IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 19, 2019, accepted December 24, 2019, date of publication December 27, 2019,
date of current version January 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2962705

Updating Data-Center Network With Ultra-Low Latency Data Plane

CHENGYUAN HUANG, JIAO ZHANG, AND TAO HUANG
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing 100876, China
School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China
Purple Mountain Laboratories, Nanjing 211111, China

Corresponding author: Jiao Zhang (jiaozhang@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61872401, and in part by the Ministry of Industry and Information Technology (MIIT) of China "SDN-Based Industrial Network Interconnection and Collaborative Platform Demonstration Application".

ABSTRACT Due to the asynchronous and distributed nature of the data plane, the transition from the initial to final state may result in serious transient congestion in software-defined networking (SDN). Moreover, with the rapid development of ultra-low latency network in data-centers, the control loop between the control and data plane becomes much longer than the ultra-low latency communication in the data plane. In this case, traffic surges significantly in the data plane during the network reconfiguration process, and it becomes harder for the SDN controllers to manipulate the update operations as they expect. In this paper, we consider the traffic variation during the network update and model it as a novel minimum demand violation problem (MDVP). Later, we prove its hardness, and propose a heuristic approximation algorithm to approach the optimal result. Our method brings flexibility for network operators to make a trade-off between the network congestion and update speed. Experiments show that our method can halve the intermediate network states and reduce the demand violation ratio by 36.7 % compared to the state-of-the-art.

INDEX TERMS Software-defined networking (SDN), network update, ultra-low latency network, minimum demand violation problem (MDVP).

I. INTRODUCTION

Because of the flexibility introduced by SDN, network operators are able to reconfigure network frequently to meet the applications' demand. In a typical SDN network, the controller uses the OpenFlow protocol [1] to update the data plane by modifying the underlying flow tables. However, different from the centralized control plane, the data plane remains a distributed system. The update process is not atomic [2]: the slow and uncontrolled update operations of Ternary Content Addressable Memory (TCAM) utilized by SDN switches and disparate transmission delays between a controller and underlay switches cause each switch to be updated independently and asynchronously. The intuitive one-shot mechanism distributes all commands to underlay switches directly, and this careless update scheme may result in serious transient congestion in the process of update [3].

This serious problem has drawn a lot of attention from both academia and industry [3]–[14]. By introducing intermediate network states, they bring more determinacy, thus manage to eliminate or minimize congestion in the process of update. These methods can be mainly categorized into 2 parts: 1) one kind of works models the network resources like spare bandwidth as the constraints in the linear programming (LP), and they find the desired routing for each state by solving the linear programming problem. 2) the other kind of works models the possible network updates and network resources as the elements in the dependency graph. The controller dynamically updates the graph by interacting with the real-time data plane.

All the previous works imply an assumption that all flows have fixed demand and their rates remain the same as their demands in the process of network update. This assumption is correct when the update can be completed before traffic varies in the data plane. In traditional networks, a typical deployment of SDN controllers is out-of-band where controllers connect to the switches within one hop and messages can be delivered directly. On the contrary, the end-to-end latencies between end-hosts are usually as high as tens of milliseconds. The short control loop and long end-to-end latency make the time consuming on the control plan much less than

The associate editor coordinating the review of this manuscript and approving it for publication was Ghufran Ahmed.

the time the traffic control mechanisms at end-hosts take effects.

However, with the emerging ultra-low latency network in data-centers, this assumption cannot be held anymore. Nowadays, the customized hardware in data-center network has already reduced the end-to-end latency to tens of microseconds [15]. Moreover, with the novel hardware-based congestion control algorithm deployed in network interface card (NIC), the convergence time of flows on a congested link can be reduced to several microseconds [16]. The control loop in such a network becomes several orders of magnitude longer than the communication in the data plane network. During multiple interactions between the control and data plane, the volume of traffic will inevitably vary a lot because of the traffic control at end-hosts.

