuable feedback.

This research is sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

This work is also supported by the *National Science Foundation* (CC-IIE 1440745) and the *National Natural Science Foundation of China* (No. 61472213 and No. 61502267).

References

[1] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," vol. 38, no. 3, pp. 105–110.

[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and others, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, pp. 1–6.

[3] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, pp. 1–6.

[4] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven SDN controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*.

[5] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)."

[6] P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "A flexible model for resource management in virtual private networks," in *Proc. of ACM SIGCOMM*.

[7] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, pp. 51–62.

[8] J. Moy, "OSPF version 2."

[9] D. Oran, "OSI IS-IS intra-domain routing protocol."

[10] T. Vu, A. Baid, H. Nguyen, and D. Raychaudhuri, "EIR: Edge-aware interdomain routing protocol for the future mobile internet."

[11] W. C. Lee, "Topology aggregation for hierarchical routing in ATM networks," vol. 25, no. 2, pp. 82–92.

[12] R. Alimi, Y. Yang, and R. Penno, "Application-layer traffic optimization (ALTO) protocol."

[13] S. Uludag, K.-S. Lui, K. Nahrstedt, and G. Brewster, "Analysis of topology aggregation techniques for QoS routing," vol. 39, no. 3, p. 7.

[14] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. O. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks." in *WIOV*.

[15] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and others, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 3–14.

[16] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, and others, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 1–14.

[17] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *INFOCOM, 2013 Proceedings IEEE*, pp. 854–862.

[18] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science DMZ: A network design pattern for data-intensive science," vol. 22, no. 2, pp. 173–185.

[19] J. L. Sobrinho, "Algebra and algorithms for QoS path computation and hop-by-hop routing in the internet," vol. 10, no. 4, pp. 541–550.

[20] H. Geng, X. Shi, X. Yin, Z. Wang, and H. Zhang, "Algebra and algorithms for efficient and correct multipath QoS routing in link state networks," in *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pp. 261–266.

[21] J. Telgen, "Identifying redundant constraints and implicit equalities in systems of linear constraints," vol. 29, no. 10, pp. 1209–1222.

[22] Y. Bartal, F. Y. Chin, M. Chrobak, S. P. Fung, W. Jawor, R. Lavi, J. Sgall, and T. Tich, "Online competitive algorithms for maximizing weighted throughput of unit jobs," in *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 187–198.

[23] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," vol. 29, no. 9, pp. 1765–1775.

[24] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," vol. 12, no. 1, pp. 2–16.

[25] R. K. Mok, E. W. Chan, and R. K. Chang, "Measuring the quality of experience of HTTP video streaming," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pp. 485–492.

[26] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4p: Provider portal for applications," vol. 38, no. 4, pp. 351–362.

[27] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 31–36.

[28] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, pp. 19–19.

[29] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 435–446.

[30] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined internet exchange," vol. 44, no. 4, pp. 551–562.

[31] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using SOL," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 223–237.

[32] Y. Hong, "Privacy-preserving collaborative optimization."

[33] J. Vaidya, "Privacy-preserving linear programming," in *Proceedings of the 2009 ACM symposium on Applied Computing*, pp. 2002–2007.

[34] J. Li and M. J. Atallah, "Secure and private collaborative linear programming," in *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, pp. 1–8.

[35] J. Medved, N. Bahadur, H. Ananthakrishnan, X. Liu, R. Varga, and A. Clemm, "A data model for network topologies."

[36] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, and others, "Composing software defined networks." in *NSDI*, pp. 1–13.

[37] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: simplifying SDN programming using algorithmic policies," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 87–98.

[38] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, D. M. Swany, S. Trocha, and J. Zurawski, "PerfSONAR: A service oriented architecture for multi-domain network monitoring," in *International Conference on Service-Oriented Computing*, pp. 241–254.

[39] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 85–96.

[40] B. Wong and E. G. Sirer, "ClosestNode.com: an open access, scalable, shared geocast service for distributed systems," vol. 40, no. 1, pp. 62–64.

SFP: Toward a Scalable, Efficient, Stable Protocol for Federation of Software Defined Networks

Franck Le[*] Christopher Leet[‡] Christian Makaya[*] Miguel Rio[†] Xin Wang[+‡] Y. Richard Yang[+‡]
IBM[*] Tongji[+] UCL[†] Yale[‡]

***Abstract*—As more networks deploy software defined networking (SDN) to take advantage of its benefits, including logically centralized, more flexible network-control programming, the inter-connection of such SDN-based networks has become a major remaining challenge. Traditional interconnection approaches such as BGP are designed for less flexible networks and hence can have serious efficiency, scalablity and stability issues such as the compact policy program instantiation (CPPI) issue. In this paper, we define the requirements of the inter-connection of SDN networks and propose a novel protocol, called the SDN Federation Protocol (SFP), to achieve scalable, efficient, and stable interconnection of SDN networks. Instead of being a traditional push protocol such as BGP, SFP adopts a novel pub-sub model to substantially increase flexibility, efficiency and scalability. Going beyond only packet handling, SFP introduces flexible network information spaces, such as the packet space and the flowset space, to achieve more efficient, autonomous federation of network resources spanning across multiple networks.**

Keywords: SDN, interdomain, BGP, SFP

I. Introduction

A major recent development in computer networking is the introduction of software defined networking (SDN), which allows a network to define its behaviors through centralized policies at a conceptually centralized network controller. With the emergence of key components including (1) south-bound datapath models (*e.g.*, Openflow [1], P4 [2]), (2) high-level programming models (*e.g.*, Frenetic [3], Maple [4]), (3) north-bound application interfaces (*e.g.*, IETF supa), and (4) emerging killer SDN applications (*e.g.*, B4 [5]), SDN has made significant progress in realizing the vision that a network can be effectively controlled using a simple, centralized control-plane program with a global view.

As individual networks start to deploy SDN, it is a natural next step to interconnect such SDN networks. Although there are many settings where fully localized, transparent deployment of SDN is enough to reap the benefits of SDN (*e.g.*, edge domains [6]), there are also many settings where the interconnection of individually administrated SDN networks can be beneficial, for example, in expanded reachability settings, where one network needs to reach entities hosted in another network, and in pooled resources settings, where multiple networks benefit from resources made available from each other, as long as the sharing conforms to their local policy programs. In the rest of this paper, when we refer to an SDN network, we mean an individually administrated network. We refer to the interconnection of individually administrated SDN networks as realizing the federation of involved SDN networks or SDN federation for short.

Achieving SDN federation can be challenging, and hence existing work (*e.g.*, SDX [7], SD-WAN) focuses on limited settings. Since some might think that one can still use traditional interdomain protocols to achieve SDN federation, consider the issues when applying BGP, the well-developed, *de facto* interdomain protocol of the Internet, to achieve SDN federation. Unfortunately, applying BGP to SDN federation can have at least two basic mismatches, and both can result in substantial efficiency and scalability issues. First, designed in a traditional networking setting using only destination-IP based routing, BGP uses a single-dimension (*i.e.*, the destination-IP dimension) routing information model. An SDN network, however, can conduct highly flexible, multi-dimension routing, based on a large number of decision dimensions, including not only destination IP, but also source IP, protocol, and port numbers. Naive conversion of multi-dimension SDN decisions to the BGP single-dimension information model can lead to dimension cross-product explosions, resulting in serious scalability issues. Second, designed in a traditional networking setting with limited programmability, BGP is fundamentally a *full instantiation* information-exchange protocol, in that the program decisions at each network need to be fully instantiated as data (*i.e.*, BGP routing information base) and then exchanged among networks. The extremely large decision space of SDN, and in particular, the two-layer SDN decision model, where the routing information base layer is only a cache of the SDN program layer, can make full instantiation unfeasible.

The objective of this paper is to conduct a systematic investigation and design of SDN federation. Instead of adopting an incremental approach of adapting an existing design such as BGP, we conduct a clean design of SDN federation, to address fundamental challenges introduced by SDN, including multi-dimension decision, two-layer SDN decision.

It turns out that the aforementioned challenges can be addressed using simple, extensible techniques, which we introduce as key components of a novel protocol called the SDN federation protocol (SFP). Specifically, to avoid full instantiation, SFP uses a simple, novel pub-sub model to constrain the space where program behaviors are converted into data to be exchanged; to avoid dimension cross-product explosion, SFP introduces novel, information instantiation graph; going beyond only packet handling, SFP introduces flexible network information spaces, such as the flowset space, to achieve more efficient, autonomous federation of network

resources spanning across multiple networks.

The rest of this paper is organized as follows. Section II gives the requirements of SFP. Section III evaluates related work and lists the issues of existing work. Sections IV is the main technical section and gives the SFP design. Section V gives future research directions to conclude the paper.

II. Design Requirements

We list the following requirements for a SDN federation protocol:

- *Efficiency of resource integration across networks*: The protocol should be low overhead, and scalable. The scalability requirement applies to memory, network, and compute resources. For memory resources, for example, under the BGP paradigm, an AS advertises all of its routes to its neighbors, even if a neighbor may have no interest nor traffic to specific destinations. In contrast, a different communication model may have domains subscribe to specific header spaces they are interested in, and only receive information (including updates) associated with those header spaces. For network resources, the protocol should make efficient usage of the bandwidth across domains. Finally, for compute resources, a domain must be able to reply to a neighbor's query within a reasonable time period.

- *Autonomy*: There is a trade-off between autonomy and stability: When every node implements the same algorithm, stability can more easily be guaranteed (*e.g.*, shortest path routing). However, this model is rigid, and does not allow each domain to implement its own policy. Conversely, in a system where each domain can implement its own policy, policies from different domains may conflict and result in instabilities [8]. The protocol should offer a large degree of autonomy to its domains while still guaranteeing the stability of the system.

- *Privacy*: Although a first domain may send multiple queries to a second domain, the first domain should not be able to learn information that the second domain considers sensitive and chooses not to reveal to the first domain.

III. Related Work

With the preceding requirements, we now evaluate related work. Since SDN was initially developed for single administrative domains (*e.g.*, enterprise, datacenter) [6], *i.e.*, intra-domain management and control, only recently are there a number of proposals to extend the benefits of SDN across domains (*e.g.*, [7], [9]–[13]). Many of these proposals adopt an incremental approach and suggest extending, or defining a new protocol similar to, BGP – the current de-facto inter-domain routing protocol – to carry additional information [9], [11], [12]. However, extending BGP to support SDN does not satisfy all of the requirements in Section II. In particular, using BGP presents the following issues:

Cross-product explosion: One of the main benefits of SDN is the support for highly flexible, multi-dimension routing, based on a large number of decision dimensions, including IP addresses, protocol, and port numbers. In contrast, BGP uses only the destination IP as the routing decision. Simply extending the single dimension (*i.e.*, destination IP) of BGP to multiple dimensions can lead to dimension cross-product explosions, and scalability issues. For example, routing for N destinations taking into account up the source IP addresses and destination ports may require up to N x M x P entries, with M representing the number of source IP addresses, and P the number of destination ports.

Full instantiation: BGP is a full instantiation information-exchange protocol. It requires every Autonomous System to compute and advertise the best route for every destination *a priori*. SDN has a much larger decision space, and adopting this model where every network computes and exchanges its decisions for every set of flows *a priori* may be infeasible.

Domain backup: Networks may require reachability despite link failures, router failures, and network partitions, as long as a physical path to the destination exists. However, BGP does not support domain backup and prevents paths having traversed a given Autonomous System from being re-injected into that same Autonomous System. Despite it being a design decision, opertional networks may require domain bcakup [14], [15].

Multi-flows handling: Policies in SDN networks not only control the reachability of flows but also manage resource allocations (*e.g.*, bandwidth) of flows which are not independent from each other. Hence, multi-flow resource querying is more complicated than a single flow due to sharing links, concurrency of flows, and internal sensitive information preservation which is not considered in BGP model.

Although protocols beyond BGP have been proposed, they cannot satisfy the requirements in Section II.

Specifically, a main interdomain protocol for SDN is SDX [7], which enables more expressive policies than conventional hop-by-hop, destination-based forwarding without requiring extensions to BGP. However, SDX relies on a Route Server deployed at Internet Exchange Point (IXP). Autonomous Systems write their policies to the Route Server, and only Autonomous Systems participating at the IXP can therefore benefit from it. In contrast, our requirements include that it is not restricted to Autonomous Systems present at specific IXPs. In addition, our requirements include resource integration to provide resource information (*e.g.*, available bandwidth) across networks. IXP cannot satisfy this requirement.

On the standardization side there has been some limited efforts to standardize interfaces between SDN controllers in different domains. SDNi [9] defines 3 elements: an SDN aggregator to aggregate intra-domain information, a RestAPI

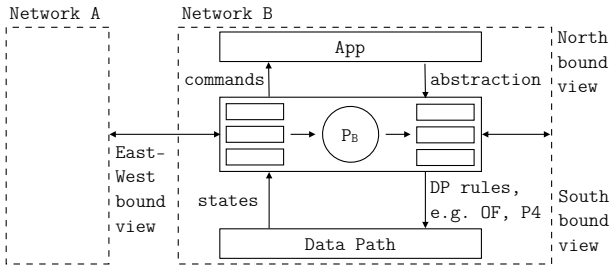


Fig. 1. The unifying model-views framework.

to obtain information from other domains and SDNi wrapper to share information from federated controllers.

Bueno et al. propose the Network Control Layer (NCL) [16], a software framework solution based on SDN, OpenFlow and Network as a Service (NaaS) paradigms. Petropoulos et al. [17] propose a way of coordinating Quality of Service (QoS) across different domains. They do not address all of the requirements in Section II.

IV. SFP Design

We now present the design of SFP. We start with an overview of the basic structure of SFP in Section IV.A. We then present in Section IV.B a key component of SFP, the SFP packet space view.

A. Overview

The model-views context: Considering SDN as a core concept, we design SFP in the context of a unified model-views framework, where the logically centralized control program P_A of network A defines the core model. To be concrete, we show an example P_A:

```
f(Packet pkt):
   if (pkt.tcpSrcPort != 80)
      return drop
   else
      srcClass = map_policy(pkt.Ipv4Src)
      return shortestPath(srcClass, pkt.Ipv4Dst)
```

It is from this model that multiple views are derived, where the views include the north bound (NB), which is the interface of the network to applications using the network; the south bound (SB), which is the realization from the control plane to the data plane; and the east-west bound (EWB), which is the interface between peering networks and hence is the focus of this paper. See Fig. 1 for illustration of this unifying framework. Adopting a unified model-views framework leads to multiple benefits, including automation (as views should be derivable from the model), sharing of common functions when computing multiple views, and consistency (among views, as implied from their consistency to the model).

Each view is different because it has different targets; for example SB is for switches and EWB is for peering networks. Although the focus of this paper is SFP, which provides the EWB view, since a reader may be more familiar with the SB view, we compare the EWB view with the SB view in Table I, to allow the reader better appreciate the SFP design.

Southbound	**East-west Bound**
Switch places physical constraints on SB, as SB must conform to switch computing models such as OpenFlow, and no loop in routing pipeline	No such constraints, but preferable to be similar to existing concepts such as BGP
No privacy concerns	Privacy concerns
Must compile into distributed devices	No such constraint
Controller can process packets (only partial compilation required)	Program must be fully compiled and transmitted

TABLE I
EWB vs SB pipeline comparison.

EWB spaces: In a high level, what a network B provides to another network A is how P_B will handle each packet that A is interested in. In the basic level, P_B processes each packet independently. Hence, the basic information that B needs is how each packet in a *packet space* is handled. As networks provide more services such as QoS and security services, P_B classifies packets into flows. Since the processing of flows can be correlated, in particular, in resource sharing case, we say that A may request information for a set of flows. We refer to this as the *flowset space*, where each point in the space is a set of flows.

In the general case, to better organize information, the EWB view consists of multiple spaces, with two predefined spaces: the packet space and the flowset space.

Basic SFP message flow: With the basic concepts of EWB spaces, we specify the basic SFP message flow, as illustrated in Fig. 2. It is a sub/pub protocol, where a network A sends a sequence of subscription interests to network B. Each subscription is for a subspace of an EWB space that B supports.

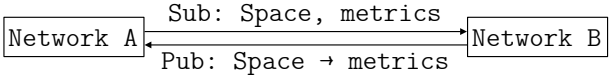


Fig. 2. Basic SFP message flow.

Network A can make multiple subscriptions, for example, with the first for the packet space to obtain reachability for a subset of IP destination addresses and then a set of flows that A plans to schedule. A then can send a sequence of flowset space subscriptions to understand the resource constraints for each flowset.

Example 1: A first sends a packet space subscription to B, and then B replies all packets that it can route:

```
A->B subscription:
  packet space: all packets
  metric: reachability
B-> reply:
  reachable subsets for each individual packet
```

As simple as Example 1 is, it shows that SFP can take BGP as a special case.

Example 2: A may be a organization with a set of flows that it can schedule. Then, we sends a flowset consisting of two flows, each defined by a standard OpenFlow 5-tuple:

```
A->B subscription:
  flowset space: A single flowset point consisting of two flows;
  metric: available bw
B->A reply: feasible region for the flowset
  (e.g., x1 <= 10; x2 <= 5; x1+x2 <= 7)
```

As simple as Example 2 is, it is provides resource constraints that are lacking in previous designs.

B. SFP Packet Space View

The preceding section defines the basic message flow. How the messages are computed and encoded are not specified. Although how messages are computed do not need to be standardized, message encoding need to be standardized for interoperability. In this section, we discuss both message computing and encoding for the packet space, which is the most fundamental space of the EWB.

Problem definition: We define the message computing and encoding problem of the packet space as the following compact policy program instantiation (CPPI) problem: given a program P which takes a point in packet space and returns a routing decision for that point's packets, and a queried subset of packet space, Q, what is the most compact instantiation of P's behavior over Q?

Such an instantiation can be used as a RIB to communicate an network's controller's program to other networks in response to queries. For this instantiation to be practical in production networks, however, it must also meet the following requirements.

- Correctness: P must be represented accurately.
- Updatability: Changes to P must be communicated without complete retransmission.
- Mergability: P's encoding must allow other networks to merge information into its RIB.
- Obfuscation: P's encoding should not reveal unqueried information about P, preserving P's privacy.

Solution architecture: The SFP packet space protocol attempts to solve the compact policy program instantiation problem while ensuring correctness, updatability, mergability and obfuscation by encoding P's behavior over subsets of packet space as pipeline table formatted RIBs. We present the protocol's architecture in Fig. 3.

Consider two networks, controlled by `Controller A` and `Controller B`, running SFP. Upon start up, SFP obtains its controller's routing programs P_A and P_B and compiles each program into a flow table pipeline, which is updated should its program or the data it was compiled with change.

Suppose that `A` sends `B` a packet space query. Upon receipt, `B` passes the query to its SFP app, which selects the subset of the query it is willing to respond to, selects the flow table rules in its pipeline queried by this subset, and formats these rules into a RIB. This RIB is then obfuscated to hide any information that it contains in excess of the query's scope, and passed back to `B`, which sends it out as a reply to `A`.

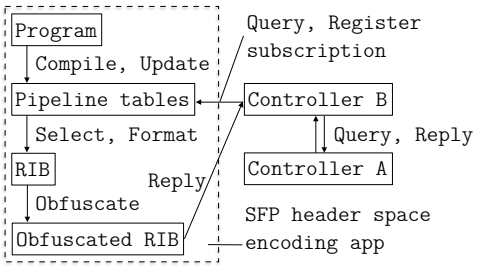


Fig. 3. packet space encoding architecture.

Having given an overview of SFP's packet space encoding architecture, we now examine each of the architecture's three stages in more detail.

Program compilation and update: Program compilation is the transformation of a generic program, expressed in a high level language, into a pipeline of flow tables. Programs are compiled into flow tables because flow tables allow efficient querying and require little additional processing to be transmitted efficiently. Compilation occurs "only-once", offline, avoiding the burden of per request program compilation and removing a potentially lengthy computation from the runtime environment.

We present the compiler's architecture in Fig. 4. First SFP compiles the program into a per instruction table (PIT) pipeline using two synergistic techniques: Symbolic map and Flow Explore. Subsequently, SFP compresses the PIT pipeline into a more compact representation via a third technique, Deep Expansion.

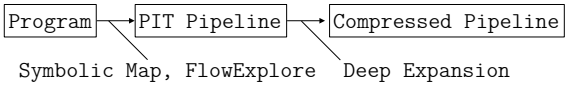


Fig. 4. Program to pipeline compiler architecture.

Per Instruction Table (PIT) pipelines: The key insight underlying the per instruction table (PIT) pipeline is that any program instruction I can be converted into a flow table which matches on I's argument's bindings and whose actions reflect I's execution on a given binding.

For example, consider the instruction `I-example: if (pkt.dstPort < 1024)`. We can transparently replace `I-example` with the table `I-example-PIT`.

```
I-example-PIT:
  pri  dstPort  Action
  ---  -------  ---------------
  1    1        goto if block
  1    ...      ...
  1    2014     goto if block
  1    2015     goto else block
  1    ...      ...

I-symbolic-mapped:
  pri dstPort      Action
  --- -----------  ---------------
  1   10000000000  goto if block
  2   1xxxxxxxxxx  goto else block
  3   xxxxxxxxxxx  goto if block
```

Symbolic map: While we can encode any I as a PIT by enumerating all the values in that I's domain, network programs often reference variables whose domain is too large to practically enumerate - an IPv6 address, for example, can

take 2^{128} distinct values. Fortunately, certain common types of Is have succinct PIT encodings. Such Is, specifically comparisons, conditionals, lookup tables, memory reads and switch function invocations (*e.g.* checksum) are referred to as symbolic mappable, and their encoding as symbolic mapping. `I-example`, for instance, can be symbolically mapped to the table `I-symbolic-mapped`.

The first step to converting generic programs written in high level languages to PIT pipelines is thus passing over the program, identifying symbolic mappable Is and symbolic mapping them.

Flow-Explore: To complete the transformation of a program into a PIT pipeline we must encode the remaining non-symbolic mappable instructions. Such encoding is accomplished by exploring all possible execution paths through the program and adding the results of encoding some I during exploration to that I's flow table, a process referred to as Flow-Explore.

Deep Expansion: While a PIT pipeline is a valid flow table representation of a program, it is often possible to compress it by merging PITs, conserving transmission bandwidth. Our compression transformation Deep Expansion uses heuristics such as shared header fields to identify and combine mergable PITs.

Selecting, formatting, and subscribing queried rules: Upon receiving a query, SFP selects the subset of the query it is willing to respond to Q and then selects the rules in its program's pipeline tables queried by Q. Such selection is accomplished simply by recursively exploring each path through the pipeline, marking each rule for transmission whose match fields correspond either to values within Q or to intermediate values set earlier in the path currently being explored, and terminating the exploration of any path whose rules' match fields correspond to neither value.

After exploration, the rules marked for transmission are used to construct a reduced pipeline which only matches on values within Q. This pipeline constitutes SFP's pipeline formatted RIB, and is conceptually ready for transfer through SFP's controller's EWB interface.

As an example, consider formulating a RIB in response to the query {`dstAddr = (130.91.6.0/30, 18.0.0.0/8), dstPort = x`} from the single table pipeline `example-pipeline`. Rules with their `dstAddr` in the `130.91.6.0/30` block and the default drop rule fall within the queried packet space and are selected for transmission, generating the RIB `example-RIB`.

`example-pipeline:`

dstAddr	dstPort	action
130.91.6.0/32	< 1024	out(1)
130.91.6.1/32	< 1024	out(2)
130.91.6.2/31	< 1024	out(1)
130.91.6.2/31	>= 1024	out(3)
130.91.6.4/30	< 1024	out(4)
130.99.6.8/30	< 1024	out(1)
x.x.x.x	x	drop

`example-RIB:`

dstAddr	dstPort	action
130.91.6.0/32	< 1024	out(1)
130.91.6.1/32	< 1024	out(2)
130.91.6.2/31	< 1024	out(1)
130.91.6.2/31	>= 1024	out(3)
x.x.x.x	x	drop

Rules transmitted to a controller are also marked as subscribed to that controller. If a subscribed rule is affected or eclipsed by an update to the controller's program or to the data that the program depends on, the updated rule or rules are transmitted to the subscribed controller.

Obfuscation: After encoding P_B in an RIB, SFP obfuscates the RIB, limiting the information that the RIB shares and guaranteeing B's privacy. In particular, obfuscation provides two privacy guarantees to controllers:

- Flow privacy: Controllers' network flows are hidden.
- Program privacy: Controllers' programs are hidden.

A naïve RIB containing a subset of a program's pipeline tables may reveal information about the program's structure and the flows it prescribes, necessitating obfuscation.

SFP obfuscates RIBs by transforming them with an obfuscation operator. One potential obfuscation operator, for example, is to first map each terminal action in an RIB's pipeline to a single bit label - `1` if the action transmits packets it applies to and `0` if the action drops them, and then reduce the pipeline by merging rules that map to the same label where possible using TCAM range encoding.

Mapping terminal RIB pipeline actions to generic transmit/drop labels hides the flows prescribed by the pipeline, helping achieve flow privacy. Reducing the pipeline by merging rules that map to the same generic label blurs the pipeline's decisions, helping achieve program privacy. Further investigation into mechanisms to provide different levels of privacy is ongoing.

As an example, consider the obfuscation of `example-RIB`:

`after-map:`

dstAddr	dstPort	label
130.91.6.0/32	< 1024	1
130.91.6.1/32	< 1024	1
130.91.6.2/31	< 1024	1
130.91.6.2/31	>= 1024	1
x.x.x.x	x	0

`after-reduce:`

dstAddr	dstPort	label
130.91.6.0/31	< 1024	1
130.91.6.2/31	x	1
x.x.x.x	x	0

First, `example-RIB` is converted to `after-map` by mapping each of its terminal actions to the labels `0` and `1`. Next, the `after-map` table is reduced to the `after-reduce` table by merging `after-map`'s first two pairs of rules, exploiting their shared label `1` and mergable match fields.

V. Future Research Directions

Although SFP allows more flexible routing than BGP, and strives to address several of its issues, a number of challenges still need to be addressed. Below we list key research directions to fully realize the benefits of SFP.

SFP in multi-hop settings: This paper presented the communication paradigm between two adjacent domains. Extending the exchange and negotiation of network flows across multiple domains introduces new challenges. For example, how can a controller merge its local information with the information received from its peers? What operations can be executed? What output and format can be sent by the node to its neighbors?

SFP for multiflow queries: We illustrated how a domain can subscribe a flow to its neighbor. Ultimately, a domain

should be able to subscribe to multiple concurrent flows. This capability would allow the domain to learn of the impact when sending multiple flows simultaneously. For example, the flows may share a physical link. As a result, the flows would not get the total of the advertised bandwidths, but a lower throughput when sent concurrently. Querying for multiple flows may also allow a domain to infer the risk of flows experiencing disruption in the event of a shared node or link failure. However, can a domain reply to a multiflow query and still preserve its internal sensitive information?

SFP correctness and stability analysis: We will identify desired properties of correctness, and ensure that the protocol satisfies them. For example, in the absence of changes to the network topology, it is commonly desirable for a routing protocol to quickly converge to a stable state devoid of routing anomalies (*e.g.*, black holes, forwarding loops) [18]–[21]. Routing protocols should not be vulnerable to permanent route oscillations [8], [22], [23].

SFP for fully integrated interconnection: As a result of failures, a domain may be partitioned into multiple components which can no longer directly communicate with each other. There may exist physical paths between the components through external domains. However, BGP does not support domain backup and prevents those physical paths from being visible to the components. As a result, the components may become disconnected. To support domain backup, ISPs often have to resort to ad-hoc solutions [14], [15]. We will investigate and compare means to support domain backup as part of the proposed protocol.

ACKNOWLEDGEMENT

This research was supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions; NSFC #61672385, FAST Magellan. This research was also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291. [Online]. Available: http://doi.acm.org/10.1145/2034773.2034812

[4] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. ACM, 2013, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486030

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, 2007.

[7] A. Gupta, L. Vanbever, M. Shahbaz, S. P. D. B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," in *Proceedings og SIGCOMM 2014*. IEEE, August 2014, pp. 233–239.

[8] R. Mahajan, D. Wetherall, and T. Anderson, "Towards coordinated interdomain traffic engineering," in *Proceedings of Third Workshop on Hot Topics in Networks (HotNets-III)*, 2004.

[9] "Inter SDN Controller Communication (SDNi)," http://events.linuxfoundation.org/sites/events/files/slides/ODL-SDNi_0.pdf.

[10] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multidomain SDN controllers," in *Proceedings of IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[11] P. Lin, J. Bi, and Y. Wang, "East-West Bridge for SDN Network Peering," in *Proceedings of ICoC 2013*, 2013.

[12] "BGP based SDN http://network-insight.net/2015/09/bgp-based-sdn/," 2015.

[13] F. Benamrane, M. B. mamoun, and R. Benaini, "An East-West interface for distributed SDN control plane: Implementation and evaluation," *Computers & Electrical Engineering*, vol. 57, p. 162175, January 2017.

[14] F. Le, G. G. Xie, D. Pei, J. Wang, and H. Zhang, "Shedding light on the glue logic of the Internet routing architecture," in *ACM SIGCOMM Computer Communication Review*, 2008.

[15] Cisco, "OSPF Redistribution Among Different OSPF Processes," http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/4170-ospfprocesses.html, 2016.

[16] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin, "An OpenNaaS Based SDN Framework for Dynamic QoS Control," in *Proceedings of IEEE Conference on SDN for Future Networks and Services (SDN4FNS)*, 2013.

[17] G. Petropoulos, F. Sardis, S. Spirou, and T. Mahmoodi, "Software-defined inter-networking: Enabling coordinated QoS control across the Internet," in *Proceedings of ICT 2016*, 2016.