In this paper, we study the transient congestion caused by network updates in the ultra-low latency data plane network. We model the problem as the linear programming as previous works, but considering the flow dynamics during the network update process. To guarantee the performance of high-value critical flows that have fixed demands for bandwidth, we firstly propose the minimum demand violation problem (MDVP) to minimize the overall negative impacts on flows. In MDVP, we propose the novel objective, *demand violation ratio*, to quantify such negative impacts. The main design goals of our method are: *1) minimum demand violation ratio, 2) controllable network congestion, 3) reduced intermediate network states*. In the process of network update, we consider each network state and estimate the resulting flow rates in each stage. Later we use the estimated new flow rate as the input of the next-stage update. By this mean, we keep pace with the real-time flow dynamics tightly, and have better performance than prior static update mechanisms which ignore flow dynamics.

Moreover, we prove the hardness of the original MDVP problem, and relax the original problem to fit it into the standard LP solver. Later, we design an approximation algorithm to compute the update sequence, and we evaluate our algorithms in both intra- and inter- data-center networks. Experiments show that by introducing the appropriate congestion extent parameter, λ, our algorithm can bring flexibility for network operators to do a trade-off between the network congestion and the introduced intermediate states. When setting a proper λ and the number of flows existing in the network is 3000, the intermediate network states can be halved. Finally, in our experiment, we show that our algorithm can reduce the demand violation ratio for flows by 36.7 %, compared to prior works.

The remainder of the paper is organized as follows. § II presents background of ultra-low latency network and SDN update problem. In § III we introduce our network model and describe our motivation examples. In § IV we formulate the minimum demand violation problem (MDVP) and the design details of our algorithms to solve the problem. We evaluate the simulation results in § V, and we conclude in § VI.

II. BACKGROUND AND RELATED WORK

The recent SDN network is composed of the decoupled control and data plane, which results in greater latency than the traditional integrated network. This makes it harder for the SDN controllers to manipulate switches precisely, and can disrupt the network performance seriously. On the other hand, the new trend in data-center network is the pursuit of ultra-low latency, and it desires a network providing a stable end-to-end delay which is usually lower than 100 us. This poses new challenges for SDN network update. In this section, we will briefly introduce the development of the ultra-low latency network and network update problem.

A. ULTRA-LOW LATENCY NETWORK

With the popularity of new applications like distributed machine learning and distributed key-value store, the data-center network is desired to offer a much lower and stabler network latency. As reported in [17], an additional 500us latency can degrade the performance of distributed machine learning application (Spark MLlib) by 60%. However, most of current commodity cloud data-center such as Microsoft Azure and Amazon Web Service (AWS), can only guarantee a network latency at the millisecond level, and they cannot control the tail-latency precisely because of the uncontrollable network queue buildup and packet loss. Moreover, even under the simple circumstance where one source server sends data to one sink server, the mean end-to-end latency of traditional TCP is 20X larger than the basic link latency [18]. This is because the traditional TCP/IP stack is kernel-based, and it requires CPU to run multi-level stacks to accomplish complex network functions. It poses great challenges for CPU to control latency as it has to serve the whole system, not limited to the network functions. When CPU is called to perform the network functions, it will cause great overhead, including context switching, system interrupts, memory copy, etc, which all contribute to the high latency spike.

To relieve the burden on CPU, more customized hardware like remote direct memory access (RDMA) and smartNIC, are designed to take more networking jobs [15], [19]. These techniques greatly reduce the long-tailed latency of kernel-based approaches by totally bypassing the kernel and CPU. Recently, the advanced cloud providers like Microsoft and Alibaba, have been reported to deploy RDMA in their commodity public cloud data-centers at large scale [15], [16]. As shown in [15], in Microsoft's data-center, RDMA effectively reduces the 99-th percentile latency from 700us (TCP/IP) to 90us. Moreover, [16] shows that by designing an effective congestion control mechanism, they control the network latency lower than 10us in their cluster. The ultra-low end-to-end latency shows a great opportunity to react to network events like link failures, network congestion, quickly.

B. THE SDN UPDATE PROBLEM

A network update can involve multiple unsynchronized devices at the data plane, so achieving the consistency is challenging during the updates. Specifically, the whole SDN update problem can be categorized into 3 parts in terms of different properties [20], [21]. *1) Forwarding black hole problem.* The forwarding black hole problem refers to the case where a packet entering an SDN switch cannot match any rule in the forwarding table. The packet will be dropped or forwarded to the SDN controller according to the default actions of SDN switches. *2) Forwarding loop problem.* The forwarding loop problem means that a packet suffers the forwarding loops and cannot be routed to its destination during the network update. These two problems are indeed caused by the route inconsistency, and Reitblatt *et al.* [2] firstly study the problem of route consistency in SDN. They propose the well-known two-phase commit mechanism, a version-stamping mechanism, to ensure the consistent update. This scheme assigns a version number to each configuration and stamps the packets entering the network at the ingress switch. When a network update occurs, this mechanism persists: each packet is processed either using the configuration in place prior to the update, or the configuration in place after the update, but never a mixture of the two. By default, the network operators should adopt one of the approaches like two-phase commit mechanism to avoid the serious problems caused by the route inconsistency.

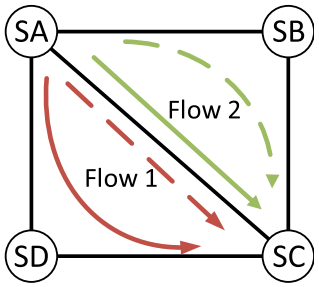


FIGURE 1. In this example, all edges have a unit capacity and both the red and green flows have a unit size. Both red and green flows are updated directly. Solid flows represent stage 1, and dashed flows represent stage 2. If both red and green flows are updated in one shot, red flow can be firstly migrated to the link (switch A, switch C) due to the asynchrony of switches. Thus, both the red flow in stage 2 and green flow in stage 1 are routed through the same link (switch A, switch C). The overall size of these flows is 2, and the capacity constraints for the link (switch A, switch C) is violated.

Even the route consistency guarantees the routing of a single flow, when involving multiple flows, *3) transient congestion* can still happen during the network update, as we illustrate in Fig. 1. [12] thinks of network update problem as the multi-commodity flows problem. [7] considers the dependencies among multiple updates, and uses the classic dependency graph to model the dependencies between network stages. [11] moves one step further than [7] to avoid high overhead in update ordering, they divide the global dependencies among updates into multiple local restrictions. Given that a congestion-free update plan does not always exist, [14] aims to minimize transient congestion instead of eliminating congestion. The previous works concern little about the long update duration, which disrupts the efficiency of updates. [13] aims to address the real-time route update, which jointly considers the optimization of flow route selection in the control plane and update schedule in the data plane. [8] and [6] propose to update the network in a fast and distributed manner to overcome the low scalability of centralized SDN controllers. To make the readers have a better understanding of the SDN update problem, we also present the representative scenarios of the problem in Table 1.

TABLE 1. Scenarios of network update problem.

Network update problem	Representative scenario
Forwarding black hole	Load balance and VM migration
Forwarding loop	Load balance and traffic engineering
Transient Link congestion	Load balance and switch failure repair

III. MOTIVATION AND PRELIMINARIES

In this section, we firstly introduce our network model for the network update problem, and list the key variations. Further, we introduce the motivation using the examples shown in Fig. 3 and Fig. 4 to help readers have a better understanding of our solution which is elaborated in § IV.