[18] T. Griffin and G. Wilfong, "An analysis of BGP convergence properties," *ACM SIGCOMM Computer Communication Review*, 1999.

[19] J. L. Sobrinho, "An algebraic theory of dynamic network routing," *IEEE/ACM Transactions on Networking (TON)*, 2005.

[20] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks," *SIGCOMM Comput. Commun. Rev.*, 2005.

[21] F. Le, G. Xie, and H. Zhang, "Theory and new primitives for safely connecting routing protocol instances," *ACM SIGCOMM Computer Communication Review*, 2010.

[22] T. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Transactions on Networking (ToN)*, 2002.

[23] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Transactions on Networking (TON)*, 2001.

Embracing Big Data with Compressive Sensing: A Green Approach in Industrial Wireless Networks

Linghe Kong, Daqiang Zhang, Zongjian He, Qiao Xiang, Jiafu Wan, and Meixia Tao

ABSTRACT

New-generation industries heavily rely on big data to improve their efficiency. Such big data are commonly collected by smart nodes and transmitted to the cloud via wireless. Due to the limited size of smart node, the shortage of energy is always a critical issue, and the wireless data transmission is extremely a big power consumer. Aiming to reduce the energy consumption in wireless, this article introduces a potential breach from data redundancy. If redundant data are no longer collected, a large amount of wireless transmissions can be cancelled and their energy saved. Motivated by this breach, this article proposes a compressive-sensing-based collection framework to minimize the amount of collection while guaranteeing data quality. This framework is verified by experiments and extensive real-trace-driven simulations.

INTRODUCTION

The Industry 4.0 revolution is taking place in this big data era. Benefiting from the analysis of big data, customized services can be provided, production efficiencies are optimized, and emerging industries are gradually growing. In quite a few modern industries, big data are collected by smart nodes and transmitted via wireless. For example, in a manufacturing plant, ubiquitous sensors gather environmental data to support the fine-grained adaptation of cooling systems; smart urban crowdsensing applications [1] acquire real-time data from thousands of mobile phones to make local communities and cities more sustainable.

For the smart node, whether sensor or phone, wireless transmission is one of the biggest electricity burners. A field test [2] shows that the power consumption of WiFi in a popular smartphone is about 500 mW, while its battery is only 1200 mAh and 3.7V Li-Ion. In other words, this smartphone could support at most $1200 \times 3.7/500 = 8.88$-hour WiFi transmission even in the ideal case.

To address the dilemma between the demand of big data collection and the limited energy in smart nodes as shown in Fig. 1, it is urgent to design a novel green collection solution. Such a solution is promising to facilitate big-data-based modern industry.

Plenty of techniques have been developed for energy-efficient wireless networks, such as antenna gain [3] and placement strategy [4]. Considering the feature of big data, this article introduces a new direction: data redundancy.

Redundant data widely exist in big data, and they usually contribute little to the efficiency improvement of next-generation industries. If we do not collect these redundant data and only collect the principal data, huge amounts of power consumption will be saved. Promising as it seems, however, one challenging problem arises: How can we distinguish whether a certain datum is principal or redundant before collecting all data? No wireless transmission can be saved in traditional methods because they have to collect all data and then analyze the redundancy.

To tackle the challenge, this article proposes a novel compressive-sensing-based collection framework. Compressive sensing [5] is an advanced mathematic theory for data completion using very few sampled data. The proposed green collection framework consists of two core components. First, to reduce the number of transmissions, an online learning component predicts the minimal amount of data that needs to be collected. These data are considered as the principal data, and their amount is constrained by compressive sensing. Second, a local control component running at every node further tunes the collection strategy according to the dynamics and unexpected situations. Combining these two components, this framework reduces power consumption and guarantees data quality simultaneously. Extensive real-trace-driven simulations are conducted to demonstrate the efficacy and efficiency of the proposed framework. Open issues and future research directions on this green collection framework are also discussed.

The proposed solution is a general framework. It is easy to add customized components into this framework according to the demands of industrial applications. We believe the green collection framework has wider implications and prospects for big-data-based industry than those explored in this article.

Aiming to reduce the energy consumption in wireless, the authors introduce a potential breach from data redundancy. If redundant data are no longer collected, a large amount of wireless transmissions can be cancelled and their energy saved. Motivated by this breach, the authors propose a compressive-sensing-based collection framework to minimize the amount of collection while guaranteeing data quality.

Linghe Kong and Meixia Tao are with Shanghai Jiao Tong University; Daqiang Zhang, Zongjian He, and Qiao Xiang are with Tongji University; Jiafu Wan is with South China University of Technology.

0163-6804/16/$25.00 © 2016 IEEE

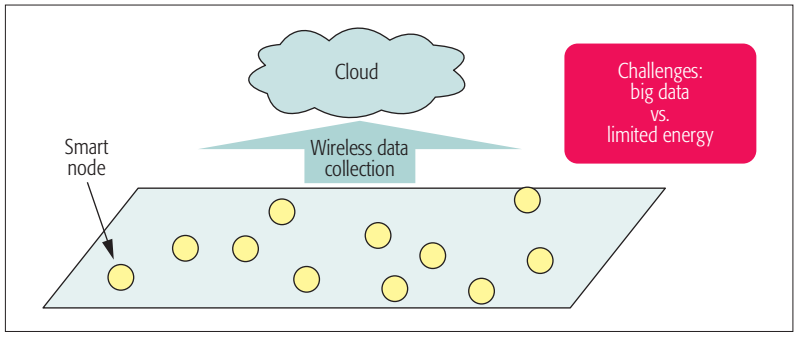


Figure 1. In wireless data collection applications, the distributed smart nodes sense and transmit data to the cloud. The dilemma is between the big data to be transmitted and the limited energy in smart nodes.

Background

The green collection framework typically works at the intersection of three research areas: big data, energy-efficient data collection, and compressive sensing.

Big Data

Both academia and industry are paying a great deal of attention to the explosive growth of data. Lynch [6] posed the open question in *Nature*: How do your data grow? This article attracted many scientists to start working in the area of big data. Baraniuk [7] reported in *Science* that the bottleneck of signal processing now is data deluge: the amount of data generated worldwide (1250 billion GB in 2010), which is dominated by sensory data, is growing by 58 percent per year. The revolution of big-data-driven industry has spread all over the world.

Energy-Efficient Data Collection

The energy constraint is a critical problem in big data collection. According to [8], battery capacity has only doubled in the past 35 years. Moreover, the hazardous sensing environment precludes manual battery replacement. The energy constraint is unlikely to be solved in the near future due to the size limitation of smart nodes.

However, collecting and transmitting big data consume a lot of power in smart nodes. For example, the transmission power of WiFi is up to 500 mW, LTE is up to 200 mW, Bluetooth is up to 100 mW, and ZigBee is up to 5 mW.

Hence, green methods are investigated from the physical layer to the application layer in wireless networks [9–12]. Although existing solutions are highly diverse, none of them take data redundancy into consideration.

Compressive Sensing

Compressive sensing [5] is a generic method to recover the whole condition with only a few sampled data. Several effective compressive sensing applications have been developed in the data completion field [13] (e.g., traffic estimation and video streaming). It has been proven that the whole environment can be near-optimally recovered even if there are more than 70 percent sensory data are missing [14], which motivates us to exploit compressive sensing to reduce the amount of data collection.

Problem Statement

In a big data collection system, smart nodes are usually distributed in the given area to sense data and transmit these data to the cloud via wireless communications. The cloud analyzes the collected data and provides customized service or production. Suppose there are a total of n nodes, and the period of monitoring time is evenly divided into t time slots. Every node collects data once per time slot at most. With the growth of the scale in industrial applications, the total collected data are very big.

The big data can be represented as a large matrix X, where every element is the data collected by one node at one time slot. A matrix with no empty elements means that all data are collected, which indicates 100 percent data quality but costs nt wireless transmissions.

On one hand, to reduce the number of transmissions, it is desired that only principal data are collected. Assume that the amount of principal data is ρ and $\rho << nt$. On the other hand, to guarantee the data quality, it is desired that the principal data are adequate to represent the whole big data, that is, the recovered matrix $\hat{X}$ is close to the complete X, where $\hat{X}$ is the matrix computed by compressive sensing using only principal data.

From the above, we state the problem as follows: The green collection problem aims to minimize the amount of principal data ρ for energy saving and is constrained by $\hat{X} \approx X$ for quality assurance.

Two main metrics are defined to measure the performance of green collection solutions:

- Energy consumption ratio α: This ratio can be approximated as ρ/nt, that is, transmitting the amount of principal data over transmitting the total big data, in which we consider the consumption is equal for every transmission.
- Data error ratio ε: The average error between recovered matrix and complete matrix, that is, $\varepsilon = ||\hat{X} - X||/||X||$.

Real Data Analysis

Before describing the design of a green collection framework, we analyze the redundancy feature in big data. We observe that most big datasets have obvious redundancy. The possible reasons include redundant smart nodes deployed for data collection, nodes in close or the same locations sensing similar data, and sensory data usually having strong correlation with time variance.

Then we introduce three real traces and validate their low-rank properties, which implies the data redundancy in common sensory data. The three datasets are gathered by real projects.

The Intel Indoor experiment was gathered by the Intel Berkeley Research lab. There are totally 54 nodes placed in a 40 m × 30 m room. Every node reports data every 30 s. From all the gathered data, we select 50 nodes' × 4000 slots' data to form a complete dataset.

The GreenOrbs project is a real-world sensor network for forest surveillance. More than 500 nodes are scattered on Tianmu Mountain, China, and gather temperature, light, and humidity data once every 5 min. We select 249 × 500 data from GreenOrbs.

The OceanSense project contributes our third

dataset. This dataset contains 20 nodes deployed in the sea of Taipingjiao, China. Each node reports temperature and light every 2 min. We select 15 × 1000 data from OceanSense.

From the three selected datasets, we generate six complete matrices: Indoor-Temp, Indoor-Light, Forest-Temp, Forest-Light, Ocean-Temp, and Ocean-Light for analysis.

Redundancy Discovery

Data at different locations over different times are usually not independent, resulting in a low-rank structure (i.e., some data are redundant). In order to mine the redundancy, we analyze the selected matrices using singular value decomposition (SVD) [14], which is an effective non-parametric technique for revealing a low-rank structure. In Fig. 2, we illustrate the cumulative distribution function (CDF) of top- percent singular values in the selected matrices. The X-axis presents the normalized number of singular values. The Y-axis presents the ratio of the cumulative values of top-percent singular values. This figure implies that the sum of all singular values is always contributed by only a few top singular values in real data. For example, the top 5 percent singular values contribute 92 percent of sum singular values in Indoor-Temp. The universal existence of such trends reveals the low-rank structures in these traces. These redundancy features indicate that big data can be near-optimally recovered by compressive sensing even if only a few data are collected.

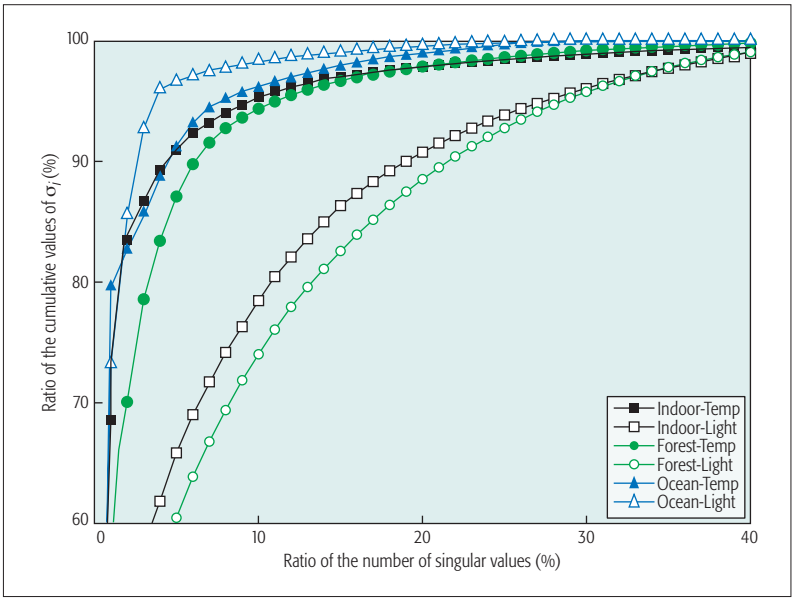


Figure 2. CDF of singular values to explore the redundancy feature in real datasets.

Green Collection Framework

Inspired by the observed feature, a novel green collection framework is designed in this section.

Design Overview

The architecture of our green collection framework is illustrated in Fig. 3, which consists of two core components.

First, the **online learning** component runs at the cloud side. Leveraging the historical data and compressive sensing, this component predicts the minimal amount of data that needs to be collected in the near future. Then this component transforms the data amount to be the collecting probability and reports the probability to every node.

Second, the **local control** component runs at every node. Since the collecting probability provided by the online learning component is an average value from the global view, it may not be suitable for an individual node. Resorting to the adaptive control theory, this component adaptively tunes one node's collecting probability according to the dynamics and unexpected situation.

The advantages of this framework include:

- This framework is easy to implement in practice.
- The high-complexity compressive sensing and prediction are computed at the centralized cloud side. The distributed computing at the node side is low-complexity local control.
- This framework tactfully takes advantage of compressive sensing to reduce the power consumption while guaranteeing the data quality by both global prediction and local adjustment.

Detailed designs of two components are introduced in the following two subsections. Moreover, we pose open problems and discuss some future research directions about this framework.

Online Learning Component

As per the analysis in the previous section, data redundancy universally exists in big data. Existing mathematical tools can analyze the redundant data at the cloud side after collecting all data. However, because a certain node has no global view, when it senses a datum, it is not easy to distinguish whether this datum is principal or redundant. Thus, it cannot locally decide whether to transmit this datum to the cloud or not.

Leveraging the advantage of compressive sensing, this problem can be simplified. Note that compressive sensing can achieve near-optimal matrix completion if a minimal amount of data (related to the rank) are collected and these data are randomly distributed in the matrix. Thus, instead of distinguishing an individual datum, our design aims to acquire the minimal amount of data and near-optimally recover all of the data by compressive sensing.

The online learning component operates as follows.

- Predicting the minimal amount of data collection for compressive sensing. First, the change of rank in historical data can be analyzed at the cloud side using SVD. Second, applying the prediction methods [15] on the historical ranks, we can estimate the rank in the next time slot. To achieve an accurate prediction, the classic ARIMA model is adopted for rank estimation, which considers both trend and periodicity. Third, the minimal amount of data K can be derived by compressive sensing theory [13].
- Adding the margin in the predicted amount. Since the environment is dynamic, a predicted K may not be adequate for near-optimal recovery. Hence, we introduce some margin into the pre-

dicted amount by $\rho = \beta K$, where β is the margin coefficient, and we define ρ as the amount of principal data in this article.

• Computing the collecting probability. The collecting probability P can be computed by $P = \rho/nt$, which indicates that the collecting probability of every node at every time slot is P. The cloud will broadcast this probability to all nodes once it has P.

Local Control Component

After receiving P from the cloud, a smart node could collect data at every time slot with probability P. However, this probability is an average result from the global view without considering the individual difference of every node. For example, in a noise detection application, indoor noise changes less frequently than outdoor noise does. Obviously, the collected data from an outdoor node are more important than those from an indoor node.

To achieve a better data quality, the local control component is designed. This component runs at every node and self-adapts the value of P_i according to P, dynamics, unexpected issues, neighbor status, residual energy, and link quality.

Dynamics: The recent sensed datum is compared to the previous sensed data. If the change of data is stable or periodic, P_i can be decreased gradually. If an aperiodic and frequent change of data happens, P_i is gradually increased.

Unexpected Issue: If any unexpected issue is detected, P_i could be increased sharply. For example, a smartphone detects a traffic crash; if there is no other smartphone nearby, this smartphone enlarges P_i immediately.

Neighbor Status: Using the same example of the traffic crash, if there are many smartphones nearby, each one could keep its P_i for data collection.

Residual Energy: When the residual energy in a node is not enough, it reduces its collecting rate P_i for energy saving and reports this condition to the cloud.

Link Quality: The transmission power depends on the link quality of the wireless channel. Generally, a poor channel caused by interference or mobility results in large transmission power and multiple retransmissions. Hence, P_i is reduced when the link quality becomes poor.

The local control component is not limited to the above aspects. More aspects can be appended to this control component as input.

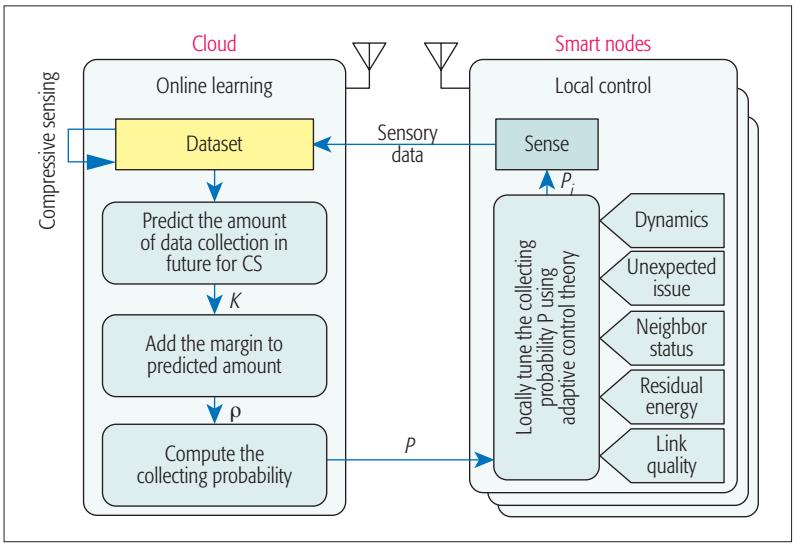


Figure 3. Architecture of the green collection framework.

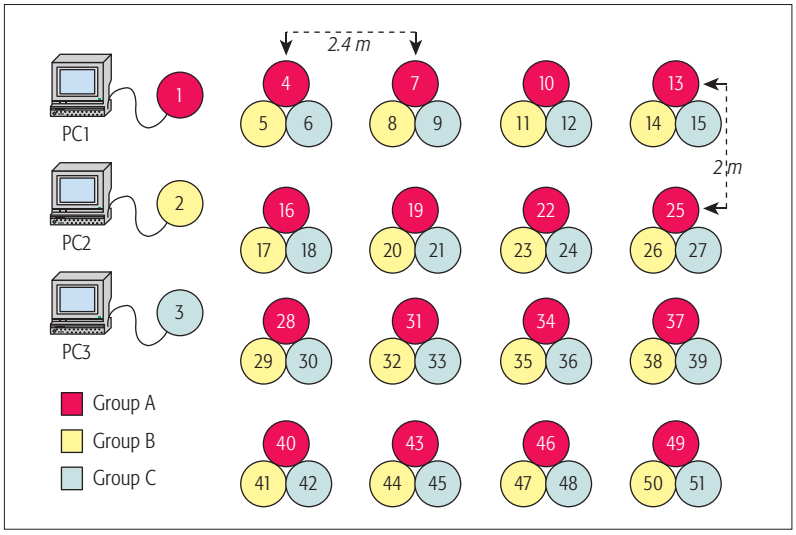


Figure 4. Experiment settings.

Performance Evaluation

We implement a real testbed and conduct trace-driven simulations to evaluate the performance of the proposed green collection framework (GCF).

Experimental Implementation

Experimental Testbed: Our testbed includes a total of 51 TelosB sensor nodes. They are divided into three groups, A, B, and C, carrying out different data collection methods for comparison. Each group has 16 nodes to sense environmental data and 1 sink node to gather these data. In a 7.2 m × 6 m open-air area, 4 × 4 = 16 positions are selected to deploy nodes as a grid. As shown in Fig. 4, at each position, there are three sensor nodes, which belong to three respective groups. A total of 48 nodes are deployed in the area. The other three sink nodes are connected to three laptops.

Each group with 17 sensor nodes organizes its own network. These nodes transmit data using ZigBee. There are 16 ZigBee channels in the 2.4 GHz industrial, scientific, and medical (ISM) band. The three groups of WSNs work during the same period with three non-overlapping channels, so there is no interference among them.

Implementation Setting: There are some common configurations for the three groups. The duration of every time slot is set as 1 min. The collected data are stored in a database in the laptop according to our customized format including timestamp, node ID, temperature, humidity, light, voltage, and received signal strength indicator (RSSI).

The individual configuration for each group is as follows. Group A: Collection Tree Protocol (CTP). The TinyOS library provides the code of CTP. The radio is always on for data transmission. Group B: Fixed low-duty-cycle (FLDC12.5), one cycle is set to 4 h with 0.5-h active state and 3.5-hour sleep state. Thus, the duty-cycle is 12.5

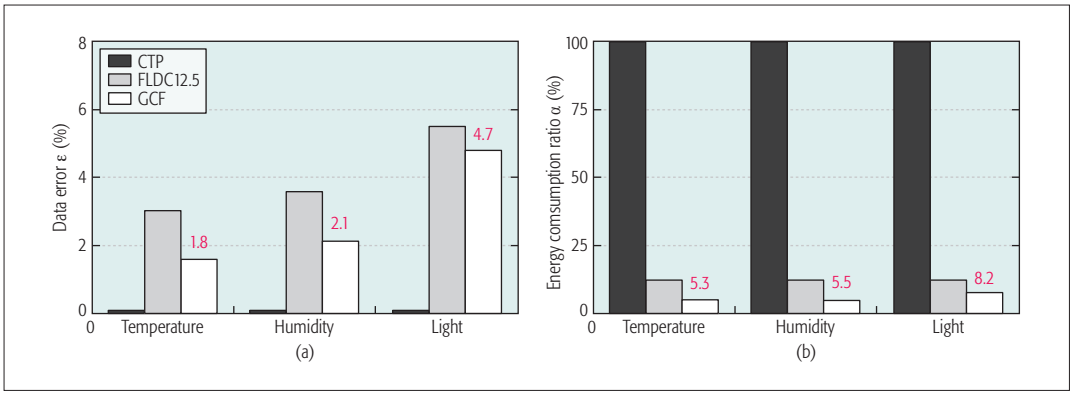


Figure 5. Experiment results: a) experimental performance: data factor; b) experimental performance: energy consumption.

GCF outperforms classic data collection methods in this experiment. Compared with CTP, GCF is much better on energy efficiency within the requirement of data quality. Compared with FLDC12.5, both methods can achieve the data quality, but GCF performs much better on power consumption.

percent. (Different duty cycles are also tested at 50, 25, and 6.25 percent. We only show FLDC12.5 here because it has the lowest power consumption subject to the error ratio ≤ 5 percent.) Group C: GCF with the requirement of error ratio ≤ 5 percent.

Experiment Results: The data quality and power consumption are compared among three groups. All three groups run 10 days for temperature, 10 days for humidity, and 10 days for light collection and recovery.

The metric of data quality is the data error ratio ε. Figure 5a plots the histogram of ε in different groups in different environments. Group A: Since CTP gathers all data, it has no error, ε = 0. Group B: FLDC12.5 loses 87.5 percent environmental data. Although the missing data are estimated by compressive sensing, error ratios are 3.1 percent in temperature, 3.6 percent in humidity, and 5.4 percent in light. Group C: GCF offers satisfactory results on data quality. Due to the accurate prediction and local feedback control, GCF displays ε = 1.8 percent in temperature, 2.1 percent in humidity, and 4.7 percent in light after compressive sensing. In summary, the comparison result indicates that the GCF can ensure the accuracy requirement.

The power consumption is measured by the energy consumption ratio α. The results of energy consumption ratio are displayed in Fig. 5b. Group A: The radio keeps turning on in CTP, so α = 100 percent. Group B: Since the duty cycle is fixed in FLDC, α =12.5 percent. Group C: The number of transmissions in GCF changes according to the dynamic environment. From Fig. 5b, we observe that α = 5.3 percent in temperature, 5.5 percent in humidity, and 8.2 percent in light. The results imply that GCF is better than FLDC12.5 and much better than CTP in energy saving in our experiment.

GCF outperforms classic data collection methods in this experiment. Compared to CTP, GCF is much better on energy efficiency within the requirement of data quality. Compared to FLDC12.5, both methods can achieve the data quality, but GCF performs much better in power consumption.

Trace-Driven Simulation

Simulation Setting: Although the experiment verifies the efficacy and efficiency of GCF, it only carries out in an experimental scenario with some limitations such as small-scale sensor networks and small area. In order to test the extensive applicability of GCF, we conduct the simulations based on the three real datasets introduced earlier. These three datasets are on diverse scales (50, 249, 15 nodes), diverse areas (40 m × 30 m, 200 m × 100 m, and 300 m × 100 m), and diverse scenarios (indoor, forest, and ocean). Every dataset is simulated by CTP, FLDC50, FLDC25, FLDC12.5, FLDC6.25, and GCF, respectively. The requirement of data error ratio is still set ≤ 5 percent.

Simulation Results: Figure 6 shows the data error and energy consumption ratios of different data collection methods among indoor, forest, and ocean datasets in our simulations.

We can find in Figs. 6a, 6b, and 6c that every ε is 0 for in CTP; for FLDC, the smaller duty cycles result in larger error ratios; and the error ratios of GCF are less than 5 percent in all three scenarios.

The energy consumption ratios of CTP, FLDC50, FLDC25, FLDC12.5, and FLDC6.25 are 100, 50, 25, 12.5, and 6.25 in Figs. 6b, 6d, and 6f. These values are fixed, and are independent of scenarios or environments. Nevertheless, such ratios of GCF are dynamic corresponding to the diverse scenarios or environments. Most energy consumption ratios α of GCF are smaller than 10 percent. For example, α in Ocean-Temp and Ocean-Light are only 6.0 percent, and in Indoor-Temp 7.5 percent.

In summary, the results in the simulation are similar to the performance in the experiment. The proposed GCF guarantees the data quality with low energy consumption. The most important property of GCF is its self-adaptation to the dynamics, making it outperform existing methods in nearly all scenarios.

Discussion

Using data redundancy and compressive sensing to reduce power consumption is a new concept in wireless big data collection. Open issues and research directions are worth investigation in the future.

There are still two open issues in the proposed framework. First, the current GCF cannot achieve a minimal amount of data collection because it adopts random collection from a global view but does not optimize the collection amount on every individual node. A more accurate collection method is desired to save more

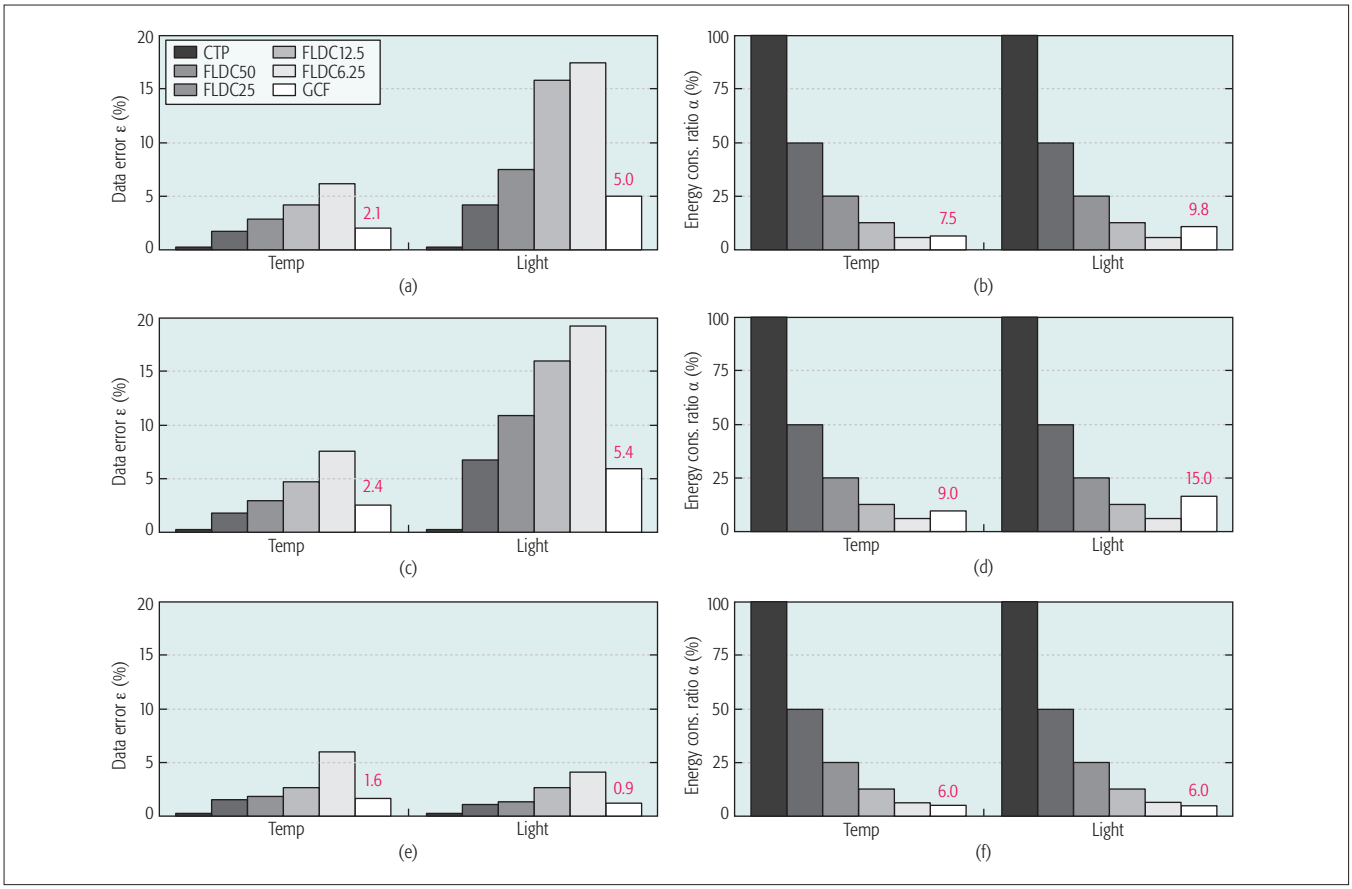


Figure 6. Simulation results: a) indoor dataset simulation: data error; b) indoor dataset simulation: energy consumption; c) forest dataset simulation: data error; d) forest dataset simulaiton: energy consumption; e) ocean dataset simulation: data error; f) ocean dataset simulation: energy consumption.

power consumption. Second, there is a trade-off between the recovery accuracy and the total collection amount. Deriving the theoretical curve to present the trade-off relationship is still an open issue.

The GCF also produces several promising research directions. One valuable direction is to study the correlation between multi-source data to further reduce the amount of data collection. For example, we can collect some light data to estimate not only the light but also the temperature distribution due to their high correlation. The second significant direction is false data detection. To maintain the data quality, false data should be detected and removed from the collected principal data. In addition, this work only considers the correlation in the time domain. If the positions or trajectories are known, space correlation could further optimize the amount of mobile data collection. Last but not least. in a multihop network, network coding and other data aggregation techniques can be taken into account to further reduce the total amount of data collection.

Conclusion

A green collection framework is proposed in this article to save energy in big-data-based smart industries. The core contribution of this framework is to reduce the number of transmissions by leveraging the compressive sensing theory. The evaluation results demonstrate that the proposed framework dramatically decreases the power consumption compared to existing approaches while the data quality is guaranteed.

Acknowledgment

This research was supported in part by National Science Foundation of China grant 61303202, 61472283, 61103185, China Postdoctoral Science Foundation grant 2014M560334 and 2015T80433, the Fundamental Research Funds for the Central Universities No. 2015ZZ079, No. 2013KJ034, No. 2100219043, the Fok Ying-Tong Education Foundation, China grant No. 142006, and the work by Meixia Tao is supported by the NSF of China under grant 61322102.

References

[1] G. Cardone *et al.*, "Fostering Participaction in Smart Cities: A Geo-Social Crowdsensing Platform," *IEEE Commun. Mag.*, vol. 51, no. 6, 2013, pp.112–19.
[2] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," *Proc. USENIX Annual Technical Conf.*, vol. 14, 2010.
[3] D. Feng *et al.*, "A Survey of Energy-Efficient Wireless Communications," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 1, 2013, pp. 167–78.
[4] J. Lloret *et al.*, "A Wireless Sensor Network Deployment for Rural and Forest Fire Detection and Verification," *Sensors*, vol. 9, no. 11, 2009, pp. 8722–47.
[5] D. L. Donoho, "Compressed Sensing," *IEEE Trans. Info. Theory*, vol. 52, no. 4, 2006, pp. 1289–1306.
[6] C. Lynch, "Big Data: How Do Your Data Grow?" *Nature*, vol. 455, no. 7209, 2008, pp. 28–29.
[7] R. G. Baraniuk, "More Is Less: Signal Processing and the Data Deluge," *Science*, vol. 331, no. 6018, 2011, pp. 717–19.
[8] T. He *et al.*, "Energy-Efficient Surveillance System Using Wireless Sensor Networks, *Proc. ACM MobiSys*, 2004, pp. 270–83.

[9] M. Dong *et al*., "Joint Optimization of Lifetime and Transport Delay Under Reliability Constraint Wireless Sensor Networks," *IEEE Trans. Parallel Distrib. Sys*., vol. 27, no. 1, 2016, pp. 225–36.
[10] G. Han *et al*., "Green Routing Algorithms for Wireless Multimedia Sensor Networks," *IEEE Wireless Commun*., 2016.
[11] S. He *et al*., "Emd: Energy-Efficient Delay-Tolerant P2P Message Dissemination in Wireless Sensor and Actor Networks," *IEEE JSAC*, vol. 31, 2013, pp. 75–84.
[12] X. Zhang *et al*., "Cross-Layer-Based Modeling for Quality of Service Guarantees in Mobile Wireless Networks," *IEEE Commun. Mag*., vol. 44, no. 1, 2006, pp. 100–06.
[13] E. J. Candes and Y. Plan, "Matrix Completion with Noise," *Proc. IEEE*, vol. 98, no. 6, 2010, pp. 925–36.
[14] L. Kong *et al*., "Data Loss and Reconstruction in Wireless Sensor Networks," *IEEE Trans. Parallel Distrib. Sys*., vol. 25, no. 11, 2014, pp. 2818–28.
[15] W. Shen *et al.*, "A New Energy Prediction Approach for Intrusion Detection in Cluster-Based Wireless Sensor Networks," *Springer Green Communications and Networking*, 2012, pp. 1–12.

Biogrpahies

Linghe Kong is currently an associate professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. Before that, he was a postdoctoral researcher at McGill University, Canada. He received his Ph.D. degree from Shanghai Jiao Tong University in 2012, his Master's degree from TELECOM SudParis in 2007, and his B.E. degree from Xidian University in 2005. His research interests include wireless communication, sensor networks, mobile computing, the Internet of Things, and smart energy systems.

Daqiang Zhang received his B.Sc. degree in management science and M.Sc. degree in computer science from Anhui University, Heifei, China, in 2003 and 2006, and his Ph.D. degree in computer science from Shanghai Jiao Tong University, China, in 2010. His research includes mobile computing, distributed computing, and wireless sensor networks. Currently, he is an associate professor with the School of Software Engineering, Tongji University, Shanghai.

Zongjian He received his Ph.D. degree from the Department of Computing, Hong Kong Polytechnic University in 2015, and his M.Sc. and B.Eng. degree from Tongji University, Shanghai, China, in 2007 and 2004, respectively. He is currently an assistant professor in the School of Software Engineering, Tongji University. His research interests include wireless sensor networks, vehicular networks, participatory sensing applications, and mobile computing.

Qiao Xiang is currently a postdoctoral fellow at Tongji University and Yale University. From 2014 to 2015, he was a postdoctoral fellow at McGill University. He received his Master's and Ph.D. degrees from Wayne State University in 2012 and 2014, respectively. He received his Bachelor's degree from Nankai University in 2007.

Jiafu Wan has been a professor in the School of Mechanical and Automotive Engineering at South China University of Technology since September 2015. Thus far, he has authored/co-authored more than 60 journal papers and 30 international conference papers. His research interests include Industry 4.0, cyber-physical systems, the Internet of Things, industrial wireless networks, cloud computing, embedded systems, and industrial robotics.

Meixia Tao received her B.S. degree in electronic engineering from Fudan University, Shanghai, China, in 1999, and her Ph.D. degree in electrical and electronic engineering from Hong Kong University of Science and Technology in 2003. She is currently a professor with the Department of Electronic Engineering, Shanghai Jiao Tong University. Prior to that, she was an assistant professor at the Department of Electrical and Computer Engineering, National University of Singapore from 2004 to 2007.

Magellan: Generating Multi-Table Datapath from Datapath Oblivious Algorithmic SDN Policies

Andreas Voellmy[+] Shenshen Chen[*] Yale University[+] Xing Wang[*] Tongji University[*] Y. Richard Yang[*+]

ABSTRACT

Despite the emergence of multi-table pipelining as a key feature of next-generation SDN data-path models, there is no existing work that addresses the substantial programming challenge of utilizing multi-tables automatically. In this paper, we present Magellan, the first system that addresses the aforementioned challenge. Introducing two novel, substantial algorithms, map-explore and table-design, Magellan achieves automatic derivation and population of multi-table pipelines from a datapath-oblivious, high-level SDN program written in a general-purpose language. Comparing the flow tables generated by Magellan with those produced from standard SDN controllers including OpenDaylight and Floodlight, we show that Magellan uses between 46-68x fewer rules.

CCS Concepts

•**Networks** → **Programming interfaces;**

Keywords

SDN, Programming model, Multi-table pipeline

1. INTRODUCTION

Multi-table pipelining has emerged as the foundation of the next generation SDN datapath models. Avoiding key issues such as unnecessary combinatorial explosions to substantially reduce datapath table sizes, multi-table pipelining is essential for making SDN practical. At the same time, the introduction of multi-tables also adds additional SDN programming tasks including designing effective layout of pipelines and populating the content of multiple tables.

In this work, we investigate how to automatically derive and populate multi-table pipelines from datapath-oblivious algorithmic policies (AP) [1]. We choose the algorithmic policies model because it is highly flexible and hence poses minimal constraints on SDN programming. The model is also general, and hence, can be used to express other models. As a result of the generality, if we can compute high-quality multi-table pipelines for algorithmic policies, we can convert other policies into malgorithmic policies, and use algorithmic policies as a powerful intermediate language for implementing other high-level SDN programming models.

On the other hand, effectively utilizing multi-table pipelines from algorithmic policies can be extremely challenging, because APs are expressed in a general-purpose programming language with arbitrary complex control structures (*e.g.*, conditional statements, loops), and the control structures of APs can be completely oblivious to the existence of multi-tables. Hence, it is not clear at all whether one can effectively program multi-table pipelines from such APs. We refer to this as the *oblivious multi-table programming challenge*.

The main contribution of this paper is the development of Magellan, the first system that addresses the oblivious multi-table programming challenge. The core of Magellan consists of two novel, substantial algorithms: the map-explore algorithm and the table design algorithm. Specifically, the map-explore algorithm conducts a novel, efficient form of hybrid *symbolic* (map) and *direct* (explore) execution of a multi-table oblivious program written in a general-purpose language, resulting in a data-structure called *explorer graph*. The table design algorithm partitions the dataflow graph of program and merges tables.

2. ARCHITECTURE COMPONENTS

The high-level objective of Magellan is simple to state: automate the tasks of table design and table population for general-purpose APs.

To achieve the goal, Magellan introduces a sophisticated compiler and runtime system shown in Figure 1.

- The static analysis and transformation proceeds in two steps: native compilation, and bytecode rewriting. A native compiler converts the user program to an objective code format to remove the extra complexity of high-level programming language which makes program analysis complex. We refer to the result as the bytecode. The purpose of bytecode rewriting is first to identify and organize the compact-mappable statements which can be represented

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil
 ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2959064

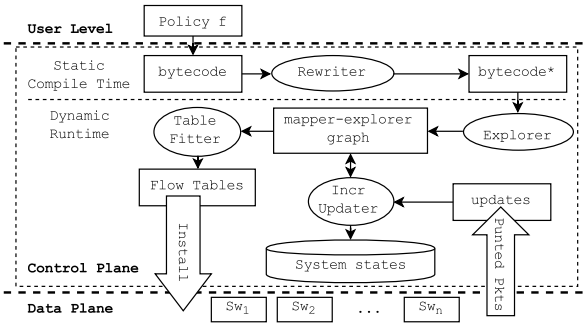


Figure 1: Magellan system components and work flow.

by a compact flow table with a small range output into `brk` statements and others into $xblocks$.

- The dynamic runtime part has two key components: Explorer and Table Fitter. The goal of Explorer is to generate the mapper-explorer graph whose nodes indicating instructions and links indicating control flow between instructions. By exploration of the program, a node in the graph includes all possible values for inputs and outputs of the instruction generating the node. After generating the mapper-explorer graph, a simple algorithm can generate flow tables from the graph. Finally, a table design algorithm in the Table Fitter merges tables to reduce the number since the number of flow tables is limited in real switches. In the next section, we will show a main step by using an example in the table design. The Incremental Updater will directly update mapper-explorer graph and then Magellan recomputes the content of flow tables.

3. EXAMPLE: AP TO DATAFLOW GRAPH

Here we give an example to show the translation from an AP program to a dataflow graph which is used in the table design. Bellow is the AP example:

```
// Program: Static-Example
onPacket(p) {
 x = macSrc;
 if (x > 4) {y = hostTable[macDst];} else {y = 1;}
 egress = [2 * y]; }
```

To remove language-specific constructs and simplify program analysis, we convert programs to a generic, simple, streamlined labeled instruction set (IR)) that uses conditional and unconditional jumps for all control flows, and also we do a simple compile optimizaiton to replace all x with macSrc and remove x = macSrc:

```
L2: cjump (macSrc > 4) L3 L5
L3: y = hostTable[macDst]
L4: jump L6
L5: y = 1
L6: egress = [2 * y]
```

In order to generate Magellan dataflow graph, we also need to convert jump statements (belong to control flow) to data dependency. So we introduce guard variable (g in the following example):

```
L1: g = (macSrc > 4)
L2: if g: y = hostTable[macDst]
L3: if !g: y = 1
L4: egress = [2 * y]
```

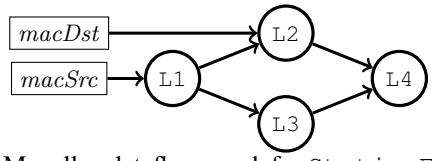


Figure 2: Magellan dataflow graph for `Static-Example`.

System	Hosts	Rules	Time (s)	Med RTT(ms)
POX	70	18787	96	9.7
Floodlight	70	4699	37	2.1
OpenDaylight	70	4769	32	0.6
Pyretic	70	-	> 1500	
Magellan	70	142	25	0.3
POX	140	13107	389	11.9
Floodlight	140	16451	200	6.1
OpenDaylight	140	19349	150	1.2
Pyretic	140	-	-	
Magellan	140	282	123	0.6

Figure 3: End-to-end performance comparison.

Then we generate the dataflow graph for this program as shown in Figure 2. The table design algorithm will partition the dataflow graph into regions, and merge all nodes in one region. The dataflow graph guarantees the merging is correct comparing with control flow graph.

4. PRELIMINARY EVALUATIONS

We compare Magellan with a range of state-of-the-art commercial and academic SDN systems, including OpenDaylight, Floodlight, POX, and Pyretic. We evaluate all systems using Open vSwitch (OVS) version 2.0.2, and conduct evaluations in a range of settings. In this poster, we report the results of the L2-learning-and-routing policy, because it is available in each system from the system's authors (with minor variations). Specifically, for each system, after allowing appropriate initialization of hosts and controller, we perform an all-to-all ping among the hosts, record the RTT of each ping, measure the time for all hosts to complete this task. After completing the task, we retrieve and count all Openflow rules installed in the switch. Figure 3 lists the number of rules, task completion time, and median ping RTT [1]for each system with $H = 70$ and $H = 140$ hosts and. We observe that for 70 hosts, Magellan uses 33x fewer rules than OpenDaylight and Floodlight, while for 140 hosts, Magellan uses between 46-68x fewer rules than other systems. This rule compression is due to leveraging multi-table pipelines: all other systems generate rules into a single table, and therefore generate approximately H^2 rules, while Magellan generates approximately only $2 * H$ rules.

5. REFERENCES

[1] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying sdn programming using algorithmic policies. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 87–98. ACM, 2013.

[1]Tests of Pyretic at both 70 and 140 hosts failed and these measurements are therefore omitted.

FAST: A Simple Programming Abstraction for Complex State-Dependent SDN Programming *

Kai Gao
Tsinghua Univeristy
gaok12@mails.tsinghua.edu.cn

Chen Gu
Tongji University
gc19931011jy@gmail.com

Qiao Xiang
Tongji/Yale University
qiao.xiang@cs.yale.edu

Yang Richard Yang
Tongji/Yale University
yry@cs.yale.edu

Jun Bi
Tsinghua University
junbi@tsinghua.edu.cn

ABSTRACT

Handling state dependencies is a major challenge in modern SDN programming, but existing frameworks do not provide sufficient abstractions nor tools to address this challenge. In this paper, we propose a novel, high-level programming abstraction and implement the *Function Automation SysTem (FAST)*. With the two key features, *i.e.*, *automated state dependency tracking* and *efficient re-execution scheduling*, we demonstrate that FAST substantially simplifies state-dependent SDN programming and boosts the performance.

CCS Concepts

•Networks → Programming interfaces; Network control algorithms;

Keywords

SDN, Programming abstraction, State dependency

1. INTRODUCTION

A common characteristic of many network control-plane functions is that their computation depends on network states. For example, basic routing algorithms such as the shortest path depend on the topology, QoS-based routing depends on both topology and the current resource allocations, and security functions (*e.g.*, access control, policy-based forwarding) depend heavily on the current state of configured security policies.

Implementing aforementioned control-plane functions in a correct and efficient manner, however, can be complex. Existing frameworks such as OpenDaylight and ONOS recognize this complexity and introduce datastores and broker services with a powerful pub/sub API to enable state dependent programming. However, many programming complexities remain.

Firstly, it is still the responsibility of the programmers to handle the complexity of identifying dependent data and subscribing to their changes. This, however, is not trivial. For example, a production implementation of the Dijkstra algorithm using a priority queue[2] touches only a subset of links, depending on the specific source and destination locations. Missing a subscription to a touched link can lead to *inconsistency* between the calculated path and the current network topology. On the other hand, naive approaches to simplify the programming by oversubscribing to changes on all links in the whole topology, lead to *unnecessary* re-executions.

Secondly and more importantly, it can be more complex than one might think to handle state changes correctly. Consider a control function f which depends on a certain state variable v_s. When the value of v_s changes, the naive solution, which simply update the outcome by re-executing f using the new value, can lead to errors. Consider the QoS routing example which finds a path and makes bandwidth reservations along the path. Assume a link on the path fails, and the function is to be re-executed to find an alternative. It is important that the previously reserved bandwidth be released first, to avoid "garbage" bandwidth reservations. However, releasing the reservations may trigger other data change events and lead to cascading effects.

We address the preceding complexities by exploring a novel and substantially simpler control-plane program-

* This research is supported by the ***National Science Foundation*** (CNS-1018502, CC*IIE 1440745), the ***National Natual Science Foundation of China*** (No.61472213 and No.61502267) and the ***Google Faculty Research Award***.

SIGCOMM '16, August 22-26, 2016, Florianopolis , Brazil
 ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2960424

ming abstraction where state changes are transparent to programmers. The tasks to manage state dependency tracking and schedule re-executions are automated by our runtime system, FAST.

2. PROGRAMMING ABSTRACTION

There are two different views for a programmer in FAST, as demonstrated in Figure 1.

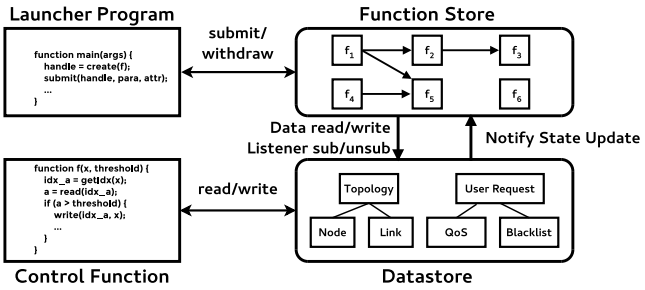


Figure 1: Programmers' view of FAST.

The **datastore** provides the API for **control functions** to access and modify persistent network state. As FAST handles data changes internally, the control function f contains no event driven code and is programmed to execute with the current state.

The **function store** is the key concept of FAST. It is built on top of the datastore and manages the meta information about submitted control functions for better analysis and scheduling. The **launcher programs** can submit certain control functions to the FAST function store and withdraw them when appropriate. They can also configure the *parameters* passed to the control functions and set *attributes* to specify the behaviours of certain control functions. For example, launchers can specify the *precedence* relationship between different control functions to create a workflow, or using *groups* to enforce that all updates in a group be committed as a single transaction.

3. SYSTEM COMPONENTS

FAST function store is driven by a sophisticated runtime, shown in Figure 2, which includes novel algorithms to automate complex tasks of state-dependent programming. Specifically, the runtime system consists of the following key components.

Event dispatcher Classify events such as state changes and submission/withdrawal of control functions, and dispatch them to corresponding components.
Restoration module Restore the system state after certain events so that the current state is consistent with the status of the function instances.
Min-makespan scheduler Schedule execution orders to minimize the maximum sum of control plane computing latency and the control path outbound latency.
Instance executor Execute and monitor the control functions to obtain fine-grained state dependencies.

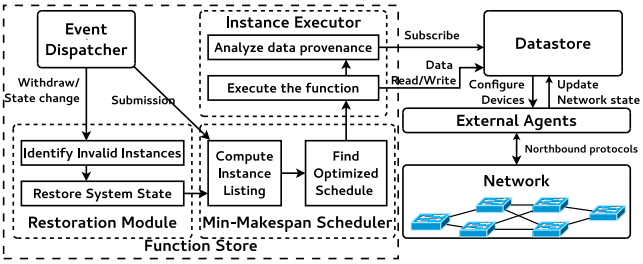


Figure 2: Runtime system for FAST.

Datastore Store the persistent network states and also act as the the storage for function metadata.

4. PRELIMINARY EVALUATION

We implement a prototype of FAST and evaluate its performance using Open vSwitch to simulate real topologies. We demonstrate the efficiency of FAST arisen from state awareness and fine-grained dependency tracking.

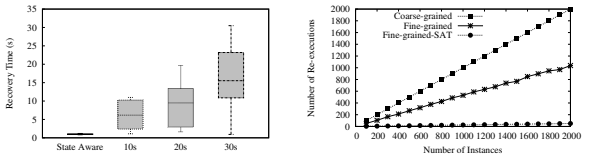
Figure 3a demonstrates the recovery time of restoring end-to-end connectivities after the control plane is notified of a random link failure event. We compare a routing function using FAST with periodic path computation engines in Floodlight[1] with various timeout values. We observe that being state-aware substantially improves the end-to-end recovery time.

Figure 3b shows how the number of re-executions changes with the number of running functions for different state tracking strategies. From the result, we see that re-executions both increase linearly but the slope with fine-grained state tracking is much smaller as expected, because FAST only monitors the touched links so that some link change events will not trigger re-executions. In addition, we find that allowing users to specify the satisfiability attribute of instances in FAST substantially reduces the number of re-executions, which is approximately 1/40 and 1/20 of those caused by the coarse-grained system and FAST without SAT attributes.

5. REFERENCES

[1] Floodlight OpenFlow Controller. http://floodlight.openflowhub.org/.

[2] F. B. Zhan. Three fastest shortest path algorithms on real road networks: Data structures and procedures. *Journal of geographic information and decision analysis*, 1(1):69–82, 1997.



(a) Benefits of state-awareness (b) Benefits of being fine-grained

Figure 3: Evaluation results.

2016 IEEE 24th International Conference on Network Protocols (ICNP)
Poster Paper

ORSAP: Abstracting Routing State on Demand

Kai Gao†, Chen Gu*, Qiao Xiang‡, Xin Wang*, Y. Richard Yang‡, Jun Bi†
*Tongji University †Tsinghua University ‡Yale University

***Abstract*—Providing an interface for network applications to access network state, Software-Defined Networking (SDN) northbound API protocol is the foundation for the development of programmable networks with adaptive applications. However, with the growing network scale and applications' need for routing state at multi-domain level, feeding complete routing states to applications would jeopardize their scalability and network providers' privacy. Thus a good routing state abstraction is needed, which must be on-demand so that different applications can receive customized abstract state suiting their needs. Moreover, it must be minimal and equivalent, *i.e.*, containing all the necessary information for applications to make decisions as the complete state does with no redundancy. Current routing state abstractions are not on-demand, and adopt extreme aggregation approaches (*e.g.*, the *big switch*) to provide a minimal abstraction with the price of severe information loss. For instance, bottleneck links shared between flows are concealed, leading applications to make sub-optimal decisions. In this paper, we design *ORSAP*, the first on-demand routing state abstraction protocol, through which network applications can describe their demands while Internet service providers can provide the on-demand minimal equivalent routing state accordingly. ORSAP ensures applications' scalability, protects network providers' privacy, and significantly reduces the traffic to disseminate the information. Experiments show that with ORSAP and the abstraction engine we introduced in this paper, one can achieve a state abstraction ratio of up to 60% with an extremely low computation time even with large networks and complex application queries.**

I. Introduction

Providing applications with the network information including the routes and the attributes of the links on the routes is fundamental for developing adaptive network applications. Thus many SDN controllers, such as ONOS [1] and OpenDaylight [2], have provided the API for applications to access the global routing state.

However, feeding a *complete routing state* to applications causes severe privacy leaks for network service providers and brings significant challenges on the scalability of applications. And these drawbacks are magnified with the growing network size and applications' needs for routing state on the multi-domain level. It is important that an *abstract routing state* be provided, which should not only guarantee privacy and scalability but also reduce the load of information updates.

Despite these substantial benefits, designing such an routing state abstraction is a non-trivial task because a series of challenges need to be resolved: *1)* The abstraction must be *on-demand*. Different applications may have different requirements on the routing state. A good abstraction should allow applications to specify their needs and return customized abstract routing states. *2)* The abstraction must ensure that for any routing state query, there is *no information loss* in the returned abstraction. In other words, applications should be able to make the same decision based on the returned routing state as they do based on the complete routing state. *3)* The abstraction must be *scalable*. Returning a complete routing state for a given query is infeasible because the number of routes between a source-destination pair grows exponentially with the increase of the scales of network and attached properties. *4)* The abstraction must protect the *privacy* of network providers. Exposing the complete routing state to applications will cause massive information leaks, making the network more vulnerable.

Existing routing abstractions usually adopt two strategies. One is the *big switch* approach, in which a network is abstracted as a single node with all the details hidden from applications [3]–[5]. The other one is the *aggregation* approach, in which the network is divided into several groups and each group is aggregated as a single node [6], [7]. Though both approaches provide certain support for application scalability and network provider privacy, they are application-oblivious, and suffer from severe routing information loss, *e.g.*, bottleneck links shared between flows are concealed from applications, leading them to make sub-optimal decisions. Thus neither of them is the correct direction to an efficient on-demand routing state abstraction.

In this work, we explore the feasibility and benefits of providing such an on-demand routing state abstraction. In particular, we propose ORSAP, the first *On-demand Routing State Abstraction Protocol*. ORSAP allows network applications to specify their needs on routing state, and network administrators to provide network information based on the policies. We have also built a prototype providing the on-demand routing state abstraction service with ORSAP. A core component of the prototype system is a routing state abstraction engine that computes the abstract routing state with the complete network state, the set of paths computed from an application's query request and the network policies specified by administrators. The engine is capable of providing the *minimal equivalent* routing state in the sense that the result contains all the necessary information for the application to make decisions as the complete routing state does, and with no redundant information.

II. Protocol Design

The protocol is based on the Application-Layer Traffic Optimization [4] protocol, with the additional extensions:

Flow requests Applications must provide the interested flows as well as the desired attributes using the `FlowRequest` as demonstrated below. Each flow is specified by its header field

978-1-5090-3281-5/16/$31.00 ©2016 IEEE

values and is uniquely identified by its UUID. Applications can also provide additional application-specific constraints, which can be leveraged to further reduce the size of the abstract routing state.

```
object {
    FlowSpecMap flows;
    JSONString  attributes<0..*>;
    [JSONString  extra-constraints<0..*>;]
} FlowRequest;

object-map {
        UUID -> FlowSpec;
} FlowSpecMap;

object {
    IPv4Address source;
    IPv4Address destination;
        ...
} FlowSpec;
```

Routing state encoding Applications will receive the routing state in a format that is very similar to path vectors where a list of traversed links is provided for each flow. The representation is more compact to support links with multiple appearances and fine-grained attributes. Bottlenecks can also be identified where the paths for different flows have intersects.

The redundancy elimination algorithm One principle in designing ORSAP is to reduce the data size as much as possible while maintaining the *equivalence* condition. Our RE algorithm can identify and eliminate all redundant constraints and computes the *minimal equivalent* routing state, which contains the *minimal* number of links among all possible routing states that are equivalent to the complete routing state.

III. System Overview

Figure 1 gives an overview of our ORSAP system, which leverages the power of SDN and provides a general framework of computing on-demand abstract routing state. The system consists of five major components:

Path computation engine The path computation engine (PCE) finds the paths for the flows specified by the user request. It can be very efficient for SDN controllers with centralized routing algorithms.

Constraint compiling engine The constraint compiling engine (CCE) takes the policies from the network administrators and compiles them into *linear* constraints on the nodes/links.

NOS Adapter The NOS adapter provides a unified interface for the system to access the network information such as topologies and attributes for nodes/links.

Routing state assembler The routing state assembler combines the information collected by PCE, CCE and the NOS adaptor to construct the complete routing state.

The RE Abstraction engine The abstraction engine is the core of our framework. It uses the *redundancy elimination* algorithm to find the minimal equivalent routing state and can leverage parallel processing to reduce the execution time.

IV. Preliminary Evaluations

We evaluate our prototype system and demonstrate the preliminary results in Figure 2, where the compression ratio

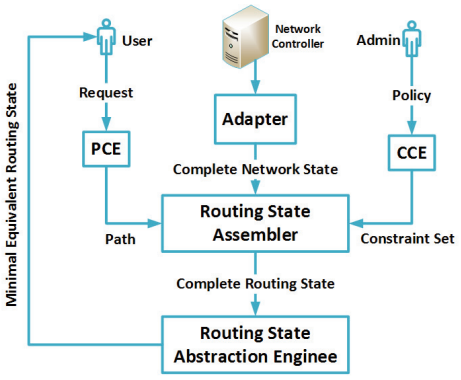
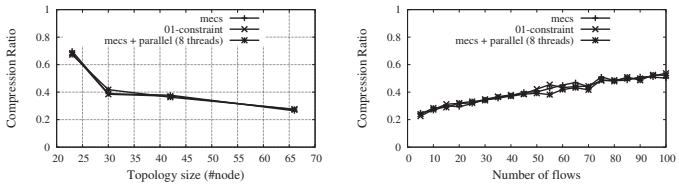


Fig. 1: Architecture of the ORSAP system.