A. NETWORK MODEL

A network can be modeled as a directed graph $G=(V,E)$, where V is the set of switches, and E is the set of links between switches. For each link $e \in E$, the capacity of e is denoted by C_e. We denote the set of flows in data-centers as F, thus a single flow $f \in F$. A flow f is routed from origin to destination, and let $P(f)$ be all the feasible paths for the flow f. When network operators plan a network update, the maximum number of update stages should be determined firstly, and we denote the maximum number of update stages by n. Let the set of update stages be $S = \{1, 2, \ldots, s, \ldots, n-1, n\}$, and a specific network stage $s \in S$. Note that the initial and final configurations are at stage 1 and stage n, respectively. For each network stage, a flow f will select one feasible path p among all feasible paths $P(f)$, thus a selected path $p \in P(f)$.

When a flow f goes through a link e, f will consume certain bandwidth according to its demand and real-time network condition, and we denote the demand by d_f. However, the demand of flow f cannot be always guaranteed because congestion happens during network update, and the real-time flow rate r_f^s can be lower than its demand d_f. As we show in Fig. 2, there is a flow f whose demand is d_f at state s. Assume its demand is fully satisfied because of the sufficient bandwidth resource during state s, so the flow rate equals its demand, i.e., $r_f^s = d_f^s$. Later, the network state starts to transform from state s to $s+1$. However, due to the unsynchronized network devices and flows, flow f encounters network congestion during the network state changes. Because of the

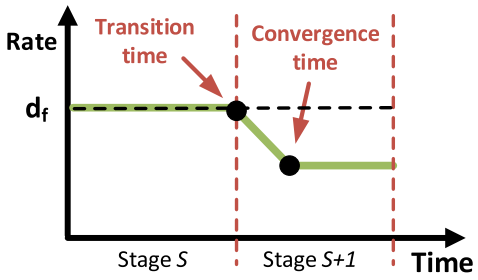


FIGURE 2. Flow rate changes when the network stage transforms.

TABLE 2. Key notations for network model.

G	The directed network graph $G = (V, E)$.
V	The set of switches.
E	The set of links between switches.
e	A link in G.
F	The set of all network flows. All flows are routed in a single path.
f	A network flow in G.
$P(f)$	All possible paths that flow f can go through.
p	The path that flow f goes through.
S	The number of network updates.
s	A state in the sequence of network updates.
d_f	The demand of flow f.
r_f^s	The rate of flow f in stage s.
C_e	The link capacity of link e.
$x_{f,p}^s$	A binary value to represent whether the flow f goes through path p in stage s.

ultra-low latency in the data plane, the flow f will perceive the congestion at the beginning of state $s+1$. Further, it will react to it quickly and converge to a new rate r_f^s' during the state $s+1$, as shown in Fig. 2.

For convenience, we summarize important notations in Table 2.

B. FLOW CLASSIFICATION

A modern data-center hosts diverse applications, and the flows corresponding to these applications have widely varying demands. One type of applications generates high-value data, and the corresponding flows are sensitive to congestion and network delay. This kind of flows wants to consume a constant amount of bandwidth to guarantee the quality of data transmission, and this is also the kind of flows which SDN controller manipulates in this paper.

Furthermore, we categorize the flows into 2 parts in this paper: 1) *satisfied flows*, 2) *throttled flows*. If the demand of a flow, d_f, is fully satisfied during every intermediate state, we denote this kind of flow as the *satisfied flows*. On the other hand, if a flow encounters congestion during the network states transform, the ultra-low latency in the data plane will cause the quick perception of congestion, and the effective congestion control algorithms at end-hosts will lead to the quick convergence on the congested link. By contrast, due to the out-of-band deployment and the complex functions SDN controller performs, the latency of SDN control loop is usually as high as tens of milliseconds. When the SDN controller collects flow information at state s and computes update plan for the following states, it is likely that the flow rates already change significantly. Hence, the assumption of previous works that the flow rates equal to their demands in every intermediate state will not be held. We denote the congested flows whose demand is not satisfied at one intermediate state as *throttled flows*. It is noteworthy that the flow rates will not exceed their demands, so the maximum rate of a flow f, at any state is d_f^s.