(a) (b)

Fig. 2: Factors on the compression ratio.

denotes the portion of non-redundant links after the abstraction. As we can see from Figure 2(a), the compression ratio decreases as the topology size grows, which indicates that the ORSAP service has improved the scalability. Meanwhile, from Figure 2(b) we can infer that even the compression ratio grows with more flows, it can still reduce more than 40% of the complete routing state for flow requests of a moderate size.

V. Acknowledgement

This project is supported by the *National Science Foundation* (CNS-1018502, CC*IIE1440745), the *National Natural Science Foundation of China* (No.61472213, No.61502267) and the *Google Faculty Research Award*.

References

[1] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[2] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven SDN controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[3] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4), RFC 4271," 2006.

[4] R. Alimi, R. Penno, S. Previdi, A. Tian, Y.-S. Wang, and Y. R. Yang, "The ALTO protocol, RFC 7285," 2014.

[5] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *ACM CoNEXT'13*.

[6] T. Vu, A. Baid, H. Nguyen, and D. Raychaudhuri, "EIR: Edge-aware Interdomain Routing protocol for the future mobile Internet," *WINLAB, Rutgers University, Tech. Rep. WINLAB-TR-414*, 2013.

[7] W. C. Lee, "Topology aggregation for hierarchical routing in ATM networks," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 2, pp. 82–92, 1995.

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 23, 2020

K. Gao
Sichuan University
Y. Lee
Huawei
S. Randriamasy
Nokia Bell Labs
Y. Yang
Yale University
J. Zhang
Tongji University
July 22, 2019

ALTO Extension: Path Vector
draft-ietf-alto-path-vector-08

Abstract

This document defines an ALTO extension that allows a resource to provide not only preferences of network paths but also correlations of network paths, including aggregations of network components and their properties on the paths between different PIDs or endpoints. The extended information can be used to improve the robustness and performance for applications in some new usage scenarios, such as high-speed data transfers and traffic optimization using in-network storage and computation. This document introduces abstract network element (ANE) as an abstraction for aggregations of network components. It extends the base protocol and the Unified Property extension to enable the capability of encoding such information in a "path vector", i.e., an array of ANEs that are traversed by traffic from a source to a destination.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The ALTO protocol is aimed to provide applications with knowledge of the underlying network topologies from the point of views of ISPs. The base protocol [RFC7285] defines cost maps and endpoint cost services that expose the preferences of network paths for a set of source and destination pairs.

While the preferences of network paths are already sufficient for a wide range of applications, new application traffic patterns and new network technologies are emerging that are well beyond the domain for which existing ALTO maps are engineered, including but not limited to:

Very-high-speed data transfers: Applications, such as Content Distribution Network (CDN) overlays, geo-distributed data centers and large-scale data analytics, are foundations of many Internet services today and have very large traffic between a source and a destination. Thus, the interference between traffic of different source and destination pairs cannot be omitted, which cannot be provided by or inferred from existing ALTO base protocol and extensions.

In-network storage and computation: Emerging networking technologies such as network function virtualization and mobile edge computing provide storage and computation inside the network. Applications can leverage these resources to further improve their performance, for example, using in-network caching to reduce latency and bandwidth from a given source to multiple clients. However, existing ALTO extensions provide no map resources to discover available in-network services, nor any information to help ALTO clients determine how to effectively and efficiently use these services.

This document specifies a new extension to incorporate these newly emerged scenarios into the ALTO framework. The essence of this extension is that an ALTO server exposes correlations of network paths in additional to preferences of network paths.

The correlations of network paths are represented by path vectors. Each element in a path vector, which is referred to as an abstract network element (ANE), is the aggregation of network components on the path, such as routers, switches, links and clusters of in-network servers. If an abstract network element appears in multiple network paths, the traffic along these paths will join at this abstract network element and are subject to the corresponding resource constraints.

The availability of the path correlations by itself can help ALTO clients conduct better traffic scheduling. For example, an ALTO client can use the path correlations to conduct more intelligent end-to-end measurement and identify traffic bottlenecks.

By augmenting these abstract network elements with different properties, an ALTO server can provide a more fine-grained view of the network. ALTO clients can use this view to derive information such as shared risk resource groups, capacity regions and available in-network cache locations, which can be used to improve the robustness and performance of the application traffic.

2. Terminology

This document extends the ALTO base protocol [RFC7285] and the Unified Property Map extension [I-D.ietf-alto-unified-props-new]. In additional to the ones defined in these documents, this document also uses the following additional terms:

- Abstract network element (ANE): An abstract network element is an abstraction of network components. It can be a link, a middleboxes, a virtualized network function (VNF), etc., or their aggregations. In a response, each abstract network element has a unique ANE identifier.

- Path vector: A path vector is an array of ANE identifiers. It presents an abstract network path between source/destination points such as PIDs or endpoints.

- Path vector resource: A path vector resource refers to an ALTO resource which supports the extension defined in this document.

-

- Path vector response: A path vector response refers to the multipart/related message returned by a path vector resource. It consists of a path vector part, i.e., the (endpoint) cost map part which contains the path vector information, and a property map part.

3. Use Cases

This section describes typical use cases of the path vector extension. These use cases provide new usage scenarios of the ALTO framework.

3.1. Shared Risk Resource Group

Consider an application which controls 4 end hosts (eh1, eh2, eh3 and eh4), which are connected by an ISP network with 5 switches (sw1, sw2, sw3, sw4 and sw5) and 5 links (l1, l2, l3, l4 and l5), as shown in Figure 1. Assume the end hosts are running data storage services and some analytics tasks, which requires high data availability. In order to determine the replica placement, the application must know how the end hosts will be partitioned if certain network failures happen.

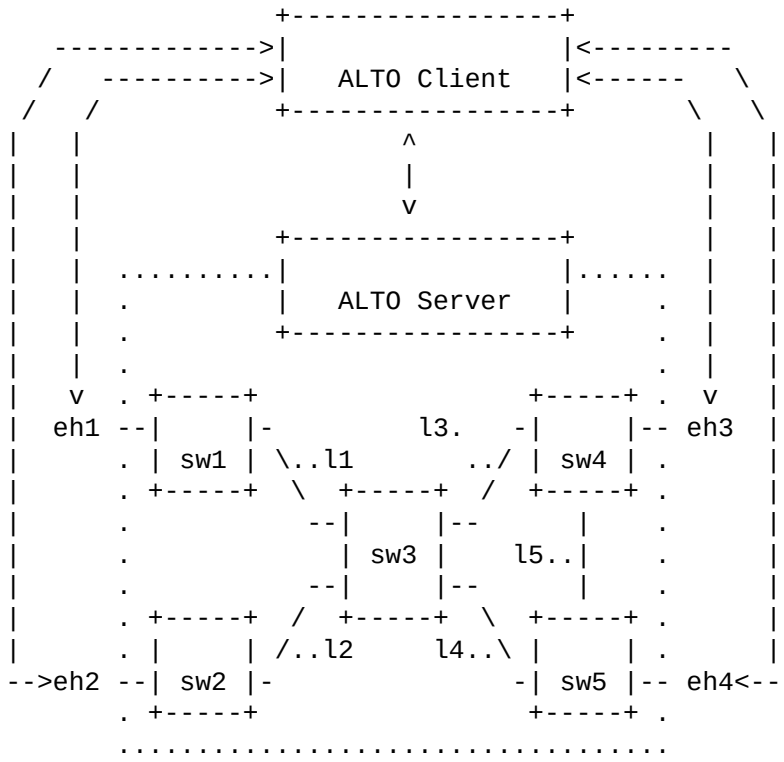


Figure 1: Topology for the Shared Risk Resource Group and the Capacity Region Use Cases

For that purpose, the application uses an ALTO client, which communicates with an ALTO server provided by the ISP network. Since the Endpoint Cost Service with only scalar cost values cannot provide essential information for the application, thus, both the client and the server have the path vector extension enabled.

Assume the ISP uses shortest path routing. For simplicity, consider the data availability on eh4. The network components on the paths from all other end hosts to eh4 are as follows:

```
eh1->eh4: sw1, l1, sw3, l4, sw5
eh2->eh4: sw2, l2, sw3, l4, sw5
eh3->eh4: sw4, l5, sw5
```

These network components can be categorized into 5 categories:

1. Failure will only disconnect eh1 to eh4: sw1, l1.

2. Failure will only disconnect eh2 to eh4: sw2, l2.

3. Failure will only disconnect eh3 to eh4: sw4, l5.

4. Failure will only disconnect eh1 and eh2 to eh4: sw3, l4.

5. Failure will disconnect eh1, eh2 and eh3 to eh4: sw5.

The ALTO server can then aggregate sw1 and l1 as an abstract network element, ane1. By applying the aggregation to the categories, the response may be as follows:

```
eh1->eh4: ane1, ane4, ane5
eh2->eh4: ane2, ane4, ane5
eh3->eh4: ane3, ane5
```

Thus, the application can still derive the potential network partitions for all possible network failures without knowing the exact network topology, which protects the privacy of the ISP.

3.2. Capacity Region

This use case uses the same topology and application settings as in Section 3.1 as shown in Figure 1. Assume the capacity of each link is 10 Gbps, except l5 whose capacity is 5 Gbps . Assume the application is running a map-reduce task, where the optimal traffic scheduling is usually referred to the co-flow scheduling problem. Consider a simplified co-flow scheduling problem, e.g., the first stage of a map-reduce task which needs to transfer data from two data nodes (eh1 and eh3) to the mappers (eh2 and eh4). In order to optimize the job completion time, the application needs to determine the bottleneck of the transfers.

If the ALTO server encodes the routing cost as bandwidth of the path, the client will obtain the following information:

```
eh1->eh2: 10 Gbps,
eh1->eh4: 10 Gbps,
eh3->eh2: 10 Gbps,
eh3->eh4:  5 Gbps.
```

However, it does not provide sufficient information to determine the bottleneck. With the path vector extension, the ALTO server will first return the correlations of network paths between eh1, eh3 and eh2, eh4, as follows:

```
eh1->eh2: ane1 (l1), ane2 (l2),
eh1->eh4: ane1 (l1), ane4 (l4),
eh3->eh2: ane3 (l3), ane2 (l2),
eh3->eh3: ane5 (l5).
```

Meanwhile, the ALTO server can also return the capacity of each ANE:

```
ane1.capacity = 10 Gbps,
ane2.capacity = 10 Gbps,
ane3.capacity = 10 Gbps,
ane4.capacity = 10 Gbps,
ane5.capacity =  5 Gbps.
```

With the correlation of network paths and the link capacity property, the client is able to derive the capacity region of data transfer rates. Let x1 denote the transfer rate of eh1->eh2, x2 denote the rate of eh1->eh4, x3 denote the rate of eh3->eh2, and x4 denote the rate of eh3->eh4. The application can derive the following information from the responses:

	eh1->eh2	eh1->eh4	eh3->eh2	eh3->eh4	capaity
ane1	1	1	0	0	10 Gbps
ane2	1	0	1	0	10 Gbps
ane3	0	0	1	0	10 Gbps
ane4	0	1	0	0	10 Gbps
ane5	0	0	0	1	5 Gbps

Specifically, the coefficient matrix on the left hand side is the transposition of the matrix directly derived from the path vector part, and the right-hand-side vector is directly derived from the property map part. Thus, the bandwidth constraints of the data transfers are as follows:

```
x1 + x2 <= 10 Gbps (ane1),
x1 + x3 <= 10 Gbps (ane2),
x2 + x3 <= 10 Gbps (ane3),
x2      <= 10 Gbps (ane4),
x4      <=  5 Gbps (ane5).
```

3.3. In-Network Caching

Consider an application which controls 3 end hosts (eh1, eh2 and eh3), which are connected by an ISP network and the Internet, as shown in Figure 2. Assume two clients at end hosts eh2 and eh3 are downloading the same data from a data server at eh1. Meanwhile, the network provider offers an in-network caching service at the gateway.

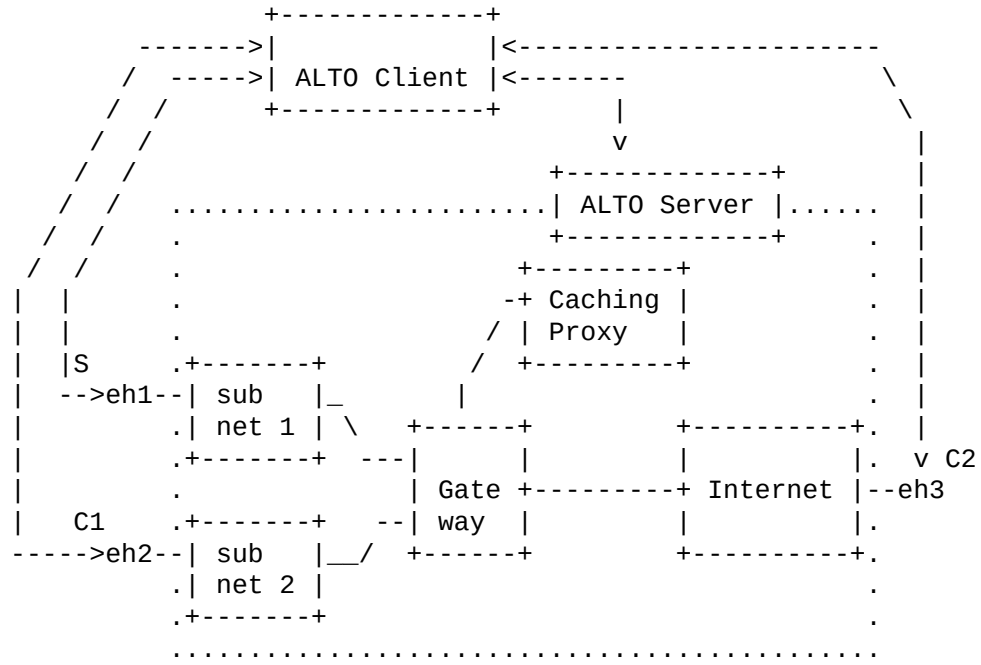


Figure 2: Topology for the In-Network Caching Use Case.

With the path vector extension enabled, the ALTO server can expose two types of information

Without the traffic correlation information, the ALTO client cannot know whether or how the traffic goes through the proxy. For example, if subnet1 and subnet2 are directly connected and the traffic from eh1 to eh2 bypasses the gateway, the in-network cache can only be used for traffic from C2 to S and is less effective.

4. Overview

This section gives a top-down overview of approaches adopted by the path vector extension, with discussions to fully explore the design space. It is assumed that readers are familiar with both the base protocol [RFC7285] and the Unified Property Map extension [I-D.ietf-alto-unified-props-new].

4.1. Workflow

The workflow of the base ALTO protocol consists of one round of communication: An ALTO client sends a request to an ALTO server, and the ALTO server returns a response, as shown in Figure 3. Each response contains only one type of ALTO resources, e.g., network maps, cost maps, or property maps.

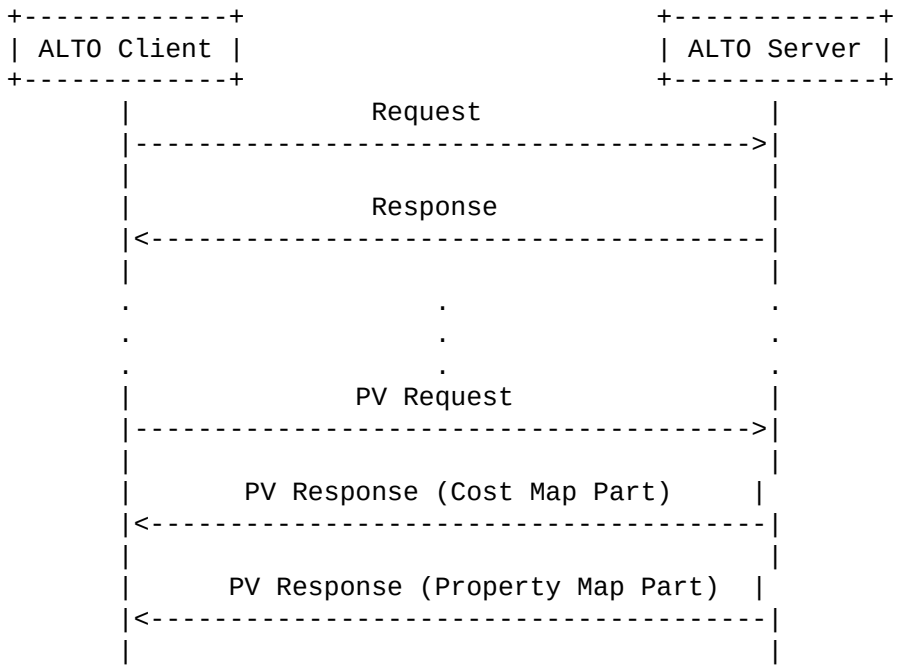


Figure 3: Information Exchange Process of the base ALTO Protocol and the Path Vector Extension

The path vector extension, on the other hand, CAN be decomposed to two types of information resources. First, path vectors, which represent the correlations of network paths for all <source, destination> pairs in the requst, CAN be encoded as an (endpoint) cost map with an extended cost type. Second, properties associated with the ANEs CAN be encoded as a property map.

Instead of making two consecutive queries, however, the path vector extension adopts a workflow which also consists of only one round of communication, based on the following reasons:

1. ANE Computation Flexibility. For better scalability, flexibility and privacy, Abstract Network Elements MAY be constructed on demand, and potentially based on the properties (See Section 4.2 for more details). If sources and destinations are not in the

same request as the properties, an ALTO server either CANNOT construct ANEs on-demand, or MUST wait until both requests are received.

2. Server Scalability. As ANEs are constructed on demand, mappings of each ANE to its underlying network devices and resources CAN be different in different queries. In order to respond to the second request correctly, an ALTO server MUST store the mapping of each path vector request until the client fully retrieves the property information, which CAN substantially harm the server scalability and potentially lead to Denial-of-Service attacks.

Thus, the path vector extension encapsulates all essential information in one request, and returns both path vectors and properties associated with the ANEs in a single response. See Section 4.3 for more details.

4.2. Abstract Network Element

A key design in the path vector extension is abstract network element. Abstract network elements can be statically generated, for example, based on geo-locations, OSPF areas, or simply the raw network topology. They CAN also be generated dynamically, based on a client's request. This on-demand ANE generation allows for better scalability, flexibility and privacy enhancement.

Consider an extreme case where the client only queries the bandwidth between one source and one destination in the topology shown in Figure 4. Without knowing in prior the desired property, an ALTO server MAY need to include all network components on the paths for high accuracy. However, with the prior knowledge that the client only asks for the bandwidth information, an ALTO server CAN either 1) selectively pick the link with the smallest available bandwidth, or 2) dynamically generate a new ANE whose available bandwidth is the smallest value of the links' on the path. Thus, an ALTO server can provide accurate information with very little leak of its internal network topology. ANEs MAY also be constructed based on algebraic aggregations, please see [TON2019] for more details.

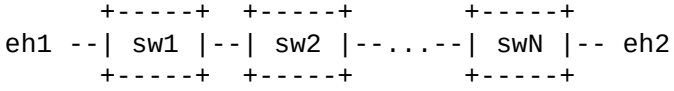


Figure 4: Topology for Dynamic ANE Example.

An ANE is uniquely identified by an ANE identifier (see Section 5.1) in the same response. However, since ANEs CAN be generated dynamically, an ALTO client MUST NOT assume that ANEs with the same

identifier but from different queries refer to the same aggregation of network components. This approach simplifies the management of ANE identifiers at ALTO servers, and increases the difficulty to infer the real network topology with cross queries. It is RECOMMENDED that the identifiers of statically generated ANEs be anonymized in the path vector response, for example, by shuffling the ANEs and shrinking their identifier space to [1, N], where N is the number of ANEs etc.

4.3. Protocol Extensions

Section 4.1 has well articulated the reasons to complete the information exchange in a single round of communication. This section introduces the three major extended components to the base ALTO protocol and the Unified Property Map extension, as shown in Table 1.

Component	IRD	Request	Response
Path Vector Cost Type	Yes	Yes	Yes
Property Negotiation	Yes	Yes	Yes
Multipart Message	Yes	No	Yes

Table 1: Extended Components and Where They Apply.

4.3.1. Path Vector Cost Type

Existing cost modes defined in [RFC7285] allow only scalar cost values. However, the path vector extension MUST convey vector format information. To fulfill this requirement, this document defines a new cost mode named "array", which indicates that the cost value MUST be interpreted as an array of JSONValue. This document also introduces a new cost metric "ane-path" to convey an array of ANE identifiers.

The combination of the "array" cost mode and the "ane-path" cost metric also complies best with the ALTO base protocol, where cost mode specifies the interpretation of a cost value, and cost metric conveys the meaning.

4.3.2. Property Negotiation

Similar to cost types, an ALTO server MAY only support a given set of ANE properties in a path vector information resource. Meanwhile, an ALTO client MAY only require a subset of the available properties. Thus, a property negotiation process is required.

This document uses a similar approach as the negotiation process of cost types: the available properties for a given resource are announced in the Information Resource Directory and more specifically, in a new capability called "ane-properties"; the selected properties SHOULD be specified in a new filter called "ane-properties" in the request body; the response MUST return and only return the selected properties for the ANEs in the response, if applicable.

4.3.3. Multipart/Related Message

Path vectors and the property map containing the ANEs are two different types of objects, but they need to be encoded in one message. One approach is to define a new media type to contain both objects, but this violates modular design.

This document uses standard-conforming usage of "multipart/related" media type defined in [RFC2387] to elegantly combine the objects. Path vectors are encoded as a cost map or an endpoint cost map, and the property map is encoded as a Unified Propert Map. They are encapsulated as parts of a multipart message. The modular composition allows ALTO servers and clients to reuse the data models of the existing information resources. Specifically, this document addresses the following practical issues using "multipart/related".

4.3.3.1. Identifying the Media Type of the Root Object

ALTO uses media type to indicate the type of an entry in the Information Resource Directory (IRD) (e.g., "application/alto-costmap+json" for cost map and "application/alto-endpointcost+json" for endpoint cost map). Simply putting "multipart/related" as the media type, however, makes it impossible for an ALTO client to identify the type of service provided by related entries.

To address this issue, this document uses the "type" parameter to indicate the root object of a multipart/related message. For a cost map resource, the "media-type" in the IRD entry MUST be "multipart/related" with the parameter "type=application/alto-costmap+json"; for an Endpoint Cost Service, the parameter MUST be "type=application/alto-endpointcost+json".

4.3.3.2. References to Part Messages

The ALTO SSE extension (see [I-D.ietf-alto-incr-update-sse]) uses "client-id" to demultiplex push updates. However, "client-id" is provided for each request, which introduces ambiguity when applying SSE to a path vector resource.

To address this issue, an ALTO server MUST assign a unique identifier to each part of the "multipart/related" response message. This identifier, referred to as a Part Resource ID (See Section 5.6 for details), MUST be present in the part message's "Resource-Id" header. The MIME part header MUST also contain the "Content-Type" header, whose value is the media type of the part (e.g., "application/alto-costmap+json", "application/alto-endpointcost+json", or "application/alto-propmap+json").

If an ALTO server provides incremental updates for this path vector resource, it MUST generate incremental updates for each part separately. The client-id MUST have the following format:

pv-client-id '.' part-resource-id

where pv-client-id is the client-id assigned to the path vector request, and part-resource-id is the "Resource-Id" header value of the part. The media-type MUST match the "Content-Type" of the part.

The same problem happens inside the part messages as well. The two parts MUST contain a version tag, which SHOULD contain a unique Resource ID. This document requires the resource-id in a Version Tag to have the following format:

pv-resource-id '.' part-resource-id

where pv-resource-id is the resource ID of the path vector resource in the IRD entry, and the part-resource-id has the same value as the "Resource-Id" header of the part.

4.3.3.3. Order of Part Messages

According to RFC 2387 [RFC2387], the path vector part, whose media type is the same as the "type" parameter of the multipart response message, is the root object. Thus, it is the element the application processes first. Even though the "start" parameter allows it to be placed anywhere in the part sequence, it is RECOMMENDED that the parts arrive in the same order as they are processed, i.e., the path vector part is always put as the first part, followed by the property map part. It is also RECOMMENDED that when doing so, an ALTO server SHOULD NOT set the "start" parameter, which implies the first part is the root object.

5. Basic Data Types

5.1. ANE Identifier

An ANE identifier is encoded as a JSON string. The string MUST be no more than 64 characters, and it MUST NOT contain characters other than US-ASCII alphanumeric characters (U+0030-U+0039, U+0041-U+005A, and U+0061-U+007A), the hyphen ("-", U+002D), the colon (":", U+003A), the at sign ("@", code point U+0040), the low line ("_", U+005F), or the "." separator (U+002E). The "." separator is reserved for future use and MUST NOT be used unless specifically indicated in this document, or an extension document.

The type ANEIdentifier is used in this document to indicate a string of this format.

5.2. Path Vector Cost Type

This document defines a new cost type, which is referred to as the "path vector" cost type. An ALTO server MUST offer this cost type if it supports the path vector extension.

5.2.1. Cost Metric: ane-path

This cost metric conveys an array of ANE identifiers, where each identifier uniquely represents an ANE traversed by traffic from a source to a destination.

5.2.2. Cost Mode: array

This cost mode indicates that every cost value in a cost map or an endpoint cost map MUST be interpreted as a JSON array object.

Note that this cost mode only requires the cost value to be a JSON array of JSONValue. However, an ALTO server that enables this extension MUST return a JSON array of ANEIdentifier (Section 5.1) when the cost metric is "ane-path".

5.3. ANE Domain

This document specifies a new ALTO entity domain called "ane" in addition to the ones in [I-D.ietf-alto-unified-props-new]. The ANE domain associates property values with the ANEs in a network. The entity in ANE domain is often used in the path vector by cost maps or endpoint cost resources. Accordingly, the ANE domain always depends on a cost map or an endpoint cost map.

5.3.1. Entity Domain Type

ane

5.3.2. Domain-Specific Entity Identifier

The entity identifier of ANE domain uses the same encoding as ANEIdentifier (Section 5.1).

5.3.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with ANEs.

5.4. New Resource-Specific Entity Domain Exports

5.4.1. ANE Domain of Cost Map Resource

If an ALTO cost map resource supports "ane-path" cost metric, it can export an "ane" typed entity domain defined by the union of all sets of ANE names, where each set of ANE names are an "ane-path" metric cost value in this ALTO cost map resource.

5.4.2. ANE Domain of Endpoint Cost Resource

If an ALTO endpoint cost resource supports "ane-path" cost metric, it can export an "ane" typed entity domain defined by the union of all sets of ANE names, where each set of ANE names are an "ane-path" metric cost value in this ALTO endpoint cost resource.

5.5. ANE Properties

5.5.1. ANE Property: Maximum Reservable Bandwidth

The maximum reservable bandwidth property conveys the maximum bandwidth that can be reserved for traffic from a source to a destination and is indicated by the property name "maxresbw". The value MUST be encoded as a numerical cost value as defined in Section 6.1.2.1 of [RFC7285] and the unit is bit per second.

If this property is requested but is missing for a given ANE, it MUST be interpreted as that the ANE does not support bandwidth reservation but have sufficiently large bandwidth for all traffic that traverses it.

5.5.2. ANE Property: Persistent Entity

The persistent entity property conveys the physical or logical network entities (e.g., links, in-network caching service) that are contained by an abstract network element. It is indicated by the property name "persistent-entity". The value is encoded as a JSON array of entity identifiers ([I-D.ietf-alto-unified-props-new]). These entity identifiers are persistent so that a client CAN further query their properties for future use.

If this property is requested but is missing for a given ANE, it MUST be interpreted as that no such entities exist in this ANE.

5.6. Part Resource ID

A Part Resource ID is encoded as a JSON string with the same format as that of the Resource ID (Section 10.2 of [RFC7285]).

WARNING: Even though the client-id assigned to a path vector request and the Part Resource ID MAY contain up to 64 characters by their own definition. Their concatenation (see Section 4.3.3.2) MUST also conform to the same length constraint. The same requirement applies to the resource ID of the path vector resource, too. Thus, it is RECOMMENDED to limit the length of resource ID and client ID related to a path vector resource to 31 characters.

6. Service Extensions

6.1. Multipart Filtered Cost Map for Path Vector

This document introduces a new ALTO resource called multipart filtered cost map resource, which allows an ALTO server to provide other ALTO resources associated to the cost map resource in the same response.

6.1.1. Media Type

The media type of the multipart filtered cost map resource is "multipart/related;type=application/alto-costmap+json".

6.1.2. HTTP Method

The multipart filtered cost map is requested using the HTTP POST method.

6.1.3. Accept Input Parameters

The input parameters of the multipart filtered cost map are supplied in the body of an HTTP POST request. This document extends the input parameters to a filtered cost map with a data format indicated by the media type "application/alto-costmapfilter+json", which is a JSON object of type PVReqFilteredCostMap, where:

```
object {
  [PropertyName ane-properties<0..*>;]
} PVReqFilteredCostMap : ReqFilteredCostMap;
```

with fields:

ane-properties: A list of properties that are associated with the ANEs. Each property in this list MUST match one of the supported ANE properties indicated in the resource's "ane-properties" capability. If the field is NOT present, it MUST be interpreted as an empty list, indicating that the ALTO server MUST NOT return any property in the unified property part.

6.1.4. Capabilities

The multipart filtered cost map resource extends the capabilities defined in Section 11.3.2.4 of [RFC7285]. The capabilities are defined by a JSON object of type PVFilteredCostMapCapabilities:

```
object {
  [PropertyName ane-properties<0..*>;]
} PVFilteredCostMapCapabilities : FilteredCostMapCapabilities;
```

with fields:

cost-type-names: The "cost-type-names" field MUST only include the path vector cost type, unless explicitly documented by a future extension. This also implies that the path vector cost type MUST be defined in the "cost-types" of the Information Resource Directory's "meta" field.

ane-properties: Defines a list of ANE properties that can be returned. If the field is NOT present, it MUST be interpreted as an empty list, indicating the ALTO server CANNOT provide any ANE property.

6.1.5. Uses

The resource ID of the network map based on which the PIDs in the returned cost map will be defined. If this resource supports "persistent-entities", it MUST also include ALL the resources that exposes the entities that MAY appear in the response.

6.1.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

The "Content-Type" header of the response MUST be "multipart/related" as defined by [RFC2387] with the following parameters:

type: The type parameter MUST be "application/alto-costmap+json". Note that [RFC2387] permits both parameters with and without the double quotes.

start: The start parameter MUST be a quoted string where the quoted part has the same value as the "Resource-ID" header in the first part.

boundary: The boundary parameter is as defined in [RFC2387].

The body of the response consists of two parts.

The first part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-costmap+json".

The body of the first part MUST be a JSON object with the same format as defined in Section 11.2.3.6 of [RFC7285]. The JSON object MUST include the "vtag" field in the "meta" field, which provides the version tag of the returned cost map. The resource ID of the version tag MUST follow the format in Section 4.3.3.2. The "meta" field MUST also include the "dependent-vtags" field, whose value is a single-element array to indicate the version tag of the network map used, where the network map is specified in the "uses" attribute of the multipart filtered cost map resource in IRD.

The second part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-propmap+json".

The body of the second part MUST be a JSON object with the same format as defined in Section 4.6 of [I-D.ietf-alto-unified-props-new]. The JSON object MUST include the "dependent-vtags" field in the "meta" field. The value of the "dependent-vtags" field MUST be an array of VersionTag objects as defined by Section 10.3 of [RFC7285]. The "vtag" of the first part MUST be included in the "dependent-vtags". If "persistent-entities" is requested, the version tags of the dependent resources that MAY expose the entities in the response MUST also be included. The PropertyMapData has one member for each ANE identifier that appears in the first part, where the EntityProps has one member for each property requested by the client if applicable.

6.2. Multipart Endpoint Cost Service for Path Vector

This document introduces a new ALTO resource called multipart endpoint cost resource, which allows an ALTO server to provide other ALTO resources associated to the endpoint cost resource in the same response.

6.2.1. Media Type

The media type of the multipart endpoint cost resource is "multipart/related;type=application/alto-endpointcost+json".

6.2.2. HTTP Method

The multipart endpoint cost resource is requested using the HTTP POST method.

6.2.3. Accept Input Parameters

The input parameters of the multipart endpoint cost resource are supplied in the body of an HTTP POST request. This document extends the input parameters to an endpoint cost map with a data format indicated by the media type "application/alto-endpointcostparams+json", which is a JSON object of type PVEndpointCostParams, where

```
object {
  [PropertyName ane-properties<0..*>;]
} PVReqEndpointcost : ReqEndpointcost;
```

with fields:

ane-properties: This document defines the "ane-properties" in PVReqEndpointcost as the same as in PVReqFilteredCostMap. See Section 6.1.3.

6.2.4. Capabilities

The capabilities of the multipart endpoint cost resource are defined by a JSON object of type PVEndpointcostCapabilities, which is defined as the same as PVFilteredCostMapCapabilities. See Section 6.1.4.

6.2.5. Uses

If a multipart endpoint cost resource supports "persistent-entities", the "uses" field in its IRD entry MUST include ALL the resources which exposes the entities that MAY appear in the response.

6.2.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

The "Content-Type" header of the response MUST be "multipart/related" as defined by [RFC2387] with the following parameters:

type: The type parameter MUST be "application/alto-endpointcost+json".

start: The start parameter MUST be a quoted string where the quoted part has the same value as the "Resource-ID" header in the first part.

boundary: The boundary parameter is as defined in [RFC2387].

The body consists of two parts:

The first part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-endpointcost+json".

The body of the first part MUST be a JSON object with the same format as defined in Section 11.5.1.6 of [RFC7285]. The JSON object MUST include the "vtag" field in the "meta" field, which provides the version tag of the returned endpoint cost map. The resource ID of the version tag MUST follow the format in Section 4.3.3.2.

The second part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-propmap+json".

The body of the second part MUST be a JSON object with the same format as defined in Section 4.6 of [I-D.ietf-alto-unified-props-new]. The JSON object MUST include the "dependent-vtags" field in the "meta" field. The value of the "dependent-vtags" field MUST be an array of VersionTag objects as defined by Section 10.3 of [RFC7285]. The "vtag" of the first part MUST be included in the "dependent-vtags". If "persistent-entities" is requested, the version tags of the dependent resources that MAY expose the entities in the response MUST also be included. The PropertyMapData has one member for each ANE identifier that appears in the first part, where the EntityProps has one member for each property requested by the client if applicable.

7. Examples

This section lists some examples of path vector queries and the corresponding responses. Some long lines are truncated for better readability.

7.1. Example: Information Resource Directory

Below is an example of an Information Resource Directory which enables the path vector extension. Some critical modifications include:

- The "path-vector" cost type (Section 5.2) is defined in the "cost-types" of the "meta" field.

- The "cost-map-pv" information resource provides a multipart filtered cost map resource, which exposes the Maximum Reservable Bandwidth ("maxresbw") property.

- The "http-proxy-props" information resource provides a filtered unified property map resource, which exposes the HTTP proxy entity domain (encoded as "http-proxy") and the "price" property. Note that HTTP proxy is NOT a valid entity domain yet and is used here only for demonstration.

- The "endpoint-cost-pv" information resource provides a multipart endpoint cost resource. It exposes the Maximum Reservable Bandwidth ("maxresbw") property and the Persistent Entity property ("persistent-entities"). The persistent entities MAY come from the "http-proxy-props" resource.

- The "update-pv" information resource provides the incremental update ([I-D.ietf-alto-incr-update-sse]) service for the "endpoint-cost-pv" resource.

```
{
  "meta": {
    "cost-types": {
      "path-vector": {
        "cost-mode": "array",
        "cost-metric": "ane-path"
      }
    }
  },
  "resources": {
    "my-default-networkmap": {
      "uri" : "http://alto.example.com/networkmap",
      "media-type" : "application/alto-networkmap+json"
    },
    "cost-map-pv": {
      "uri": "http://alto.example.com/costmap/pv",
      "media-type": "multipart/related;
                     type=application/alto-costmap+json",
      "accepts": "application/alto-costmapfilter+json",
      "capabilities": {
        "cost-type-names": [ "path-vector" ],
        "ane-properties": [ "maxresbw" ]
      },
      "uses": [ "my-default-networkmap" ]
    },
    "http-proxy-props": {
      "uri": "http://alto.example.com/proxy-props",
      "media-type": "application/alto-propmap+json",
      "accpets": "application/alto-propmapparams+json",
      "capabilities": {
        "mappings": {
          "http-proxy": [ "price" ]
        }
      }
    },
    "endpoint-cost-pv": {
      "uri": "http://alto.exmaple.com/endpointcost/pv",
      "media-type": "multipart/related;
                     type=application/alto-endpointcost+json",
      "accepts": "application/alto-endpointcostparams+json",
      "capabilities": {
        "cost-type-names": [ "path-vector" ],
        "ane-properties": [ "maxresbw", "persistent-entities" ]
      },
      "uses": [ "http-proxy-props" ]
    },
    "update-pv": {
      "uri": "http://alto.example.com/updates/pv",
```

```
        "media-type": "text/event-stream",
        "uses": [ "endpoint-cost-pv" ],
        "accepts": "application/alto-updatestreamparams+json",
        "capabilities": {
          "support-stream-control": true
        }
      }
    }
  }
```

7.2. Example: Multipart Filtered Cost Map

The following examples demonstrate the request to the "cost-map-pv" resource and the corresponding response.

The request uses the path vector cost type in the "cost-type" field. The "ane-properties" field is missing, indicating that the client only requests for the path vector but not the ANE properties.

The response consists of two parts. The first part returns the array of ANE identifiers for each source and destination pair. There are three ANEs, where "ane:L001" is shared by traffic from "PID1" to both "PID2" and "PID3".

The second part returns an empty property map. Note that the ANE entries are omitted since they have no properties (See Section 3.1 of [I-D.ietf-alto-unified-props-new]).

```
POST /costmap/pv HTTP/1.1
Host: alto.example.com
Accept: multipart/related;type=application/alto-costmap+json,
        application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-costmapfilter+json

{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "ane-path"
  },
  "pids": {
    "srcs": [ "PID1" ],
    "dsts": [ "PID2", "PID3" ]
  }
}

HTTP/1.1 200 OK
Content-Length: [TBD]
```

```
   Content-Type: multipart/related; boundary=example-1;
                 type=application/alto-costmap+json

   --example-1
   Resource-Id: costmap
   Content-Type: application/alto-costmap+json

   {
     "meta": {
       "vtag": {
         "resource-id": "cost-map-pv.costmap",
         "tag": "d827f484cb66ce6df6b5077cb8562b0a"
       },
       "dependent-vtags": [
         {
           "resource-id": "my-default-networkmap",
           "tag": "75ed013b3cb58f896e839582504f6228"
         }
       ],
       "cost-type": {
         "cost-mode": "array",
         "cost-metric": "ane-path"
       }
     },
     "cost-map": {
       "PID1": {
         "PID2": [ "ane:L001", "ane:L003" ],
         "PID3": [ "ane:L001", "ane:L004" ]
       }
     }
   }
   --example-1
   Resource-Id: propmap
   Content-Type: application/alto-propmap+json

   {
     "meta": {
       "dependent-vtags": [
         {
           "resource-id": "cost-map-pv.costmap",
           "tag": "d827f484cb66ce6df6b5077cb8562b0a"
         }
       ]
     },
     "property-map": {
     }
   }
```

7.3. Example: Multipart Endpoint Cost Resource

The following examples demonstrate the request to the "endpoint-cost-pv" resource and the corresponding response.

The request uses the path vector cost type in the "cost-type" field, and queries the Maximum Reservable Bandwidth ANE property and the Persistent Entity property.

The response consists of two parts. The first part returns the array of ANE identifiers for each valid source and destination pair.

The second part returns the requested properties of ANEs in the first part. The "ane:NET001" element contains an HTTP proxy entity, which can be further used by the client. Since it does not contain a "maxresbw" property, the client SHOULD assume it does NOT support bandwidth reservation but will NOT become a traffic bottleneck, as specified in Section 5.5.1.

```
POST /endpointcost/pv HTTP/1.1
Host: alto.example.com
Accept: multipart/related;
        type=application/alto-endpointcost+json,
        application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-endpointcostparams+json

{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "ane-path"
  },
  "endpoints": {
    "srcs": [ "ipv4:192.0.2.2" ],
    "dsts": [ "ipv4:192.0.2.89",
              "ipv4:203.0.113.45",
              "ipv6:2001:db8::10" ]
  },
  "ane-properties": [ "maxresbw", "persistent-entities" ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-2;
              type=application/alto-endpointcost+json

--example-2
Resource-Id: ecs
```

```
   Content-Type: application/alto-endpointcost+json

   {
     "meta": {
       "vtags": {
         "resource-id": "endpoint-cost-pv.ecs",
         "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
       },
       "cost-type": {
         "cost-mode": "array",
         "cost-metric": "ane-path"
       }
     },
     "endpoint-cost-map": {
       "ipv4:192.0.2.2": {
         "ipv4:192.0.2.89":   [ "ane:NET001", "ane:L002" ],
         "ipv4:203.0.113.45": [ "ane:NET001", "ane:L003" ]
       }
     }
   }
   --example-2
   Resource-Id: propmap
   Content-Type: application/alto-propmap+json

   {
     "meta": {
       "dependent-vtags": [
         {
           "resource-id": "endpoint-cost-pv.ecs",
           "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
         },
         {
           "resource-id": "http-proxy-props",
           "tag": "bf3c8c1819d2421c9a95a9d02af557a3"
         }
       ]
     },
     "property-map": {
       "ane:NET001": {
         "persistent-entities": [ "http-proxy:192.0.2.1" ]
       },
       "ane:L002": { "maxresbw": 48000000 },
       "ane:L003": { "maxresbw": 35000000 }
     }
   }
```

7.4. Example: Incremental Updates

In this example, an ALTO client subscribes to the incremental update for the multipart endpoint cost resource "endpoint-cost-pv".

```
POST /updates/pv HTTP/1.1
Host: alto.example.com
Accept: text/event-stream
Content-Type: application/alto-updatestreamparams+json
Content-Length: [TBD]

{
  "add": {
    "ecspvsub1": {
      "resource-id": "endpoint-cost-pv",
      "input": <ecs-input>
    }
  }
}
```

Based on the server-side process defined in [I-D.ietf-alto-incr-update-sse], the ALTO server will send the "control-uri" first using Server-Sent Event (SSE), followed by the full response of the multipart message.

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: application/alto-updatestreamcontrol+json
data: {"control-uri": "http://alto.example.com/updates/streams/1414"}

event: multipart/related;boundary=example-3;
       type=application/alto-endpointcost+json,ecspvsub1
data: --example-3
data: Resource-ID: ecsmap
data: Content-Type: application/alto-endpointcost+json
data:
data: <endpoint-cost-map-entry>
data: --example-3
data: Resource-ID: propmap
data: Content-Type: application/alto-propmap+json
data:
data: <property-map-entry>
data: --example-3--
```

When the contents change, the ALTO server will publish the updates for each node in this tree separately.

```
event: application/merge-patch+json, ecspvsub1.ecsmap
data: <Merge patch for endpoint-cost-map-update>

event: application/merge-patch+json, ecspvsub1.propmap
data: <Merge patch for property-map-update>
```

8. Compatibility

8.1. Compatibility with Legacy ALTO Clients/Servers

The multipart filtered cost map resource and the multipart endpoint cost resource has no backward compatibility issue with legacy ALTO clients and servers. Although these two types of resources reuse the media types defined in the base ALTO protocol for the accept input parameters, they have different media types for responses. If the ALTO server provides these two types of resources, but the ALTO client does not support them, the ALTO client will ignore the resources without conducting any incompatibility.

8.2. Compatibility with Multi-Cost Extension

This document does not specify how to integrate the "path-vector" cost mode with the multi-cost extension [RFC8189]. Although there is no reason why somebody has to compound the path vectors with other cost types in a single query, there is no compatible issue doing it without constraint tests.

8.3. Compatibility with Incremental Update

The extension specified in this document is NOT compatible with the original incremental update extension [I-D.ietf-alto-incr-update-sse]. A legacy ALTO client CANNOT recognize the compound client-id, and a legacy ALTO server MAY use the same client-id for updates of both parts.

ALTO clients and servers MUST follow the specifications given in this document to ensure compatibility with the incremental update extension.

8.4. Compatibility with Cost Calendar

The extension specified in this document is compatible with the Cost Calendar extension [I-D.ietf-alto-cost-calendar]. When used together with the Cost Calendar extension, the cost value between a source and a destination is an array of path vectors, where the k-th path vector refers to the abstract network paths traversed in the k-th time interval by traffic from the source to the destination.

When used with time-varying properties, e.g., maximum reservable bandwidth (maxresbw), a property of a single entity may also have different values in different time intervals. In this case, an ANE with different property values MUST be considered as different ANEs.

The two extensions combined together CAN provide the historical network correlation information for a set of source and destination pairs. A network broker or client MAY use this information to derive other resource requirements such as Time-Block-Maximum Bandwidth, Bandwidth-Sliding-Window, and Time-Bandwidth-Product (TBP) (See [SENSE] for details.)

9. General Discussions

9.1. Provide Calendar for Property Map

Fetching the historical network information is useful for many traffic optimization problem. [I-D.ietf-alto-cost-calendar] already proposes an ALTO extension called Cost Calendar which provides the historical cost values using filtered cost map and endpoint cost service. However, the calendar for only path costs is not enough.

For example, as the properties of ANEs (e.g., available bandwidth and link delay) are usually the real-time network states, they change frequently in the real network. It is very helpful to get the historical value of these properties. Applications may predicate the network status using these information to better optimize their performance.

So the coming requirement may be a general calendar service for the ALTO information resources.

9.2. Constraint Tests for General Cost Types

The constraint test is a simple approach to query the data. It allows users to filter the query result by specifying some boolean tests. This approach is already used in the ALTO protocol. [RFC7285] and [RFC8189] allow ALTO clients to specify the "constraints" and "or-constraints" tests to better filter the result.

However, the current defined syntax is too simple and can only be used to test the scalar cost value. For more complex cost types, like the "array" mode defined in this document, it does not work well. It will be helpful to propose more general constraint tests to better perform the query.

In practice, it is too complex to customize a language for the general-purpose boolean tests, and can be a duplicated work. So it

may be a good idea to integrate some already defined and widely used query languages (or their subset) to solve this problem. The candidates can be XQuery and JSONiq.

9.3. General Multipart Resources Query

Querying multiple ALTO information resources continuously MAY be a general requirement. And the coming issues like inefficiency and inconsistency are also general. There is no standard solving these issues yet. So we need some approach to make the ALTO client request the compound ALTO information resources in a single query.

10. Security Considerations

This document is an extension of the base ALTO protocol, so the Security Considerations [RFC7285] of the base ALTO protocol fully apply when this extension is provided by an ALTO server.

The path vector extension requires additional considerations on two security considerations discussed in the base protocol: confidentiality of ALTO information (Section 15.3 of [RFC7285]) and availability of ALTO service (Section 15.5 of [RFC7285]).

For confidentiality of ALTO information, a network operator should be aware of that this extension may introduce a new risk: the path vector information may make network attacks easier. For example, as the path vector information may reveal more network internal structures than the base protocol, an ALTO client may detect the bottleneck link and start a distributed denial-of-service (DDoS) attack involving minimal flows to conduct the in-network congestion.

To mitigate this risk, the ALTO server should consider protection mechanisms to reduce information exposure or obfuscate the real information, in particular, in settings where the network and the application do not belong to the same trust domain. But the implementation of path vector extension involving reduction or obfuscation should guarantees the constraints on the requested properties are still accurate.

For availability of ALTO service, an ALTO server should be cognizant that using path vector extension might have a new risk: frequent requesting for path vectors might conduct intolerable increment of the server-side storage and break the ALTO server. It is known that the computation of path vectors is unlikely to be cacheable, in that the results will depend on the particular requests (e.g., where the flows are distributed). Hence, the service providing path vectors may become an entry point for denial-of-service attacks on the

availability of an ALTO server. To avoid this risk, authenticity and authorization of this ALTO service may need to be better protected.

11. IANA Considerations

11.1. ALTO Cost Mode Registry

This document specifies a new cost mode "path-vector". However, the base ALTO protocol does not have a Cost Mode Registry where new cost mode can be registered. This new cost mode will be registered once the registry is defined either in a revised version of [RFC7285] or in another future extension.

11.2. ALTO Entity Domain Registry

As proposed in Section 9.2 of [I-D.ietf-alto-unified-props-new], "ALTO Domain Entity Registry" is requested. Besides, a new domain is to be registered, listed in Table 2.

Identifier	Entity Address Encoding	Hierarchy & Inheritance
ane	See Section 5.3.2	None

Table 2: ALTO Entity Domain

11.3. ALTO Entity Property Type Registry

The "ALTO Entity Property Type Registry" is required by the ALTO Domain "ane", listed in Table 3.

Identifier	Intended Semantics
ane:maxresbw	The maximum reservable bandwidth for the ANE
ane:persistent-entities	An array of identifiers of persistent entities that reside in an ANE

Table 3: ALTO Entity Property Types

11.4. ALTO Resource Entity Domain Export Registries

11.4.1. costmap

Entity Domain Type	Export Function
ane	See Section 5.4.1

Table 4: ALTO Cost Map Entity Domain Export.

11.4.2. endpointcost

Entity Domain Type	Export Function
ane	See Section 5.4.2

Table 5: ALTO Endpoint Cost Entity Domain Export.

12. Acknowledgments

The authors would like to thank discussions with Andreas Voellmy, Erran Li, Haibin Song, Haizhou Du, Jiayuan Hu, Qiao Xiang, Tianyuan Liu, Xiao Shi, Xin Wang, and Yan Luo. The authors thank Greg Bernstein (Grotto Networks), Dawn Chen (Tongji University), Wendy Roome, and Michael Scharf for their contributions to earlier drafts.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, DOI 10.17487/RFC2387, August 1998, <https://www.rfc-editor.org/info/rfc2387>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <https://www.rfc-editor.org/info/rfc7285>.

[RFC8189] Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost Application-Layer Traffic Optimization (ALTO)", RFC 8189, DOI 10.17487/RFC8189, October 2017, <https://www.rfc-editor.org/info/rfc8189>.

13.2. Informative References

[I-D.bernstein-alto-topo]
Bernstein, G., Yang, Y., and Y. Lee, "ALTO Topology Service: Uses Cases, Requirements, and Framework", draft-bernstein-alto-topo-00 (work in progress), October 2013.

[I-D.ietf-alto-cost-calendar]
Randriamasy, S., Yang, Y., Wu, Q., Lingli, D., and N. Schwan, "ALTO Cost Calendar", draft-ietf-alto-cost-calendar-01 (work in progress), February 2017.

[I-D.ietf-alto-incr-update-sse]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", draft-ietf-alto-incr-update-sse-16 (work in progress), March 2019.

[I-D.ietf-alto-performance-metrics]
Wu, Q., Yang, Y., Lee, Y., Dhody, D., and S. Randriamasy, "ALTO Performance Cost Metrics", draft-ietf-alto-performance-metrics-06 (work in progress), November 2018.

[I-D.ietf-alto-unified-props-new]
Roome, W., Randriamasy, S., Yang, Y., and J. Zhang, "Unified Properties for the ALTO Protocol", draft-ietf-alto-unified-props-new-07 (work in progress), March 2019.

[SENSE] "Services - SENSE", 2019, <http://sense.es.net/services>.

[TON2019] Gao, K., Xiang, Q., Wang, X., Yang, Y., and J. Bi, "An objective-driven on-demand network abstraction for adaptive applications", IEEE/ACM Transactions on Networking (TON) 27, no. 2 (2019): 805-818., 2019.

Authors' Addresses

Kai Gao
Sichuan University
Chengdu 610000
China

Email: kaigao@scu.edu.cn

Young Lee
Huawei
TX
USA

Email: leeyoung@huawei.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
NOZAY 91460
FRANCE

Email: Sabine.Randriamasy@nokia-bell-labs.com

Y. Richard Yang
Yale University
51 Prospect St
New Haven CT
USA

Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: March 7, 2020

W. Roome
S. Randriamasy
Nokia Bell Labs
Y. Yang
Yale University
J. Zhang
Tongji University
K. Gao
Sichuan University
September 4, 2019

Unified Properties for the ALTO Protocol
draft-ietf-alto-unified-props-new-09

Abstract

This document extends the Application-Layer Traffic Optimization (ALTO) Protocol [RFC7285] by generalizing the concept of "endpoint properties" to generic types of entities, and by presenting those properties as maps, similar to the network and cost maps in [RFC7285].

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 7, 2020.

Table of Contents

1. Introduction

The ALTO protocol [RFC7285] introduces the concept of "properties" attached to "endpoint addresses", and defines the Endpoint Property Service (EPS) to allow ALTO clients to retrieve those properties. While useful, the EPS, as defined in [RFC7285], has at least three limitations.

First, the EPS allows properties to be associated with only endpoints which are identified by individual communication addresses like IPv4 and IPv6 addresses. It is reasonable to think that collections of endpoints, as defined by CIDRs [RFC4632] or PIDs, may also have properties. Furthermore, recent ALTO use cases show that properties of network flows [RFC7011] and routing elements [RFC7921] are also very useful. Since the EPS cannot be extended to those generic entities, new services, with new request and response messages, would have to be defined for them.

Second, the EPS only allows endpoints identified by global communication addresses. However, many other generic entities like PIDs may not have global identifiers. Even for Internet addresses, there may be some local IP addresses and anycast IP addresses which are also not global unique.

Third, the EPS is only defined as a POST-mode service. Clients must request the properties for an explicit set of endpoint addresses. By contrast, [RFC7285] defines a GET-mode cost map resource which returns all available costs, so a client can get a full set of costs once, and then processes costs lookups without querying the ALTO server. [RFC7285] does not define a similar service for endpoint properties. At first a map of endpoint properties might seem impractical, because it could require enumerating the property value for every possible endpoint. But in practice, it is highly unlikely that properties will be defined for every endpoint address. It is much more likely that properties may be defined for only a subset of endpoint addresses, and the specification of properties uses an aggregation representation to allow enumeration. This is particularly true if blocks of endpoint addresses with a common prefix (e.g., a CIDR) have the same value for a property. Entities

in other domains may very well allow aggregated representation and hence be enumerable as well.

This document specifies a new approach for defining and retrieving ALTO properties to address the three limitations:

- This document addresses the first limitation by introducing a generic concept called ALTO Entity which is a generalization of an endpoint to represent a PID, a network element, a cell in a cellular network, or other physical or logical objects used by ALTO. Each entity is included by a collection called ALTO Entity Domain. And each entity domain includes only one type of entities. Thus, each entity domain also has a type to indicate the type of entities in it.

- Additionally, this document addresses the second limitation by using resource-specific entity domains. A resource-specific entity domain is an entity domain exported by an existing ALTO information resource. And a resource-specific entity domain is named by its type and the resource id of the ALTO information resource which exports it. As each resource-specific entity domain name is unique, an entity can be uniquely identified by the name of a resource-specific entity domain and its domain-specific identifier.

- Finally, this document addresses the third limitation by defining two new types of ALTO information resources, namely Property Map (see Section 6) and Filtered Property Map (see Section 7). The former is a GET-mode resource which returns the property values for all entities in some entity domains, and is analogous to a network map or a cost map in [RFC7285]. The latter is a POST-mode resource which returns the values for a set of properties and entities requested by the client, and is analogous to a filtered network map or a filtered cost map.

This approach is extensible, because new entity domain types can be defined without revising the protocol specification defined in this document, in the same way that new cost metrics and new endpoint properties can be defined without revising the protocol specification defined in [RFC7285].

This document subsumes the Endpoint Property Service defined in [RFC7285], although that service may be retained for legacy clients (see Section 8).

2. Overview: Basic Concepts

Before we define the specification of unified properties, there are several basic concepts which we need to introduce.

2.1. Entity

The entity concept generalizes the concept of the endpoint defined in Section 2.1 of [RFC7285]. An entity is an object that can be an endpoint and is identified by its network address, but can also be an object that has a defined mapping to a set of one or more network addresses or is even not related to any network address.

Examples of eligible entities are:

- a PID, defined in [RFC7285], that has a provider defined human readable abstract identifier defined by a ALTO network map, which maps a PID to a set of ipv4 and ipv6 addresses;
- an autonomous system (AS), that has an AS number (ASN) as its identifier and maps to a set of ipv4 and ipv6 addresses;
- a region representing a country, that is identified by its country code defined by ISO 3166 and maps to a set of cellular addresses;
- a TCP/IP network flow, that has a server defined identifier consisting of the defining TCP/IP 5-Tuple, , which is an example that all endpoints are entities while not all entities are endpoints;
- a routing element, that is specified in [RFC7921] and includes routing capability information;
- an abstract network element, that has a server defined identifier and represents a network node, link or their aggregation.

2.2. Entity Property

An entity property defines a property of an entity. It is similar to the endpoint property defined by Section 7.1 of [RFC7285], but can be general besides network-aware.

For example,

- an "ipv4" entity may have a property whose value is an Autonomous System (AS) number indicating the AS which this IPv4 address is owned by;

- a "pid" entity may have a property which indicates the central geographical location of endpoints included by it.

2.3. Property Map

An ALTO property map provides a set of properties for a set of entities. These entities may be in different types. For example, an ALTO property map may define the ASN property for both "ipv4" and "ipv6" entities.

2.4. Information Resource

This document uses the same definition of the information resource as defined by [RFC7285]. Each information resource usually has a JSON format representation following a specific schema defined by its media type.

For example, an ALTO network map resource is represented by a JSON objectof type InfoResourceNetworkMap defined by the media type "application/alto-networkmap+json".

2.5. Entity Domain

An entity domain defines a set of entities in the same type. This type is also called the type of this entity domain.

Using entity domains, an ALTO property map can indicate which entities the ALTO client can query to get their properties.

2.5.1. Resource-Specific Entity Domain

To define an entity domain, one naive solution is to enumerate all entities in this entity domain. But it is inefficient when the size of the entity domain is large.

To avoid enumerating all entities, this document introduces an approach called "Resource-Specific Entity Domain" to define entity domains:

Each information resource may define several types of entity domains. And for each type of entity domain, an information resource can define at most one entity domain. For example, an ALTO netowrk map resource can define an IPv4 domain, an IPv6 domain and a pid domain. In this document, these entity domains are called resource-specific entity domains. An ALTO property map only need to indicate which types of entity domain defined by which information resources can be queried, the ALTO client will know which entities are effective to be queried.

2.5.2. Relationship between Entity and Entity Domain

In this document, an entity is owned by exact one entity domain. It requires that when an ALTO client or server references an entity, it must indicate its entity domain explicitly. Even two entities in two different entity domains may reflect to the same physical or logical object, we treat them as different entities.

Because of this rule, although the resource-specific entity domain approach has no ambiguity, it may introduce redundancy.

2.5.3. Aggregated Entity Domain

Two entities in two different resource-specific entity domains may reflect to the same physical or logical object. For example, the IPv4 entity "192.0.2.34" in the IPv4 domain of the network map "netmap1" and the IPv4 entity "192.0.2.34" in the IPv4 domain of the network map "netmap2" should indicate the same Internet endpoint addressed by the IPv4 address "192.0.2.34".

Each entity in each resource-specific entity domain may only have part of properties of its associated physical or logical object. For example, the IPv4 entity in the IPv4 domain of the network map "netmap1" only has the PID property defined by "netmap1"; same to the IPv4 entity in the IPv4 domain of the network map "netmap2". If the ALTO client wants to get the complete properties, using the resource-specific entity domain, the ALTO client has to query the IPv4 entity "192.0.2.34" twice.

To simplify the query process of the ALTO client, this document introduces the concept "Aggregated Entity Domain". An aggregated entity domain defines a union set of entities coming from multiple resource-specific entity domains in the same type. An entity in the aggregated entity domain inherits all properties defined for its associated entity in each associated resource-specific entity domains. For example, the IPv4 entity "192.0.2.34" in the aggregated entity domain between the IPv4 domain of "netmap1" and the IPv4 domain of "netmap2" has PID properties defined by both "netmap1" and "netmap2".

Note that some resource-specific entity domains may not be able to be aggregated even if they are in the same type. For example, a property map "propmap1" may define the "asn" property on both PID domains "netmap1.pid" and "netmap2.pid". But the PID "pid1" in "netmap1.pid" and the PID with the same name in "netmap2.pid" have different "asn" property values. It does not make sense to define an aggregated PID domain between "netmap1.pid" and "netmap2.pid" to provide the "propmap1.asn" property because it is ambiguous.

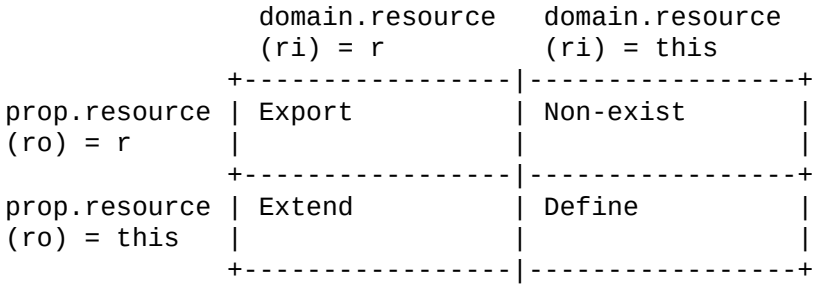
2.5.4. Resource-Specific Entity Property

According to the example of the aggregated entity domain, an entity may have multiple properties in the same type but associated to different information resources. To distinguish them, this document uses the same approach proposed by Section 10.8.1 of [RFC7285], which is called "Resource-Specific Entity Property".

2.6. Scope of Property Map

Using entity domains to organize entities, an ALTO property map resource actually provides a set of properties for some entity domains. If we ignore the syntax sugar of the aggregated entity domain, we can consider an ALTO property map resource just provides a set of (ri, di) => (ro, po) mappings, where (ri, di) means a resource-specific entity domain of type di defined by the information resource ri, and (ro, po) means a resource-specific entity property po defined by the information resource ro.

For each (ri, di) => (ro, po) mapping, the scope of an ALTO property map resource must be one of cases in the following diagram:



where "this" points to the resulting property map resource, "r" presents an existing ALTO information resource other the resulting property map resource.

- ri = ro = r ("export" mode): the property map resource just transforms the property mapping di => po defined by r into the unified representation format and exports it. For example: r = "netmap1", di = "ipv4", po = "pid". The property map resource exports the "ipv4 => pid" mapping defined by "netmap1".

- ri = r, ro = this ("extend" mode): the property map extends properties of entities in the entity domain (r, di) and defines a new property po on them. For example: the property map resource ("this") defines a "geolocation" property on domain "netmap1.pid".

- ri = ro = this ("define" mode): the property map defines a new intrinsic entity domain and defines property po for each entities in this domain. For example: the property map resource ("this") defines a new entity domain "asn" and defines a property "ipprefixes" on this domain.

- ri = this, ro = r: in the scope of a property map resource, it does not make sense that another existing ALTO information resource defines a property for this property map resource.

2.7. Entity Hierarchy and Property Inheritance

Enumerating all individual effective entities are inefficient. Some types of entities have the hierarchy format, e.g., cidr, which stand for sets of individual entities. Many entities in the same hierarchical format entity sets may have the same proprety values. To reduce the size of the property map representation, this document introduces an approach called "Property Inheritance". Individual entities can inherit the property from its hierarchical format entity set.

3. Protocol Specification: Basic Data Type

3.1. Entity Domain

3.1.1. Entity Domain Type

An entity domain has a type, which is defined by a string that MUST be no more than 64 characters, and MUST NOT contain characters other than US-ASCII alphanumeric characters (U+0030-U+0039, U+0041-U+005A, and U+0061-U+007A), hyphen ("-", U+002D), and low line ("_", U+005F). For example, the strings "ipv4", "ipv6", and "pid" are valid entity domain types.

The type EntityDomainType is used in this document to denote a JSON string confirming to the preceding requirement.

An entity domain type defines the semantics of a type of entity domains. Each entity domain type MUST be registered with the IANA. The format of the entity identifiers (see Section 3.1.3) in that type of entity domains, as well as any hierarchical or inheritance rules (see Section 3.1.4) for those entities, MUST be specified at the same time.

3.1.2. Entity Domain Name

Each entity domain is identified by an entity domain name, a string of the following format:

EntityDomainName ::= [[ResourceID] '.'] EntityDomainType

This document distinguish three types of entity domains: resource-specific entity domains, self-defined entity domain and aggregated entity domains. Their entity domain names are derived as follows.

Each ALTO information resource MAY define a resource-specific entity domain (which could be empty) in a given entity domain type. A resource-specific entity domain is identified by an entity domain name derived as follows. It MUST start with a resource ID using the ResourceID type defined in [RFC7285], followed by the "." separator (U+002E), followed by an EntityDomainType typed string. For example, if an ALTO server provides two network maps "netmap-1" and "netmap-2", they can define two different "pid" domains identified by "netmap-1.pid" and "netmap-2.pid" respectively. To be simplified, in the scope of a specific information resource, the resource-specific entity domain defined by itself can be identified by the "." EntityDomainTyep without the ResourceID.

When the associated information resource of a resource-specific entity domain is the current information resource itself, this resource-specific entity domain is a self-defined entity domain, and its ResourceID SHOULD be ignored from its entity domain name.

Given a set of ALTO information resources, there MAY be an aggregated entity domain in a given entity domain type amongst them. An aggregated entity domain is simply identified by its entity domain type. For example, given two network maps "net-map-1" and "net-map-2", "ipv4" and "ipv6" identify two aggregated Internet address entity domains (see Section 4.1) between them.

Note that the "." separator is not allowed in EntityDomainType and hence there is no ambiguity on whether an entity domain name refers to a global entity domain or a resource-specific entity domain.

3.1.3. Entity Identifier

Entities in an entity domain are identified by entity identifiers (EntityID) of the following format:

EntityID ::= EntityDomainName ':' DomainTypeSpecificEntityID

Examples from the Internet address entity domains include individual IP addresses such as "net1.ipv4:192.0.2.14" and "net1.ipv6:2001:db8::12", as well as address blocks such as "net1.ipv4:192.0.2.0/26" and "net1.ipv6:2001:db8::1/48".

The format of the second part of an entity identifier depends on the entity domain type, and MUST be specified when registering a new entity domain type. Identifiers MAY be hierarchical, and properties MAY be inherited based on that hierarchy. Again, the rules defining any hierarchy or inheritance MUST be defined when the entity domain type is registered.

The type EntityID is used in this document to denote a JSON string representing an entity identifier in this format.

Note that two entity identifiers with different textual representations may refer to the same entity, for a given entity domain. For example, the strings "net1.ipv6:2001:db8::1" and "net1.ipv6:2001:db8:0:0:0:0:0:1" refer to the same entity in the "ipv6" entity domain.

3.1.4. Hierarchy and Inheritance

To make the representation efficient, some types of entity domains MAY allow the ALTO client/server to use a hierarchical format entity identifier to represent a block of individual entities. e.g., In an IPv4 domain "net1.ipv4", a cidr "net1.ipv4:192.0.2.0/26" represents 64 individual IPv4 entities. In this case, the corresponding property inheritance rule MUST be defined for the entity domain type. The hierarchy and inheritance rule MUST have no ambiguity.

3.2. Entity Property

Each entity property has a type to indicate the encoding and the semantics of the value of this entity property, and has a name to be identified. One entity MAY have multiple properties in the same type.

3.2.1. Entity Property Type

The type EntityPropertyType is used in this document to indicate a string denoting an entity property type. The string MUST be no more than 32 characters, and it MUST NOT contain characters other than US-ASCII alphanumeric characters (U+0030-U+0039, U+0041-U+005A, and U+0061-U+007A), the hyphen ("-", U+002D), the colon (":", U+003A), or the low line ('_', U+005F).

Each entity property type MUST be registered with the IANA. The intended semantics of the entity property type MUST be specified at the same time.

To distinguish with the endpoint property type, the entity property type has the following features.

- Some entity property types may be applicable to entities in only particular types of entity domains, not all. For example, the "pid" property is not applicable to entities in a "pid" typed entity domain, but is applicable to entities in the "ipv4" or "ipv6" domains.

- The intended semantics of the value of a entity property may also depend on the the entity domain type of this entity. For example, suppose that the "geo-location" property is defined as the coordinates of a point, encoded as (say) "latitude longitude [altitude]." When applied to an entity that represents a specific host computer, identified by an address in the "ipv4" or "ipv6" entity domain, the property defines the host's location. However, when applied to an entity in a "pid" domain, the property would indicate the location of the center of all hosts in this "pid" entity.

3.2.2. Entity Property Name

Each entity property is identified by an entity property name, which is a string of the following format:

EntityPropertyName ::= [ResourceID] '.' EntityPropertyType

Similar to the endpoint property type defined in Section 10.8 of [RFC7285], each entity property may be defined by either the property map itself (self-defined) or some other specific information resource (resource-specific).

The entity property name of a resource-specific entity property starts with a string of the type ResourceID defined in [RFC7285], followed by the "." separator (U+002E) and a EntityDomainType typed string. For example, the "pid" properties of an "ipv4" entity defined by two different maps "net-map-1" and "net-map-2" are identified by "net-map-1.pid" and "net-map-2.pid" respectively.

When the associated information resource of the entity property is the current information resource itself, the ResourceID in the property name SHOULD be ignored. For example, the ".asn" property of an "ipv4" entity indicates the AS number of the AS which this IPv4 address is owned by.

3.3. Information Resource Export

Each information resource MAY export a set of entity domains and entity property mappings.

3.3.1. Resource-Specific Entity Domain Export

Each type of information resource MAY export several types of entity domains. For example, a network map resource defines a "pid" domain, a "ipv4" domain and a "ipv6" domain (which may be empty).

When a new ALTO information resource type is registered, if this type of information resource can export an existing type of entity domain, the corresponding document MUST define how to export such type of entity domain from such type of information resource.

When a new entity domain type is defined, if an existing type of information resource can export an entity domain in this entity domain type, the corresponding document MUST define how to export such type of entity domain from such type of information resource.

3.3.2. Entity Property Mapping Export

For each entity domain which could be exported by an information resource, this information resource MAY also export some mapping from this entity domain to some entity property. For example, a network map resource can map an "ipv4" entity to its "pid" property.

When a new ALTO information resource type is registered, if this type of information resource can export an entity domain in an existing entity domain type, and map entities in this entity domain to an existing type of entity property, the corresponding document MUST define how to export such type of an entity property.

When a new ALTO entity domain type or a new entity property type is defined, if an existing type of resource can export an entity domain in this entity domain type, and map entities in this entity domain to this type of entity property, the corresponding document MUST define how to export such type of an entity property.

4. Entity Domain Types

This document defines three entity domain types. The definition of each entity domain type below includes the following: (1) entity domain type name, (2) entity domain-specific entity identifiers, and (3) hierarchy and inheritance semantics. Since a global entity domain type defines a single global entity domain, we say entity domain instead of entity domain type.

4.1. Internet Address Domain Types

The document defines two entity domain types (IPv4 and IPv6) for Internet addresses. Both types are global entity domain types and hence define a corresponding global entity domain as well. Since the two domains use the same hierarchy and inheritance semantics, we define the semantics together, instead of repeating for each.

4.1.1. IPv4 Domain

4.1.1.1. Entity Domain Type

ipv4

4.1.1.2. Domain-Specific Entity Identifiers

Individual addresses are strings as specified by the IPv4Addresses rule of Section 3.2.2 of [RFC3986]; blocks of addresses are prefix-match strings as specified in Section 3.1 of [RFC4632]. For the purpose of defining properties, an individual Internet address and the corresponding full-length prefix are considered aliases for the same entity. Thus "ipv4:192.0.2.0" and "ipv4:192.0.2.0/32" are equivalent.

4.1.2. IPv6 Domain

4.1.2.1. Entity Domain Type

ipv6

4.1.2.2. Domain-Specific Entity Identifiers

Individual addresses are strings as specified by Section 4 of [RFC5952]; blocks of addresses are prefix-match strings as specified in Section 7 of [RFC5952]. For the purpose of defining properties, an individual Internet address and the corresponding 128-bit prefix are considered aliases for the same entity. That is, "ipv6:2001:db8::1" and "ipv6:2001:db8::1/128" are equivalent, and have the same set of properties.

4.1.3. Hierarchy and Inheritance of Internet Address Domains

Both Internet address domains allow property values to be inherited. Specifically, if a property P is not defined for a specific Internet address I, but P is defined for some block C which prefix-matches I, then the address I inherits the value of P defined for block C. If more than one such block defines a value for P, I inherits the value of P in the block with the longest prefix. It is important to notice

that this longest prefix rule will ensure no multiple inheritance, and hence no ambiguity.

Address blocks can also inherit properties: if a property P is not defined for a block C, but is defined for some block C' which covers all IP addresses in C, and C' has a shorter mask than C, then block C inherits the property from C'. If there are several such blocks C', C inherits from the block with the longest prefix.

As an example, suppose that a server defines a property P for the following entities:

```
ipv4:192.0.2.0/26: P=v1
ipv4:192.0.2.0/28: P=v2
ipv4:192.0.2.0/30: P=v3
ipv4:192.0.2.0:    P=v4
```

Figure 1: Defined Property Values.

Then the following entities have the indicated values:

```
ipv4:192.0.2.0:    P=v4
ipv4:192.0.2.1:    P=v3
ipv4:192.0.2.16:   P=v1
ipv4:192.0.2.32:   P=v1
ipv4:192.0.2.64:   (not defined)
ipv4:192.0.2.0/32: P=v4
ipv4:192.0.2.0/31: P=v3
ipv4:192.0.2.0/29: P=v2
ipv4:192.0.2.0/27: P=v1
ipv4:192.0.2.0/25: (not defined)
```

Figure 2: Inherited Property Values.

An ALTO server MAY explicitly indicate a property as not having a value for a particular entity. That is, a server MAY say that property P of entity X is "defined to have no value", instead of "undefined". To indicate "no value", a server MAY perform different behaviours:

- If that entity would inherit a value for that property, then the ALTO server MUST return a "null" value for that property. In this case, the ALTO client MUST recognize a "null" value as "no value" and "do not apply the inheritance rules for this property."

- If the entity would not inherit a value, then the ALTO server MAY return "null" or just omit the property. In this case, the ALTO client cannot infer the value for this property of this entity

from the Inheritance rules. So the client MUST interpret that this property has no value.

If the ALTO server does not define any properties for an entity, then the server MAY omit that entity from the response.

4.2. PID Domain

The PID domain associates property values with the PIDs in a network map. Accordingly, this entity domain always depends on a network map.

4.2.1. Entity Domain Type

pid

4.2.2. Domain-Specific Entity Identifiers

The entity identifiers are the PID names of the associated network map.

4.2.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with PIDs.

4.2.4. Relationship To Internet Addresses Domains

The PID domain and the Internet address domains are completely independent; the properties associated with a PID have no relation to the properties associated with the prefixes or endpoint addresses in that PID. An ALTO server MAY choose to assign some or all properties of a PID to the prefixes in that PID.

For example, suppose "PID1" consists of the prefix "ipv4:192.0.2.0/24", and has the property "P" with value "v1". The Internet address entities "ipv4:192.0.2.0" and "ipv4:192.0.2.0/24", in the IPv4 domain MAY have a value for the property "P", and if they do, it is not necessarily "v1".

4.3. Internet Address Properties vs. PID Properties

Because the Internet address and PID domains are completely separate, the question may arise as to which entity domain is the best for a property. In general, the Internet address domains are RECOMMENDED for properties that are closely related to the Internet address, or are associated with, and inherited through, blocks of addresses.

The PID domain is RECOMMENDED for properties that arise from the definition of the PID, rather than from the Internet address prefixes in that PID.

For example, because Internet addresses are allocated to service providers by blocks of prefixes, an "ISP" property would be best associated with the Internet address domain. On the other hand, a property that explains why a PID was formed, or how it relates a provider's network, would best be associated with the PID domain.

5. Entity Domains and Property Mappings in Information Resources

5.1. Network Map Resource

The ALTO network map resource defined by the media type "application/alto-networkmap+json" exports the following types of entity domains and entity property mappings.

5.1.1. Resource-Specific Entity Domain

An ALTO network map resource defines a "pid" domain, an "ipv4" domain and an "ipv6" domain by follows:

- The defined "pid" domain includes all PIDs in keys of the "network-map" object.
- The defined "ipv4" domain includes all IPv4 addresses appearing in the "ipv4" field of the endpoint address group of each PID.
- The defined "ipv6" domain includes all IPv6 addresses appearing in the "ipv6" field of the endpoint address group of each PID.

5.1.2. Entity Property Mapping

For each of the preceding entity domains, an ALTO network map resource provides the properties mapping as follows:

ipv4 -> pid: An "networkmap" typed resource can map an "ipv4" entity to a "pid" property whose value is a PID defined by this "networkmap" resource and including the IPv4 address of this entity.

ipv6 -> pid: An "networkmap" typed resource can map an "ipv6" entity to a "pid" property whose value is a PID defined by this "networkmap" resource and including the IPv6 address of this entity.

5.2. Endpoint Property Resource

The ALTO endpoint property resource defined by the media type "application/alto-endpointprop+json" exports the following types of entity domains and entity property mappings.

5.2.1. Resource-Specific Entity Domain

An ALTO endpoint property resource defined an "ipv4" domain and an "ipv6" domain by follows:

- The defined "ipv4" domain includes all IPv4 addresses appearing in keys of the "endpoint-properties" object.

- The defined "ipv6" domain includes all IPv6 addresses appearing in keys of the "endpoint-properties" object.

5.2.2. Entity Property Mapping

For each of the preceding entity domains, an ALTO endpoint property resource exports the properties mapping from it to each supported global endpoint property. The property value is the corresponding global endpoint property value in the "endpiont-properties" object.

5.3. Property Map Resource

To avoid the nested reference and its potential complexity, this document does not specify the export rule of resource-specific entity domain and entity property mapping for the ALTO property map resource defined by the media type "application/alto-propmap+json" (see Section 6.1).

6. Property Map

A property map returns the properties defined for all entities in one or more domains, e.g., the "location" property of entities in "pid" domain, and the "ASN" property of entities in "ipv4" and "ipv6" domains.

Section 9.4 gives an example of a property map request and its response.

6.1. Media Type

The media type of a property map is "application/alto-propmap+json".

6.2. HTTP Method

The property map is requested using the HTTP GET method.

6.3. Accept Input Parameters

None.

6.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities:

```
object {
  EntityPropertyMapping mappings;
} PropertyMapCapabilities;

object-map {
  EntityDomainName -> EntityPropertyName<1..*>;
} EntityPropertyMapping
```

with fields:

mappings: A JSON object whose keys are names of entity domains and values are the supported entity properties of the corresponding entity domains.

6.5. Uses

The "uses" field of a property map resource in an IRD entry specifies dependent resources of this property map. It is an array of the resource ID(s) of the resource(s).

6.6. Response

If the entity domains in this property map depend on other resources, the "dependent-vtags" field in the "meta" field of the response MUST be an array that includes the version tags of those resources, and the order MUST be consistent with the "uses" field of this property map resource. The data component of a property map response is named "property-map", which is a JSON object of type PropertyMapData, where:

```
object {
  PropertyMapData property-map;
} InfoResourceProperties : ResponseEntityBase;

object-map {
  EntityID -> EntityProps;
} PropertyMapData;

object {
  EntityPropertyName -> JSONValue;
} EntityProps;
```

The ResponseEntityBase type is defined in Section 8.4 of [RFC7285].

Specifically, a PropertyMapData object has one member for each entity in the property map. The entity's properties are encoded in the corresponding EntityProps object. EntityProps encodes one name/value pair for each property, where the property names are encoded as strings of type PropertyName. A protocol implementation SHOULD assume that the property value is either a JSONString or a JSON "null" value, and fail to parse if it is not, unless the implementation is using an extension to this document that indicates when and how property values of other data types are signaled.

For each entity in the property map:

- If the entity is in a resource-specific entity domain, the ALTO server SHOULD only return self-defined properties and resource-specific properties which depend on the same resource as the entity does. The ALTO client SHOULD ignore the resource-specific property in this entity if their mapping is not registered in the ALTO Resource Entity Property Transfer Registry of the type of the corresponding resource.

- If the entity is in a shared entity domain, the ALTO server SHOULD return self-defined properties and all resource-specific properties defined for all resource-specific entities which have the same domain-specific entity identifier as this entity does.

For efficiency, the ALTO server SHOULD omit property values that are inherited rather than explicitly defined; if a client needs inherited values, the client SHOULD use the entity domain's inheritance rules to deduce those values.

7. Filtered Property Map

A filtered property map returns the values of a set of properties for a set of entities selected by the client.

Section 9.5, Section 9.6, Section 9.7 and Section 9.8 give examples of filtered property map requests and responses.

7.1. Media Type

The media type of a property map resource is "application/alto-propmap+json".

7.2. HTTP Method

The filtered property map is requested using the HTTP POST method.

7.3. Accept Input Parameters

The input parameters for a filtered property map request are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-propmapparams+json", which is a JSON object of type ReqFilteredPropertyMap:

```
object {
  EntityID             entities<1..*>;
  EntityPropertyName   properties<1..*>;
} ReqFilteredPropertyMap;
```

with fields:

entities: List of entity identifiers for which the specified properties are to be returned. The ALTO server MUST interpret entries appearing multiple times as if they appeared only once. The domain of each entity MUST be included in the list of entity domains in this resource's "capabilities" field (see Section 7.4).

properties: List of properties to be returned for each entity. Each specified property MUST be included in the list of properties in this resource's "capabilities" field (see Section 7.4). The ALTO server MUST interpret entries appearing multiple times as if they appeared only once.

Note that the "entities" and "properties" fields MUST have at least one entry each.

7.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities, as defined in Section 6.4.

7.5. Uses

Same to the "uses" field of the Property Map resource (see Section 6.5).

7.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

Specifically, a filtered property map request can be invalid as follows:

- An entity identifier in "entities" in the request is invalid if:
  - The domain of this entity is not defined in the "entity-domains" capability of this resource in the IRD;
  - The entity identifier is an invalid identifier in the entity domain.

  A valid entity identifier is never an error, even if this filtered property map resource does not define any properties for it.

  If an entity identifier in "entities" in the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285], and the "value" field of the error message SHOULD indicate this entity identifier.

- A property name in "properties" in the request is invalid if this property name is not defined in the "properties" capability of this resource in the IRD.

  It is not an error that a filtered property map resource does not define a requested property's value for a particular entity. In this case, the ALTO server MUST omit that property from the response for that endpoint.

  If a property name in "properties" in the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285]. The "value" field of the error message SHOULD indicate the property name.

The response to a valid request is the same as for the Property Map (see Section 6.6), except that:

- If the requested entities include entities in the shared entity domain, the "dependent-vtags" field in its "meta" field MUST include version tags of all dependent resources appearing in the "uses" field.

- If the requested entities only include entities in resource-specific entity domains, the "dependent-vtags" field in its "meta" field MUST include version tags of resources which requested resource-specific entity domains and requested resource-specific properties are dependent on.

- The response only includes the entities and properties requested by the client. If an entity in the request is identified by a hierarchical identifier (e.g., an "ipv4" or "ipv6" address block), the response MUST cover properties for all identifiers in this hierarchical identifier.

It is important that the filtered property map response MUST include all inherited property values for the requested entities and all the entities which are able to inherit property values from them. To achieve this goal, the ALTO server MAY follow three rules:

- If a property for a requested entity is inherited from another entity not included in the request, the response SHOULD include this property for the requested entity. For example, A full property map may skip a property P for an entity A (e.g., ipv4:192.0.2.0/31) if P can be derived using inheritance from another entity B (e.g., ipv4:192.0.2.0/30). A filtered property map request may include only A but not B. In such a case, the property P SHOULD be included in the response for A.

- If there are entities covered by a requested entity but having different values for the requested properties, the response SHOULD include all those entities and the different property values for them. For example, considering a request for property P of entity A (e.g., ipv4:192.0.2.0/31), if P has value v1 for A1=ipv4:192.0.2.0/32 and v2 for A2=ipv4:192.0.2.1/32, then, the response SHOULD include A1 and A2.

- If an entity in the response is already covered by some other entities in the same response, it SHOULD be removed from the response for compactness. For example, in the previous example, the entity A=ipv4:192.0.2.0/31 SHOULD be removed because A1 and A2 cover all the addresses in A.

An ALTO client should be aware that the entities in the response MAY be different from the entities in its request.

8. Impact on Legacy ALTO Servers and ALTO Clients

8.1. Impact on Endpoint Property Service

Since the property map and the filtered property map defined in this document provide the functionality of the Endpoint Property Service (EPS) defined in Section 11.4 of [RFC7285], it is RECOMMENDED that the EPS be deprecated in favor of Property Map and Filtered Property Map. However, ALTO servers MAY provide an EPS for the benefit of legacy clients.

8.2. Impact on Resource-Specific Properties

Section 10.8 of [RFC7285] defines two categories of endpoint properties: "resource-specific" and "global". Resource-specific property names are prefixed with the ID of the resource they depend upon, while global property names have no such prefix. The property map and the filtered property map defined in this document defines the similar categories for entity properties. The difference is that there is no "global" entity properties but the "self-defined" entity properties as the special case of the "resource-specific" entity properties instead.

8.3. Impact on Other Properties

In general, there should be little or no impact on other previously defined properties. The only consideration is that properties can now be defined on blocks of entity identifiers, rather than just individual entity identifiers, which might change the semantics of a property.

9. Examples

9.1. Network Map

The examples in this section use a very simple default network map:

```
defaultpid:  ipv4:0.0.0.0/0  ipv6:::0/0
pid1:        ipv4:192.0.2.0/25
pid2:        ipv4:192.0.2.0/28  ipv4:192.0.2.16/28
pid3:        ipv4:192.0.3.0/28
pid4:        ipv4:192.0.3.16/28
```

Figure 3: Example Default Network Map

And another simple alternative network map:

```
defaultpid:  ipv4:0.0.0.0/0  ipv6:::0/0
pid1:        ipv4:192.0.2.0/28  ipv4:192.0.2.16/28
pid2:        ipv4:192.0.3.0/28  ipv4:192.0.3.16/28
```

Figure 4: Example Alternative Network Map

9.2. Property Definitions

Beyond "pid", the examples in this section use four additional properties for Internet address domains, "ISP", "ASN", "country" and "state", with the following values:

	ISP	ASN	country	state
ipv4:192.0.2.0/23:	BitsRus	-	us	-
ipv4:192.0.2.0/28:	-	12345	-	NJ
ipv4:192.0.2.16/28:	-	12345	-	CT
ipv4:192.0.2.0:	-	-	-	PA
ipv4:192.0.3.0/28:	-	12346	-	TX
ipv4:192.0.3.16/28:	-	12346	-	MN

Figure 5: Example Property Values for Internet Address Domains

And the examples in this section use the property "region" for the PID domain of the default network map with the following values:

	region
pid:defaultpid:	-
pid:pid1:	us-west
pid:pid2:	us-east
pid:pid3:	us-south
pid:pid4:	us-north

Figure 6: Example Property Values for Default Network Map’s PID Domain

Note that "-" means the value of the property for the entity is "undefined". So the entity would inherit a value for this property by the inheritance rule if possible. For example, the value of the "ISP" property for "ipv4:192.0.2.0" is "BitsRus" because of "ipv4:192.0.2.0/24". But the "region" property for "pid:defaultpid" has no value because no entity from which it can inherit.

Similar to the PID domain of the default network map, the examples in this section use the property "ASN" for the PID domain of the alternative network map with the following values:

	ASN
pid:defaultpid:	-
pid:pid1:	12345
pid:pid2:	12346

Figure 7: Example Property Values for Alternative Network Map's PID Domain

9.3. Information Resource Directory (IRD)

The following IRD defines the relevant resources of the ALTO server. It provides two property maps, one for the "ISP" and "ASN" properties, and another for the "country" and "state" properties. The server could have provided a single property map for all four properties, but did not, presumably because the organization that runs the ALTO server believes any given client is not interested in all four properties.

The server provides two filtered property maps. The first returns all four properties, and the second just returns the "pid" property for the default network map.

The filtered property maps for the "ISP", "ASN", "country" and "state" properties do not depend on the default network map (it does not have a "uses" capability), because the definitions of those properties do not depend on the default network map. The Filtered Property Map for the "pid" property does have a "uses" capability for the default network map, because that defines the values of the "pid" property.

Note that for legacy clients, the ALTO server provides an Endpoint Property Service for the "pid" property for the default network map.

```
"meta" : {
  ...
  "default-alto-network-map" : "default-network-map"
},
"resources" : {
  "default-network-map" : {
    "uri" : "http://alto.example.com/networkmap/default",
    "media-type" : "application/alto-networkmap+json"
  },
  "alt-network-map" : {
    "uri" : "http://alto.example.com/networkmap/alt",
    "media-type" : "application/alto-networkmap+json"
  },
  .... property map resources ....
  "ia-property-map" : {
```

```
      "uri" : "http://alto.example.com/propmap/full/inet-ia",
      "media-type" : "application/alto-propmap+json",
      "uses": [ "default-network-map", "alt-network-map" ],
      "capabilities" : {
        "mappings": {
          "ipv4": [ ".ISP", ".ASN" ],
          "ipv6": [ ".ISP", ".ASN" ]
        }
      }
    },
    "iacs-property-map" : {
      "uri" : "http://alto.example.com/propmap/full/inet-iacs",
      "media-type" : "application/alto-propmap+json",
      "accepts": "application/alto-propmapparams+json",
      "uses": [ "default-network-map", "alt-network-map" ],
      "capabilities" : {
        "mappings": {
          "ipv4": [ ".ISP", ".ASN", ".country", ".state" ],
          "ipv6": [ ".ISP", ".ASN", ".country", ".state" ]
        }
      }
    },
    "region-property-map": {
      "uri": "http://alto.exmaple.com/propmap/region",
      "media-type": "application/alto-propmap+json",
      "accepts": "application/alto-propmapparams+json",
      "uses" : [ "default-network-map", "alt-network-map" ],
      "capabilities": {
        "mappings": {
          "default-network-map.pid": [ ".region" ],
          "alt-network-map.pid": [ ".ASN" ],
        }
      }
    },
    "ip-pid-property-map" : {
      "uri" : "http://alto.example.com/propmap/lookup/pid",
      "media-type" : "application/alto-propmap+json",
      "accepts" : "application/alto-propmapparams+json",
      "uses" : [ "default-network-map", "alt-network-map" ],
      "capabilities" : {
        "mappings": {
          "ipv4": [ "default-network-map.pid",
                    "alt-network-map.pid" ],
          "ipv6": [ "default-network-map.pid",
                    "alt-network-map.pid" ]
        }
      }
    },
```

```
        "legacy-endpoint-property" : {
           "uri" : "http://alto.example.com/legacy/eps-pid",
           "media-type" : "application/alto-endpointprop+json",
           "accepts" : "application/alto-endpointpropparams+json",
           "capabilities" : {
             "properties" : [ "default-network-map.pid",
                              "alt-network-map.pid" ]
           }
        }
      }
```

Figure 8: Example IRD

9.4. Property Map Example

The following example uses the properties and IRD defined above to retrieve a Property Map for entities with the "ISP" and "ASN" properties.

Note that, to be compact, the response does not includes the entity "ipv4:192.0.2.0", because values of all those properties for this entity are inherited from other entities.

Also note that the entities "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28" are merged into "ipv4:192.0.2.0/27", because they have the same value of the "ASN" property. The same rule applies to the entities "ipv4:192.0.3.0/28" and "ipv4:192.0.3.0/28". Both of "ipv4:192.0.2.0/27" and "ipv4:192.0.3.0/27" omit the value for the "ISP" property, because it is inherited from "ipv4:192.0.2.0/23".

```
GET /propmap/full/inet-ia HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "property-map": {
    "ipv4:192.0.2.0/23":   {".ISP": "BitsRus"},
    "ipv4:192.0.2.0/27":   {".ASN": "12345"},
    "ipv4:192.0.3.0/27":   {".ASN": "12346"}
  }
}
```

9.5. Filtered Property Map Example #1

The following example uses the filtered property map resource to request the "ISP", "ASN" and "state" properties for several IPv4 addresses.

Note that the value of "state" for "ipv4:192.0.2.0" is the only explicitly defined property; the other values are all derived by the inheritance rules for Internet address entities.

```
POST /propmap/lookup/inet-iacs HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
  "entities" : [ "ipv4:192.0.2.0",
                 "ipv4:192.0.2.1",
                 "ipv4:192.0.2.17" ],
  "properties" : [ ".ISP", ".ASN", ".state" ]
}
```

```
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "property-map": {
    "ipv4:192.0.2.0":
           {".ISP": "BitsRus", ".ASN": "12345", ".state": "PA"},
    "ipv4:192.0.2.1":
           {".ISP": "BitsRus", ".ASN": "12345", ".state": "NJ"},
    "ipv4:192.0.2.17":
           {".ISP": "BitsRus", ".ASN": "12345", ".state": "CT"}
  }
}
```

9.6. Filtered Property Map Example #2

The following example uses the filtered property map resource to request the "ASN", "country" and "state" properties for several IPv4 prefixes.

Note that the property values for both entities "ipv4:192.0.2.0/26" and "ipv4:192.0.3.0/26" are not explicitly defined. They are inherited from the entity "ipv4:192.0.2.0/23".

Also note that some entities like "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28" in the response are not listed in the request explicitly. The response includes them because they are refinements of the requested entities and have different values for the requested properties.

The entity "ipv4:192.0.4.0/26" is not included in the response, because there are neither entities which it is inherited from, nor entities inherited from it.

```
POST /propmap/lookup/inet-iacs HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
  "entities" : [ "ipv4:192.0.2.0/26",
                 "ipv4:192.0.3.0/26",
                 "ipv4:192.0.4.0/26" ],
  "properties" : [ ".ASN", ".country", ".state" ]
}

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "property-map": {
    "ipv4:192.0.2.0/26":  {".country": "us"},
    "ipv4:192.0.2.0/28":  {".ASN": "12345",
                           ".state": "NJ"},
    "ipv4:192.0.2.16/28": {".ASN": "12345",
                           ".state": "CT"},
    "ipv4:192.0.2.0":     {".state": "PA"},
    "ipv4:192.0.3.0/26":  {".country": "us"},
    "ipv4:192.0.3.0/28":  {".ASN": "12345",
                           ".state": "TX"},
    "ipv4:192.0.3.16/28": {".ASN": "12345",
                           ".state": "MN"}
 }
}
```

9.7. Filtered Property Map Example #3

The following example uses the filtered property map resource to request the "pid" property for several IPv4 addresses and prefixes.

Note that the entity "ipv4:192.0.3.0/27" is redundant in the response. Although it can inherit a value of "defaultpid" for the

"pid" property from the entity "ipv4:0.0.0.0/0", none of addresses in it is in "defaultpid". Because blocks "ipv4:192.0.3.0/28" and "ipv4:192.0.3.16/28" have already cover all addresses in that block. So an ALTO server who wants a compact response can omit this entity.

```
POST /propmap/lookup/pid HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
  "entities" : [
                "ipv4:192.0.2.128",
                "ipv4:192.0.3.0/27" ],
  "properties" : [ "default-network-map.pid" ]
}

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "property-map": {
    "ipv4:192.0.2.128":   {"default-network-map.pid": "defaultpid"},
    "ipv4:192.0.2.0/27":  {"default-network-map.pid": "defaultpid"},
    "ipv4:192.0.3.0/28":  {"default-network-map.pid": "pid3"},
    "ipv4:192.0.3.16/28": {"default-network-map.pid": "pid4"}
  }
}
```

9.8. Filtered Property Map Example #4

The following example uses the filtered property map resource to request the "region" property for several PIDs defined in "default-network-map". The value of the "region" property for each PID is not defined by "default-network-map", but the reason why the PID is defined by the network operator.

```
POST /propmap/lookup/region HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
  "entities" : ["default-network-map.pid:pid1",
                "default-network-map.pid:pid2"],
  "properties" : [ ".region" ]
}

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "meta" : {
    "dependent-vtags" : [
       {"resource-id": "default-network-map",
        "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
    ]
  },
  "property-map": {
    "default-network-map.pid:pid1": {
      ".region": "us-west"
    },
    "default-network-map.pid:pid2": {
      ".region": "us-east"
    }
  }
}
```

10. Security Considerations

Both Property Map and Filtered Property Map defined in this document fit into the architecture of the ALTO base protocol, and hence the Security Considerations (Section 15 of [RFC7285]) of the base protocol fully apply: authenticity and integrity of ALTO information (i.e., authenticity and integrity of Property Maps), potential undesirable guidance from authenticated ALTO information (e.g., potentially imprecise or even wrong value of a property such as geo-location), confidentiality of ALTO information (e.g., exposure of a potentially sensitive entity property such as geo-location), privacy for ALTO users, and availability of ALTO services should all be considered.

A particular fundamental security consideration when an ALTO server provides a Property Map is to define precisely the policies on who can access what properties for which entities. Security mechanisms such as authentication and confidentiality mechanisms then should be applied to enforce the policy. For example, a policy can be that a property P can be accessed only by its owner (e.g., the customer who is allocated a given IP address). Then, the ALTO server will need to deploy corresponding mechanisms to realize the policy. The policy may allow non-owners to access a coarse-grained value of the property P. In such a case, the ALTO server may provide a different URI to provide the information.

11. IANA Considerations

This document defines additional application/alto-* media types, and extends the ALTO endpoint property registry.

11.1. application/alto-* Media Types

This document registers two additional ALTO media types, listed in Table 1.

Type	Subtype	Specification
application	alto-propmap+json	Section 6.1
application	alto-propmapparams+json	Section 7.3

Table 1: Additional ALTO Media Types.

Type name: application

Subtype name: This document registers multiple subtypes, as listed in Table 1.

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations: Security considerations related to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285].

Interoperability considerations: This document specifies formats of conforming messages and the interpretation thereof.

Published specification: This document is the specification for these media types; see Table 1 for the section documenting each media type.

Applications that use this media type: ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Additional information:

Magic number(s): n/a

File extension(s): This document uses the mime type to refer to protocol messages and thus does not require a file extension.

Macintosh file type code(s): n/a

Person & email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors' Addresses section.

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org).

11.2. ALTO Entity Domain Type Registry

This document requests IANA to create and maintain the "ALTO Entity Domain Type Registry", listed in Table 2.

Identifier	Entity Identifier Encoding	Hierarchy & Inheritance
ipv4	See Section 4.1.1	See Section 4.1.3
ipv6	See Section 4.1.2	See Section 4.1.3
pid	See Section 4.2	None

Table 2: ALTO Entity Domains.

This registry serves two purposes. First, it ensures uniqueness of identifiers referring to ALTO entity domains. Second, it states the requirements for allocated entity domains.

11.2.1. Consistency Procedure between ALTO Address Type Registry and ALTO Entity Domain Type Registry

One potential issue of introducing the "ALTO Entity Domain Type Registry" is its relationship with the "ALTO Address Types Registry" already defined in Section 14.4 of [RFC7285]. In particular, the entity identifier of a type of an entity domain registered in the "ALTO Entity Domain Type Registry" MAY match an address type defined in "ALTO Address Type Registry". It is necessary to precisely define and guarantee the consistency between "ALTO Address Type Registry" and "ALTO Entity Domain Registry".

We define that the ALTO Entity Domain Type Registry is consistent with ALTO Address Type Registry if two conditions are satisfied:

- When an address type is already or able to be registered in the ALTO Address Type Registry [RFC7285], the same identifier MUST be used when a corresponding entity domain type is registered in the ALTO Entity Domain Type Registry.

- If an ALTO entity domain type has the same identifier as an ALTO address type, their addresses encoding MUST be compatible.

To achieve this consistency, the following items MUST be checked before registering a new ALTO entity domain type in a future document:

- Whether the ALTO Address Type Registry contains an address type that can be used as an entity identifier for the candidate domain identifier. This has been done for the identifiers "ipv4" and "ipv6" in Table 2.

- Whether the candidate entity identifier of the type of the entity domain is able to be an endpoint address, as defined in Sections 2.1 and 2.2 of [RFC7285].

When a new ALTO entity domain type is registered, the consistency with the ALTO Address Type Registry MUST be ensured by the following procedure:

- Test: Do corresponding entity identifiers match a known "network" address type?

  * If yes (e.g., cell, MAC or socket addresses):

+ Test: Is such an address type present in the ALTO Address Type Registry?

  - If yes: Set the new ALTO entity domain type identifier to be the found ALTO address type identifier.

  - If no: Define a new ALTO entity domain type identifier and use it to register a new address type in the ALTO Address Type Registry following Section 14.4 of [RFC7285].

+ Use the new ALTO entity domain type identifier to register a new ALTO entity domain type in the ALTO Entity Domain Type Registry following Section 11.2.2 of this document.

* If no (e.g., pid name, ane name or country code): Proceed with the ALTO Entity Domain Type registration as described in Section 11.2.2.

11.2.2. ALTO Entity Domain Type Registration Process

New ALTO entity domain types are assigned after IETF Review [RFC5226] to ensure that proper documentation regarding the new ALTO entity domain types and their security considerations has been provided. RFCs defining new entity domain types SHOULD indicate how an entity in a registered type of domain is encoded as an EntityID, and, if applicable, the rules defining the entity hierarchy and property inheritance. Updates and deletions of ALTO entity domains follow the same procedure.

Registered ALTO entity domain type identifiers MUST conform to the syntactical requirements specified in Section 3.1.2. Identifiers are to be recorded and displayed as strings.

Requests to the IANA to add a new value to the registry MUST include the following information:

o Identifier: The name of the desired ALTO entity domain type.

o Entity Identifier Encoding: The procedure for encoding the identifier of an entity of the registered type as an EntityID (see Section 3.1.3). If corresponding entity identifiers of an entity domain match a known "network" address type, the Entity Identifier Encoding of this domain identifier MUST include both Address Encoding and Prefix Encoding of the same identifier registered in the ALTO Address Type Registry [RFC7285]. For the purpose of defining properties, an individual entity identifier and the

corresponding full-length prefix MUST be considered aliases for the same entity.

- Hierarchy: If the entities form a hierarchy, the procedure for determining that hierarchy.

- Inheritance: If entities can inherit property values from other entities, the procedure for determining that inheritance.

- Mapping to ALTO Address Type: A boolean value to indicate if the entity domain type can be mapped to the ALTO address type with the same identifier.

- Security Considerations: In some usage scenarios, entity identifiers carried in ALTO Protocol messages may reveal information about an ALTO client or an ALTO service provider. Applications and ALTO service providers using addresses of the registered type should be made aware of how (or if) the addressing scheme relates to private information and network proximity.

This specification requests registration of the identifiers "ipv4", "ipv6" and "pid", as shown in Table 2.

11.3. ALTO Entity Property Type Registry

This document requests IANA to create and maintain the "ALTO Entity Property Type Registry", listed in Table 3.

To distinguish with the "ALTO Endpoint Property Type Registry", each entry in this registry is an ALTO entity property type defined in Section 3.2.1. Thus, registered ALTO entity property type identifier MUST conform to the syntactical requirements specified in that section.

The initial registered ALTO entity property types are listed in Table 3.

Identifier	Intended Semantics
pid	See Section 7.1.1 of [RFC7285]

Table 3: ALTO Entity Property Types.

Requests to the IANA to add a new value to the registry MUST include the following information:

- Identifier: The unique id for the desired ALTO entity property type. The format MUST be as defined in Section 3.2.1 of this document. It includes the information of the applied ALTO entity domain and the property name.

- Intended Semantics: ALTO entity properties carry with them semantics to guide their usage by ALTO clients. Hence, a document defining a new type SHOULD provide guidance to both ALTO service providers and applications utilizing ALTO clients as to how values of the registered ALTO entity property should be interpreted.

This document requests registration of the identifier "pid", as shown in Table 3.

11.4. ALTO Resource-Specific Entity Domain Registries

11.4.1. Network Map

Media-type: application/alto-networkmap+json

Entity Domain Type	Intended Semantics
ipv4	See Section 5.1.1
ipv6	See Section 5.1.1
pid	See Section 5.1.1

Table 4: ALTO Network Map Resource-Specific Entity Domain.

11.4.2. Endpoint Property

Media-type: application/alto-endpointprop+json

Entity Domain Type	Intended Semantics
ipv4	See Section 5.2.1
ipv6	See Section 5.2.1

Table 5: ALTO Endpoint Property Resource-Specific Entity Domain.

11.5. ALTO Resource Entity Property Mapping Registries

11.5.1. Network Map

Media-type: application/alto-networkmap+json

Mapping Descriptor	Entity Domain Type	Property Type	Intended Semantics
ipv4 -> pid	ipv4	pid	See Section 5.1.2
ipv6 -> pid	ipv6	pid	See Section 5.1.2

Table 6: ALTO Network Map Entity Property Mapping.

12. Acknowledgments

The authors would like to thank discussions with Kai Gao, Qiao Xiang, Shawn Lin, Xin Wang, Danny Perez, and Vijay Gurbani. The authors thank Dawn Chen (Tongji University), and Shenshen Chen (Tongji/Yale University) for their contributions to earlier drafts.

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, DOI 10.17487/RFC4632, August 2006, <https://www.rfc-editor.org/info/rfc4632>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <https://www.rfc-editor.org/info/rfc7285>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <https://www.rfc-editor.org/info/rfc7921>.

Authors' Addresses

Wendy Roome
Nokia Bell Labs (Retired)
124 Burlington Rd
Murray Hill, NJ 07974
USA

Phone: +1-908-464-6975
Email: wendy@wdroome.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
NOZAY 91460
FRANCE

Email: Sabine.Randriamasy@nokia-bell-labs.com

Y. Richard Yang
Yale University
51 Prospect Street
New Haven, CT 06511
USA

Phone: +1-203-432-6400
Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com

Kai Gao
Sichuan University
Chengdu 610000
China

Email: kaigao@scu.edu.cn

CDNI
Internet-Draft
Intended status: Standards Track
Expires: February 15, 2020

J. Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Y. Yang
Tongji/Yale
K. Ma
Ericsson
J. Peterson
Neustar
X. Lin
J. Zhang
Tongji

August 14, 2019

Content Delivery Network Interconnection (CDNI) Request Routing: CDNI Footprint and Capabilities Advertisement using ALTO draft-ietf-alto-cdni-request-routing-alto-07

Abstract

The Content Delivery Networks Interconnection (CDNI) framework [RFC6707] defines a set of protocols to interconnect CDNs, to achieve multiple goals such as extending the reach of a given CDN to areas that are not covered by that particular CDN. One component that is needed to achieve the goal of CDNI described in [RFC7336] is the CDNI Request Routing Footprint & Capabilities Advertisement interface (FCI). [RFC8008] defines precisely the semantics of FCI and provides guidelines on the FCI protocol, but the exact protocol is explicitly outside the scope of that document. In this document, we follow the guidelines to define an FCI protocol using the Application-Layer Traffic Optimization (ALTO) protocol.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 15, 2020.

Table of Contents

1. Introduction

The ability to interconnect multiple content delivery networks (CDNs) has many benefits, including increased coverage, capability, and reliability. The Content Delivery Networks Interconnection (CDNI) framework [RFC6707] defines four interfaces to achieve interconnection of CDNs: (1) the CDNI Request Routing Interface; (2) the CDNI Metadata Interface; (3) the CDNI Logging Interface; and (4) the CDNI Control Interface.

Among the four interfaces, the CDNI Request Routing Interface provides key functions, as specified in [RFC6707]: "The CDNI Request Routing interface enables a Request Routing function in an Upstream CDN to query a Request Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request. It also allows the Downstream CDN to control what should be returned to the User Agent in the redirection message by the upstream Request Routing function." On a high level, the scope of the CDNI Request Routing Interface, therefore, contains two main tasks: (1) determining if the downstream CDN (dCDN) is willing to accept a delegated content request; (2) redirecting the content request coming from an upstream CDN (uCDN) to the proper entry point or entity in the downstream CDN.

Correspondingly, the request routing interface is broadly divided into two functionalities: (1) CDNI Footprint & Capabilities Advertisement interface (FCI); (2) CDNI Request Routing Redirection interface (RI). Since this document focuses on the first functionality, CDNI FCI, we will describe it in a more detailed way. CDNI FCI is an advertisement from a dCDN to a uCDN (push) or a query from a uCDN to a dCDN (pull) so that the uCDN knows whether it can redirect a particular user request to that dCDN.

A key component in defining CDNI FCI is defining objects describing the footprints and capabilities of a dCDN. Such objects are already in [RFC8008]. A protocol to transport and update such objects between a uCDN and a dCDN, however, is not defined. Hence, the scope of this document is to define such a protocol by introducing a new Application-Layer Traffic Optimization (ALTO) [RFC7285] service called "CDNI FCI Map Service".

There are multiple benefits in using ALTO as a transport protocol, as we discuss in Section 2.2.

The rest of this document is organized as follows. Section 2 provides non-normative background on both CDNI FCI and ALTO. Section 3 introduces the most basic service, called CDNI FCI Map, to realize CDNI FCI using ALTO. Section 4 demonstrates a key benefit of using ALTO: the ability to integrate CDNI FCI with ALTO network maps. Such integration provides a new granularity to describe footprints. Section 5 builds on filtered ALTO maps to introduce filtered CDNI FCI maps using capabilities so that a uCDN can get footprints with given capabilities instead of getting the full map which can be huge. Section 6 further shows a benefit of using ALTO: the ability to query footprint properties using ALTO unified properties. In this way, a uCDN can effectively fetch capabilities of some footprints in which it is interested. IANA and security considerations are discussed in Section 7 and Section 8 respectively.

Throughout this document, we use the terminology for CDNI defined in [RFC6707], [RFC8006], [RFC8008] and we use the terminology for ALTO defined in [RFC7285], [I-D.ietf-alto-unified-props-new].

2. Background

The design of CDNI FCI transport using ALTO depends on the understanding of both FCI semantics and ALTO. Hence, we start with a review of both.

2.1. Semantics of FCI Advertisement

The CDNI document on "Footprint and Capabilities Semantics" [RFC8008] defines the semantics of CDNI FCI, and provides guidance on what Footprint and Capabilities mean in a CDNI context and how a protocol solution should in principle look like. The definitions in [RFC8008] depend on [RFC8006]. Here we briefly summarize key related points of [RFC8008] and [RFC8006]. For a detailed discussion, the reader is referred to the RFCs.

- Footprint and capabilities are tied together and cannot be interpreted independently from each other. Hence, capabilities must be expressed on a per footprint basis. [RFC8008] integrates footprint and capabilities with an approach of "capabilities with footprint restrictions".

- Given that a large part of Footprint and Capabilities Advertisement will actually happen in contractual agreements, the semantics of CDNI Footprint and Capabilities advertisement refers to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint and/or capabilities that it has prior agreed to serve in a contract with a uCDN. Hence, server push and incremental encoding will be necessary techniques.

- Multiple types of footprints (ipv4cidr, ipv6cidr, asn and countrycode) are defined in [RFC8006].

- A "Set of IP-prefixes" can contain both full IP addresses (i.e., a /32 for IPv4 and a /128 for IPv6) and IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in such a footprint.

- For all of these mandatory-to-implement footprint types, footprints can be viewed as constraints for delegating requests to a dCDN: A dCDN footprint advertisement tells the uCDN the limitations for delegating a request to the dCDN. For IP prefixes or ASN(s), the footprint signals to the uCDN that it should consider the dCDN a candidate only if the IP address of the request routing source falls within the prefix set (or ASN, respectively). The CDNI specifications do not define how a given uCDN determines what address ranges are in a particular ASN. Similarly, for country codes, a uCDN should only consider the dCDN

a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e., the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

- The following capabilities are defined as "base" capabilities; that is, they are required in all cases and therefore constitute mandatory capabilities to be supported by the CDNI FCI: (1) Delivery Protocol; (2) Acquisition Protocol; (3) Redirection Mode; (4) Capabilities related to CDNI Logging; (5) Capabilities related to CDNI Metadata.

2.2. ALTO Background and Benefits

Application-Layer Traffic Optimization (ALTO) [RFC7285] is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important information to "guide" the resource provider selection process in distributed applications. Usually, it is assumed that an ALTO server conveys information that these applications cannot or have difficulty to measure themselves [RFC5693].

Originally, ALTO was motivated by optimizing cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. The CDNI problem statement explicitly mentions ALTO as a candidate protocol for "actual algorithms for selection of CDN or Surrogate by Request-Routing systems" [RFC6707].

The following reasons make ALTO a suitable candidate protocol for downstream CDN selection as part of CDNI request routing and in particular for an FCI protocol:

- ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additional information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise. ALTO can help a uCDN to select a proper dCDN by first providing dCDNs' capabilities as well as footprints (see

Section 3) and then providing costs of surrogates in a dCDN by ALTO cost maps.

- The semantics of an ALTO network map is an exact match for the needed information to convey a footprint by a downstream CDN, in particular if such a footprint is being expressed by IP-prefix ranges. Please see Section 4.

- Security: Identifications between uCDNs and dCDNs are extremely important. ALTO maps can be signed and hence provide inherent integrity protection. Please see Section 8.

- RESTful-Design: The ALTO protocol has undergone extensive revisions in order to provide a RESTful design regarding the client-server interaction specified by the protocol. A CDNI FCI interface based on ALTO would inherit this RESTful design. Please see Section 3.

- Error-handling: The ALTO protocol has undergone extensive revisions in order to provide sophisticated error-handling, in particular regarding unexpected cases. A CDNI FCI interface based on ALTO would inherit this thought-through and mature error-handling. Please see Section 5.

- Filtered map service: The ALTO map filtering service would allow a uCDN to query only for parts of an ALTO map. For example, filtered unified property map service can enable a uCDN to query properties of a part of footprints in an effective way (see Section 6).

- Server-initiated Notifications and Incremental Updates: When the footprint or the capabilities of a downstream CDN change (i.e., unexpectedly from the perspective of an upstream CDN), server-initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e., the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect requests to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. The newest design of ALTO supports server pushed incremental updates [I-D.ietf-alto-incr-update-sse].

- Content Availability on Hosts: A dCDN might want to express CDN capabilities in terms of certain content types (e.g., codecs/formats, or content from certain content providers). The new

endpoint property for ALTO would enable a dCDN to make such information available to an upstream CDN. This would enable a uCDN to determine if a given dCDN actually has the capabilities for a given request with respect to the type of content requested.

- Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized downstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e., can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO. ALTO Path Vector Extension [I-D.ietf-alto-path-vector] is designed to allow ALTO clients to query information such as capacity regions for a given set of flows.

3. CDNI FCI Service

The ALTO protocol is based on an ALTO Information Service Framework which consists of several services, where all ALTO services are "provided through a common transport protocol, messaging structure and encoding, and transaction model" [RFC7285]. The ALTO protocol specification [RFC7285] defines several such services, e.g., the ALTO map service.

This document defines a new ALTO Service called "CDNI FCI Service" which conveys JSON objects of media type "application/alto-cdnifci+json". These JSON objects are used to transport BaseAdvertisementObject objects defined in [RFC8008]; this document specifies how to transport such BaseAdvertisementObject objects via the ALTO protocol with the ALTO "CDNI FCI Service". Similar to other ALTO services, this document defines the ALTO information resource for the "CDNI FCI Service" as follows.

3.1. Media Type

The media type of the CDNI FCI resource is "application/alto-cdnifci+json".

3.2. HTTP Method

A CDNI FCI resource is requested using the HTTP GET method.

3.3. Accept Input Parameters

None.

3.4. Capabilities

None.

3.5. Uses

The "uses" field SHOULD NOT appear unless the CDNI FCI resource depends on some ALTO information resources. If the CDNI FCI resource has some dependent resources, the resource IDs of its dependent resources MUST be included into the "uses" field. This document only defines one potential dependent resource for the CDNI FCI resource. See Section 4 for details of when and how to use it. The future documents may extend the CDNI FCI resource and allow other dependent resources.

3.6. Response

The "meta" field of a CDNI FCI response MUST include the "vtag" field defined in Section 10.3 of [RFC7285]. This field provides the version of the retrieved CDNI FCI map.

If a CDNI FCI response depends on an ALTO information resource, it MUST include the "dependent-vtags" field, whose value is an array to indicate the version tags of the resources used, where each resource is specified in "uses" of its IRD entry.

The data component of an ALTO CDNI FCI response is named "cdni-fci", which is a JSON object of type CDNIFCIData:

```
object {
    CDNIFCIData cdni-fci;
} InfoResourceCDNIFCI : ResponseEntityBase;

object {
    BaseAdvertisementObject capabilities<1..*>;
} CDNIFCIData;
```

Specifically, a CDNIFCIData object is a JSON object that includes only one property named "capabilities", whose value is an array of BaseAdvertisementObject objects.

The syntax and semantics of BaseAdvertisementObject are well defined in Section 5.1 of [RFC8008]. A BaseAdvertisementObject object includes multiple properties, including capability-type, capability-

value and footprints, where footprints are defined in Section 4.2.2.2 of [RFC8006].

To be self-contained, we give a non-normative specification of BaseAdvertisementObject below. As mentioned above, the normative specification of BaseAdvertisementObject is in [RFC8008]

```
object {
    JSONString capability-type;
    JSONValue capability-value;
    Footprint footprints<0..*>;
} BaseAdvertisementObject;

object {
    JSONString footprint-type;
    JSONString footprint-value<1..*>;
} Footprint;
```

For each BaseAdvertisementObject, the ALTO client MUST interpret footprints appearing multiple times as if they appeared only once. If footprints in a BaseAdvertisementObject is null or empty or not appearing, the ALTO client MUST understand that the capabilities in this BaseAdvertisementObject have the "global" coverage.

Note: Further optimization of BaseAdvertisement objects to effectively provide the advertisement of capabilities with footprint restrictions is certainly possible. For example, these two examples below both describe that the dCDN can provide capabilities ["http/1.1", "https/1.1"] for the same footprints. However, the latter one is smaller in its size.

EXAMPLE 1

```
{
  "meta" : {...},
  "cdni-fci": {
    "capabilities": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "http/1.1"
          ]
        },
        "footprints": [
          <Footprint objects>
        ]
      },
      {
```

```
              "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": [
                  "https/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            }
          ]
        }
      }
```

EXAMPLE 2

```
      {
        "meta" : {...},
        "cdni-fci": {
          "capabilities": [
            {
              "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": [
                  "https/1.1",
                  "http/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            }
          ]
        }
      }
```

Since such optimizations are not required for the basic interconnection of CDNs, the specifics of such mechanisms are outside the scope of this document.

3.7. Examples

3.7.1. IRD Example

Below is the information resource directory (IRD) of a simple, example ALTO server. The server provides both base ALTO information resources (e.g., network maps) and CDNI FCI related information resources (e.g., CDNI FCI resource), demonstrating a single, integrated environment.

Specifically, the IRD announces two network maps, one CDNI FCI resource without dependency, one CDNI FCI resource depending on a network map, one filtered CDNI FCI resource to be defined in Section 5, one unified property map including "cdni-fci-capabilities" as its entity property, one filtered unified property map including "cdni-fci-capabilities" and "pid" as its entity properties, and two update stream services (one for updating CDNI FCI resources, and the other for updating property maps).

```
 GET /directory HTTP/1.1
 Host: alto.example.com
 Accept: application/alto-directory+json,application/alto-error+json

 {
   "meta" : { ... },
   "resources": {
     "my-default-network-map": {
       "uri" : "http://alto.example.com/networkmap",
       "media-type" : "application/alto-networkmap+json"
     },
     "my-eu-netmap" : {
       "uri" : "http://alto.example.com/myeunetmap",
       "media-type" : "application/alto-networkmap+json"
     },
     "my-default-cdnifci": {
       "uri" : "http://alto.example.com/cdnifci",
       "media-type": "application/alto-cdnifci+json"
     },
     "my-filtered-cdnifci" : {
       "uri" : "http://alto.example.com/cdnifci/filtered",
       "media-type" : "application/alto-cdnifci+json",
       "accepts" : "application/alto-cdnifcifilter+json",
       "uses" : [ "my-default-cdnifci" ]
     },
     "my-cdnifci-with-pid-footprints": {
       "uri" : "http://alto.example.com/networkcdnifci",
       "media-type" : "application/alto-cdnifci+json",
       "uses" : [ "my-eu-netmap" ]
     },
     "cdnifci-property-map" : {
       "uri" : "http://alto.example.com/propmap/full/cdnifci",
       "media-type" : "application/alto-propmap+json",
       "uses": [ "my-default-cdni" ],
       "capabilities" : {
         "mappings": {
           "ipv4": [ "my-default-cdni.cdni-fci-capabilities" ],
           "ipv6": [ "my-default-cdni.cdni-fci-capabilities" ],
           "countrycode": [
```

```
            "my-default-cdni.cdni-fci-capabilities" ],
          "asn": [ "my-default-cdni.cdni-fci-capabilities" ],
        }
      }
    },
    "filtered-cdnifci-property-map" : {
      "uri" : "http://alto.example.com/propmap/lookup/cdnifci-pid",
      "media-type" : "application/alto-propmap+json",
      "accepts" : "application/alto-propmapparams+json",
      "uses": [ "my-default-cdni", "my-default-network-map" ],
      "capabilities" : {
        "mappings": {
          "ipv4": [ "my-default-cdni.cdni-fci-capabilities",
                    "my-default-network-map.pid" ],
          "ipv6": [ "my-default-cdni.cdni-fci-capabilities",
                    "my-default-network-map.pid" ],
          "countrycode": [
            "my-default-cdni.cdni-fci-capabilities" ],
          "asn": [ "my-default-cdni.cdni-fci-capabilities" ],
        }
      }
    },
    "update-my-cdni-fci" : {
      "uri": "http:///alto.example.com/updates/cdnifci",
      "media-type" : "text/event-stream",
      "accepts" : "application/alto-updatestreamparams+json",
      "uses" : [
        "my-default-network-map",
        "my-eu-netmap",
        "my-default-cdnifci",
        "my-filtered-cdnifci"
        "my-cdnifci-with-pid-footprints"
      ],
      "capabilities" : {
        "incremental-change-media-types" : {
         "my-default-network-map" : "application/json-patch+json",
         "my-eu-netmap" : "application/json-patch+json",
         "my-default-cdnifci" :
         "application/merge-patch+json,application/json-patch+json",
         "my-filtered-cdnifci" :
         "application/merge-patch+jso,application/json-patch+json",
         "my-cdnifci-with-pid-footprints" :
         "application/merge-patch+json,application/json-patch+json"
        }
      }
    },
    "update-my-props": {
      "uri" : "http://alto.example.com/updates/properties",
```

```
          "media-type" : "text/event-stream",
          "uses" : [
            "cdnifci-property-map",
            "filtered-cdnifci-property-map"
          ],
          "capabilities" : {
            "incremental-change-media-types": {
             "cdnifci-property-map" :
             "application/merge-patch+json,application/json-patch+json",
             "filtered-cdnifci-property-map":
             "application/merge-patch+json,application/json-patch+json"
            }
          }
        }
      }
    }
```

3.7.2. Basic Example

In this example, we demonstrate a simple CDNI FCI resource; this resource does not depend on other resources. There are three BaseAdvertisementObjects in this map and these objects' capabilities are http/1.1 delivery protocol, [http/1.1, https/1.1] delivery protocol and https/1.1 acquisition protocol respectively.

```
  GET /cdnifci HTTP/1.1
  Host: alto.example.com
  Accept: application/alto-cdnifci+json,
          application/alto-error+json

  HTTP/1.1 200 OK
  Content-Length: XXX
  Content-Type: application/alto-cdnifci+json
  {
    "meta" : {
      "vtag": {
        "resource-id": "my-default-cdnifci",
        "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
      }
    },
    "cdni-fci": {
      "capabilities": [
        {
          "capability-type": "FCI.DeliveryProtocol",
          "capability-value": {
            "delivery-protocols": [
              "http/1.1"
            ]
```

```
            },
            "footprints": [
              <Footprint objects>
            ]
          },
          {
            "capability-type": "FCI.DeliveryProtocol",
            "capability-value": {
              "delivery-protocols": [
                "https/1.1",
                "http/1.1"
              ]
            },
            "footprints": [
              <Footprint objects>
            ]
          },
          {
            "capability-type": "FCI.AcquisitionProtocol",
            "capability-value": {
              "acquisition-protocols": [
                "https/1.1"
              ]
            },
            "footprints": [
              <Footprint objects>
            ]
          }
        ]
      }
    }
```

3.7.3. Incremental Updates Example

A benefit of using ALTO to provide CDNI FCI maps is that such maps can be updated using ALTO incremental updates. Below is an example that also shows the benefit of having both JSON merge patch and JSON patch to encode updates.

At first, an ALTO client requests updates for "my-default-cdnifci", and the ALTO server returns the "control-uri" followed by the full CDNI FCI response. Then when there is a change in the delivery-protocols in that ‘http/2‘ is removed (from http/1.1 and http/2 to only http/1.1) due to maintenance of the http/2 clusters, the ALTO server uses JSON merge patch to encode the change and pushes the change to the ALTO client. Later on, the ALTO server notifies the ALTO client that "ipv4:192.0.2.0/24" is added into the footprint for

delivery-protocol http/1.1 by sending the change encoded by JSON patch to the ALTO client.

```
POST /updates/cdnifci HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "add": {
    "my-cdnifci-stream": {
        "resource-id": "my-default-cdnifci"
    }
}

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/3141592653589"}

event: application/alto-cdnifci+json,my-default-cdnifci
data: { ... full CDNI FCI map ... }

event: application/merge-patch+json,my-default-cdnifci
data: {
data:   "meta": {
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   "cdni-fci": {
data:     "capabilities": [
data:       {
data:         "capability-type": "FCI.DeliveryProtocol",
data:         "capability-value": {
data:           "delivery-protocols": [
data:             "http/1.1"
data:           ]
data:         },
data:         "footprints": [
data:           <Footprint objects in only http/1.1>
data:         ]
data:       }
data:     ]
data:   }
```

```
     data: }

     event: application/json-patch+json,my-default-cdnifci
     data: [
     data:   {
     data:     "op": "replace",
     data:     "path": "/meta/vtag/tag",
     data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
     data:   },
     data:   { "op": "add",
     data:     "path": "/cdni-fci/capabilities/0/footprints/-",
     data:     "value": "ipv4:192.0.2.0/24"
     data:   }
     data: ]
```

4. CDNI FCI Service using ALTO Network Map

4.1. Network Map Footprint Type: altopid

The ALTO protocol defines a concept called PID to represent a group of IPv4 or IPv6 addresses which can be applied the same management policy. The PID is an alternative to the pre-defined CDNI footprint types (i.e., ipv4cidr, ipv6cidr, asn, and countrycode).

Specifically, a CDNI FCI resource can depend on an ALTO network map resource and use a new CDNI Footprint Type called "altopid" to compress its CDNI Footprint Payload.

"altopid" footprint type indicates that the corresponding footprint value is a list of PIDNames as defined in [RFC7285]. These PIDNames are references of PIDs in a network map resource. Hence a CDNI FCI with "altopid" footprints depends on a network map. For such a CDNI FCI map, the resource id of its dependent network map MUST be included in the "uses" field of its IRD entry, and the "dependent-vtag" field with a reference to this network map MUST be included in its response (see the example in Section 4.2.3).

4.2. Examples

4.2.1. IRD Example

We use the same IRD example given in Section 3.7.1.

4.2.2. ALTO Network Map for CDNI FCI Footprints Example

Below is an example network map whose resource id is "my-eu-netmap", and this map is referenced by the CDNI FCI example in Section 4.2.3.

```
GET /networkmap HTTP/1.1
Host: http://alto.example.com/myeunetmap
Accept: application/alto-networkmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/alto-networkmap+json

{
  "meta" : {
    "vtag": [
      {"resource-id": "my-eu-netmap",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
      }
    ]
  },
  "network-map" : {
    "south-france" : {
      "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25" ]
    },
    "germany" : {
      "ipv4" : [ "192.0.3.0/24"]
    }
  }
}
```

4.2.3. ALTO PID Footprints in CDNI FCI

In this example, we show a CDNI FCI resource that depends on a network map described in Section 4.2.2.

```
GET /networkcdnifci HTTP/1.1
Host: alto.example.com
Accept: application/alto-cdnifci+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-cdnifci+json

{
  "meta" : {
    "dependent-vtags" : [
      {
        "resource-id": "my-eu-netmap",
        "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
      }
    ]
  },
  "cdni-fci": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "http/1.1"
        ]
      },
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "https/1.1"
        ],
        "footprints": [
          { "footprint-type": "altopid",
            "footprint-value": [
              "germany",
              "south-france"
            ]
          }
        ]
      }
    ]
  }
}
```

4.2.4. Incremental Updates Example

In this example, the ALTO client is interested in changes of "my-cdnifci-with-pid-footprints". Considering two changes, the first one is to change footprints of http/1.1 Delivery Protocol capability, and the second one is to remove "south-france" from the footprints of https/1.1 delivery protocol capability.

```
POST /updates/cdnifci HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
```

```
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "add": {
    "my-network-map-cdnifci-stream": {
        "resource-id": "my-cdnifci-with-pid-footprints"
    }
}

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/3141592653590"}

event: application/alto-cdnifci+json,my-fci-stream
data: { ... full CDNI FCI resource ... }

event: application/merge-patch+json,my-fci-stream
data: {
data:   "meta": {
data:     "dependent-vtags" : [
data:      {
data:        "resource-id": "my-eu-netmap",
data:        "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
data:      }
data:     ],
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   {
data:     "capability-type": "FCI.DeliveryProtocol",
data:     "capability-value": {
data:       "delivery-protocols": [
data:         "http/1.1"
data:       ]
data:     },
data:     "footprints": [
data:       <All footprint objects in http/1.1>
data:     ]
data:   }
data: }

event: application/json-patch+json,my-fci-stream
data: [
```

```
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "remove",
data:     "path": "/cdni-fci/capabilities/2/footprints/0/
data:       footprint-value/1",
data:   }
data: ]
```

5. Filtered CDNI FCI using Capabilities

Section 3 and Section 4 describe CDNI FCI Service which can be used to enable a uCDN to get capabilities with footprints constrains from dCDNs. However, always getting full CDNI FCI resources from dCDNs is very inefficient, hence we introduce a new service named "Filtered CDNI FCI Service" to allow a client to filter a CDNI FCI resource using a client-given set of capabilities. For each entry of the CDNI FCI response, only if the entry contains at least one of the client-given capabilities will it be returned to the client. The relationship between a filtered CDNI FCI resource and a CDNI FCI resource is similar to the relationship between a filtered network/cost map and a network/cost map.

5.1. Media Type

A filtered CDNI FCI resource uses the same media type defined for the CDNI FCI resource in Section 3.1.

5.2. HTTP Method

A filtered CDNI FCI resource is requested using the HTTP POST method.

5.3. Accept Input Parameters

The input parameters for a filtered CDNI FCI resource are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-cdnifcifilter+json" which is a JSON object of type ReqFilteredCDNIFCI, where:

```
object {
    JSONString capability-type;
    JSONValue capability-value;
} CDNIFCICapability;

object {
    [CDNIFCICapability cdni-fci-capabilities<0..*>;]
} ReqFilteredCDNIFCI;
```

with fields:

capability-type: The same as Base Advertisement Object's capability-type defined in Section 5.1 of [RFC8008].

capability-value: The same as Base Advertisement Object's capability-value defined in Section 5.1 of [RFC8008].

cdni-fci-capabilities: A list of CDNI FCI capabilities defined in Section 5.1 of [RFC8008] for which footprints are to be returned. If a list is empty or not appearing, the ALTO server MUST interpret it as a request for the full CDNI FCI resource. The ALTO server MUST interpret entries appearing in a list multiple times as if they appeared only once. If the ALTO server does not define any footprints for a CDNI capability, it MUST omit this capability from the response.

5.4. Capabilities

None.

5.5. Uses

The resource ID of the CDNI FCI resource based on which the filtering is performed.

5.6. Response

The response MUST indicate an error, using ALTO protocol error handling specified in Section 8.5 of the ALTO protocol [RFC7285], if the request is invalid.

Specifically, a filtered CDNI FCI request is invalid if:

- the value of "capability-type" is null;

- the value of "capability-value" is null;

- the value of "capability-value" is inconsistent with "capability-type".

When a request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285], and the "value" field of the error message SHOULD indicate this CDNI FCI capability.

The ALTO server returns a filtered CDNI FCI resource for a valid request. The format of a filtered CDNI FCI resource is the same as an full CDNI FCI resource (See Section 3.6.)

The returned CDNI FCI resource MUST contain only BaseAdvertisementObject objects whose CDNI capability object is the superset of one of CDNI capability object in "cdni-fci-capabilities". Specifically, that a CDNI capability object A is the superset of another CDNI capability object B means that these two CDNI capability objects have the same capability type and mandatory properties in capability value of A MUST include mandatory properties in capability value of B semantically. See Section 5.7.2 for a concrete example.

The version tag included in the "vtag" field of the response MUST correspond to the full CDNI FCI resource from which the filtered CDNI FCI resource is provided. This ensures that a single, canonical version tag is used independently of any filtering that is requested by an ALTO client.

5.7. Examples

5.7.1. IRD Example

We use the same IRD example by Section 3.7.1.

5.7.2. Basic Example

This example filters the full CDNI FCI resource in Section 3.7.2 by selecting only http/1.1 delivery protocol capability. Only the first two BaseAdvertisementObjects in the full resource will be returned because the first object's capability is http/1.1 delivery protocol and the second object's capability is http/1.1 and https/1.1 delivery protocols which is the superset of http/1.1 delivery protocol.

```
POST /cdnifci/filtered HTTP/1.1
HOST: alto.example.com
Content-Type: application/cdnifilter+json
Accept: application/alto-cdnifci+json

{
```

```
      "cdni-fci-capabilities": [
        {
          "capability-type": "FCI.DeliveryProtocol",
          "capability-value": {
            "delivery-protocols": [
              "http/1.1"
            ]
          }
        }
      ]
    }

    HTTP/1.1 200 OK
    Content-Length: XXX
    Content-Type: application/alto-cdnifci+json
    {
      "meta" : {
        "vtag": {
          "resource-id": "my-default-cdnifci",
          "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
        }
      },
      "cdni-fci": {
        "capabilities": [
          {
            "capability-type": "FCI.DeliveryProtocol",
            "capability-value": {
              "delivery-protocols": [
                "http/1.1"
              ]
            },
            "footprints": [
              <Footprint objects>
            ]
          },
          {
            "capability-type": "FCI.DeliveryProtocol",
            "capability-value": {
              "delivery-protocols": [
                "https/1.1",
                "http/1.1"
              ]
            },
            "footprints": [
              <Footprint objects>
            ]
          }
        ]
```

```
      }
    }
```

5.7.3. Incremental Updates Example

In this example, the ALTO client only cares about the updates of one Delivery Protocol object whose value is "http/1.1". So it adds its limitation of capabilities in "input" field of the POST request.

```
  POST /updates/cdnifci HTTP/1.1
  Host: fcialtoupdate.example.com
  Accept: text/event-stream,application/alto-error+json
  Content-Type: application/alto-updatestreamparams+json
  Content-Length: ###

  { "add": {
      "my-fci-stream": {
          "resource-id": "my-filtered-cdnifci",
          "input": {
            "cdni-fci-capabilities": [
            {
              "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": [
                  "http/1.1"
                ]
              }
            }
          ]
        }
      }
    }
  }

  HTTP/1.1 200 OK
  Connection: keep-alive
  Content-Type: text/event-stream

  event: application/alto-updatestreamcontrol+json
  data: {"control-uri":
  data: "http://alto.example.com/updates/streams/3141592653590"}

  event: application/alto-cdnifci+json,my-fci-stream
  data: { ... full filtered CDNI FCI resource ... }

  event: application/merge-patch+json,my-fci-stream
  data: {
  data:   "meta": {
```

```
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   {
data:     "capability-type": "FCI.DeliveryProtocol",
data:     "capability-value": {
data:       "delivery-protocols": [
data:         "http/1.1"
data:       ]
data:     },
data:     "footprints": [
data:       <All footprint objects in http/1.1>
data:     ]
data:   }
data: }

event: application/json-patch+json,my-fci-stream
data: [
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "add",
data:     "path": "/cdni-fci/capabilities/0/footprints/-",
data:     "value": "ipv4:192.0.2.0/24"
data:   }
data: ]
```

6. Query Footprint Properties using ALTO Property Map Service

Besides retrieving footprints of given capabilities, another common requirement for uCDN is to query CDNI capabilities of given footprints.

Considering each footprint as an entity with properties including CDNI capabilities, the most natrual way to satisfy this requirement is to use the ALTO property map defined in [I-D.ietf-alto-unified-props-new]. In this section, we describe how ALTO clients look up properties for individual footprints. We firstly describe how to represent footprint objects as entities in the ALTO property map. And then we provide examples of the full property map and the filtered property map supporting CDNI capabilities, and their incremental updates.

6.1. Representing Footprint Objects as Unified Property Map Entities

A footprint object has two properties: footprint-type and footprint-value. A footprint-value is an array of footprint values conforming to the specification associated with the registered footprint type ("ipv4cidr", "ipv6cidr", "asn", and "countrycode"). Considering each ALTO entity defined in [I-D.ietf-alto-unified-props-new] also has two properties: entity domain type and domain-specific identifier, a straightforward approach to represent a footprint as an ALTO entity is to regard its footprint-type as an entity domain type, and its footprint value as a domain-specific identifier. According to [I-D.ietf-alto-unified-props-new], "ipv4" and "ipv6" are two predefined entity domain types, which can be used to represent "ipv4cidr" and "ipv6cidr" footprints respectively. However, no existing entity domain type can represent "asn" and "countrycode" footprints. To represent footprint-type "asn" and "countrycode", this document registers two new domains in Section 7 in addition to the ones in [I-D.ietf-alto-unified-props-new].

Here is an example of representing a footprint object as a set of entities in the ALTO property map.

{"footprint-type": "ipv4cidr", "footprint-value": ["192.0.2.0/24", "198.51.100.0/24"]} --> "ipv4:192.168.2.0/24", "ipv4:198.51.100.0/24"

6.1.1. ASN Domain

The ASN domain associates property values with Autonomous Systems in the Internet.

6.1.1.1. Entity Domain Type

asn

6.1.1.2. Domain-Specific Entity Identifiers

The entity identifiers of entities in an asn domain is encoded as a string consisting of the characters "as" (in lowercase) followed by the Autonomous System Number [RFC6793].

6.1.1.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with ASN.

6.1.2. COUNTRYCODE Domain

The COUNTRYCODE domain associates property values with countries.

6.1.2.1. Entity Domain Type

countrycode

6.1.2.2. Domain-Specific Entity Identifiers

The entity identifiers of entities in a countrycode domain is encoded as an ISO 3166-1 alpha-2 code [ISO3166-1] in lowercase.

6.1.2.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with country codes.

6.2. Examples

6.2.1. IRD Example

We use the same IRD example given by Section 3.7.1.

6.2.2. Property Map Example

This example shows a full property map in which entities are footprints and entities' property is "cdni-fci-capabilities".

```
GET /propmap/full/cdnifci HTTP/1.1
HOST: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "meta": {
      "dependent-vtags": [
        {"resource-id": "my-default-cdnifci",
         "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
      ]
    },
    "countrycode:us": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}]
    },
    "ipv4:192.0.2.0/24": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}]
    },
    "ipv4:198.51.100.0/24": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}]
    },
    "ipv6:2001:db8::/32": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}]
    },
    "asn:as64496": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1",
                                                     "https/1.1"]}}]
    }
  }
}
```

6.2.3. Filtered Property Map Example

In this example, we use filtered property map service to get "pid" and "cdni-fci-capabilities" properties for two footprints "ipv4:192.0.2.0/24" and "ipv6:2001:db8::/32".

```
POST /propmap/lookup/cdnifci-pid HTTP/1.1
HOST: alto.example.com
Content-Type: application/alto-propmapparams+json
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length:

{
  "entities": [
    "ipv4:192.0.2.0/24",
    "ipv6:2001:db8::/32"
  ],
  "properties": [ "my-default-cdnifci.cdni-fci-capabilities",
                  "my-default-networkmap.pid" ]
}
```

```
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "meta": {
      "dependent-vtags": [
         {"resource-id": "my-default-cdnifci",
           "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"},
         {"resource-id": "my-default-networkmap",
           "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
      ]
    },
    "ipv4:192.0.2.0/24": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}],
      "my-default-networkmap.pid": "pid1"
    },
    "ipv6:2001:db8::/32": {
      "my-default-cdnifci.cdni-fci-capabilities": [
        {"capability-type": "FCI.DeliveryProtocol",
         "capability-value": {"delivery-protocols": ["http/1.1"]}}],
      "my-default-networkmap.pid": "pid3"
    }
  }
}
```

6.2.4. Incremental Updates Example

In this example, here is a client want to request updates for the properties "cdni-fci-capabilities" and "pid" for two footprints "ipv4:192.0.2.0/24" and "countrycode:fr".

```
POST /updates/properties HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "add": {
  "property-map-including-capability-property": {
    "resource-id": "filtered-cdnifci-property-map",
      "input": {
        "properties": [ "my-default-cdnifci.cdni-fci-capabilities",
                        "my-default-networkmap.pid" ],
        "entities": [
```

```
            "ipv4:192.0.2.0/24",
            "ipv6:2001:db8::/32"
          ]
        }
      }
    }

    HTTP/1.1 200 OK
    Connection: keep-alive
    Content-Type: text/event-stream

    event: application/alto-updatestreamcontrol+json
    data: {"control-uri":
    data: "http://alto.example.com/updates/streams/1414213562373"}

    event: application/alto-cdnifci+json,my-fci-stream
    data: { ... full filtered unified property map ... }

    event: application/merge-patch+json,my-fci-stream
    data: {
    data:   "property-map":
    data:   {
    data:     "meta": {
    data:       "dependent-vtags": [
    data:         {"resource-id": "my-default-cdnifci",
    data:          "tag": "2beeac8ee23c3dd1e98a73fd30df80ece9fa5627"},
    data:         {"resource-id": "my-default-networkmap",
    data:          "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
    data:       ]
    data:     },
    data:     "ipv4:192.0.2.0/24":
    data:     {
    data:     "my-default-cdnifci.cdni-fci-capabilities": [
    data:       {"capability-type": "FCI.DeliveryProtocol",
    data:        "capability-value": {
    data:          "delivery-protocols": ["http/1.1"]}}]
    data:     }
    data:   }
    data: }

    event: application/json-patch+json,my-fci-stream
    data: {[
    data: {
    data:   { "op": "replace",
    data:     "path": "/meta/dependent-vtags/0/tag",
    data:     "value": "61b23185a50dc7b334577507e8f00ff8c3b409e4"
    data:   },
    data:   { "op": "replace",
```

```
data:     "path":
data:     "/property-map/countrycode:fr/my-default-networkmap.pid",
data:     "value": "pid5"
data:   }
data: }
data: ]}
```

7. IANA Considerations

7.1. CDNI Metadata Footprint Type Registry

Footprint Type	Description	Specification
altopid	A list of PID-names	RFCthis

Table 1: CDNI Metadata Footprint Type

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7.2. ALTO Entity Domain Type Registry

As proposed in Section 11.2 of [I-D.ietf-alto-unified-props-new], "ALTO Entity Domain Type Registry" is requested. Besides, two new entity domain types are to be registered, listed in Table 2.

Identifier	Entity Address Encoding	Hierarchy & Inheritance
asn	See Section 6.1.1.2	None
countrycode	See Section 6.1.2.2	None

Table 2: ALTO Entity Domain Types

7.3. ALTO Entity Property Type Registry

As proposed in Section 11.3 of [I-D.ietf-alto-unified-props-new], "ALTO Entity Property Type Registry" is required. Besides, a new entity property type is to be registred, listed in Table 3.

Identifier	Intended Semantics
cdni-fci-capabilities	An array of CDNI FCI capability objects

Table 3: ALTO CDNI FCI Property Type

8. Security Considerations

As an extension of the base ALTO protocol [RFC7285], this document fits into the architecture of the base protocol, and hence the Security Considerations (Section 15) of the base protocol fully apply when this extension is provided by an ALTO server.

In the context of CDNI FCI, additional security considerations should be included as follows.

For authenticity and integrity of ALTO information, an attacker may disguise itself as an ALTO server for a dCDN, and provide false capabilities and footprints to a uCDN using the CDNI FCI map. Such false information may lead a uCDN to (1) select an incorrect dCDN to serve user requests or (2) skip uCDNs in good conditions.

For potential undesirable guidance from authenticated ALTO information, dCDNs can provide a uCDN with limited capabilities and smaller footprint coverage so that dCDNs can avoid transferring traffic for a uCDN which they should have to transfer.

For confidentiality and privacy of ALTO information, footprint properties integrated with ALTO unified property may expose network location identifiers (e.g., IP addresses or fine-grained PIDs).

For availability of ALTO services, an attacker may get the potential huge full CDNI FCI maps from an ALTO server in a dCDN continuously to run out of bandwidth resources of that ALTO server or may query filtered CDNI FCI services with complex capabilities to run out of computation resources of an ALTO server.

Protection strategies described in RFC 7285 can solve problems mentioned above well. However, the isolation of full/filtered CDNI FCI maps should also be considered.

If a dCDN signs agreements with multiple uCDNs, it must isolate full/filtered CDNI FCI maps for different uCDNs in that uCDNs will not redirect requests which should not have to served by this dCDN to this dCDN and it may not disclose extra information to uCDNs.

To avoid this risk, a dCDN may consider generating URIs of different full/filtered CDNI FCI maps by hashing its company ID, a uCDN's company ID as well as their agreements. And it needs to avoid expoing all full/filtered CDNI FCI maps resources in one of its IRDs.

9. Acknowledgments

The authors would like to thank Daryl Malas, Matt Caulfield for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

10. References

10.1. Normative References

[ISO3166-1] The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1:2013, 2013.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, DOI 10.17487/RFC5693, October 2009, <https://www.rfc-editor.org/info/rfc5693>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <https://www.rfc-editor.org/info/rfc6793>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <https://www.rfc-editor.org/info/rfc7285>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

10.2. Informative References

[I-D.ietf-alto-incr-update-sse]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", draft-ietf-alto-incr-update-sse-17 (work in progress), July 2019.

[I-D.ietf-alto-path-vector]
Gao, K., Lee, Y., Randriamasy, S., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector", draft-ietf-alto-path-vector-08 (work in progress), July 2019.

[I-D.ietf-alto-unified-props-new]
Roome, W., Randriamasy, S., Yang, Y., and J. Zhang, "Unified Properties for the ALTO Protocol", draft-ietf-alto-unified-props-new-08 (work in progress), July 2019.

[I-D.jenkins-alto-cdn-use-cases]
Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-03 (work in progress), June 2012.

Authors' Addresses

Jan Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Schellingstrasse 24
Stuttgart 70174
Germany

Phone: +49-0711-8926-2801
Email: jan.seedorf@hft-stuttgart.de

Y.R. Yang
Tongji/Yale University
51 Prospect Street
New Haven, CT 06511
United States of America

Email: yry@cs.yale.edu
URI: http://www.cs.yale.edu/~yry/

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
United States of America

Phone: +1-978-844-5100
Email: kevin.j.ma@ericsson.com

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord, CA 94520
United States of America

Email: jon.peterson@neustar.biz

Xiao Shawn Lin
Tongji University
4800 Cao'an Hwy
Shanghai 201804
China

Email: x.shawn.lin@gmail.com

Jingxuan Jensen Zhang
Tongji University
4800 Cao’an Hwy
Shanghai 201804
China

Email: jingxuan.zhang@tongji.edu.cn