C. MOTIVATION EXAMPLE

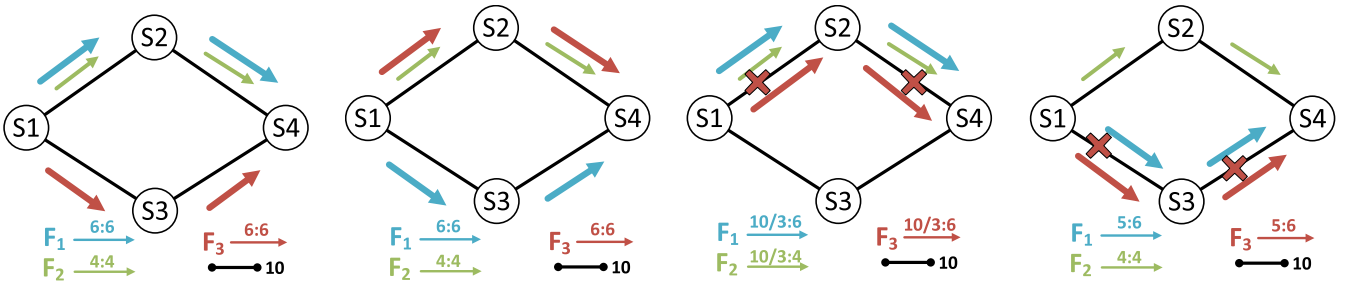
In this section, we provide two motivation examples to show that by considering the real-time network condition and flow characteristics (flow rate and demand), we can successfully obtain all the design objectives, i.e., *minimum demand violation, controllable network congestion, and reduced intermediate states.*

1) THE CASE OF THE FLOW SWAP

In motivation example 1, we show the simple network update case of the flow swap, which is depicted in Fig. 3. In our example, we have four switches S1-S4 in our network, and each link has a capacity of 10 units. There are three flows F_1, F_2, and F_3 in the network, and they are colored by blue, green and red, respectively. Each flow is marked by the format (converged flow rate in this state: flow demand), as shown in Fig. 3. In this case, the goal of network update is to swap the paths which flow F_1 and F_3 go through, as shown in the initial state (Fig. 3a) and target state (Fig. 3b). It is obvious that the demands of F_1, F_2 and F_3 are all satisfied in both initial and target state.

When the controller wants to update the network from the state (a) to (b), there are some update plans it can choose. The first one is the simple one-shot mechanism. Due to the unsynchronized nature of network devices as introduced before, a network state where all three flows are on the path $S1 \rightarrow S2 \rightarrow S4$ may happen, as shown in Fig.3c. In this case, all flows will encounter serious congestion on the path (S1, S2, S4) at a transient state because the total flow rate of all three flows ($6 + 4 + 6 = 16$) is much greater than the capacity of each link (10).

This undesired result causes the network operators to prefer to introduce more determinacy, i.e., intermediate states, in the network update process. In this example, two flows F_1, F_3 swap the paths they go through. This introduces two possible intermediate states, i.e., all flows are either on the path (S1, S2, S4) or F_1 and F_3 are on the path (S1, S3, S4). Note that transient congestions happen in both intermediate states. If the network operator introduces the intermediate state where all flows are on the path (S1, S2, S4) as shown in Fig. 3c, all flows will encounter the most serious congestion as the one-shot mechanism. Further, because of the ultra-low latency in the data plane, all flows will converge quickly during the intermediate state. The resulted flow rates can be computed according to the max-min fairness principle, which is maintained by the congestion control algorithms



(a) Initial state (b) Target state (c) Intermediate state with serious congestion (d) Intermediate state with slight congestion

FIGURE 3. Motivation example 1.

at end-hosts. Therefore, the flow rates of F_1, F_2, F_3 will be converged to $10/3, 10/3, 10/3$, respectively. The result shows that all flows are throttled at the intermediate state, and their demands are all not satisfied.

On the other hand, if we migrate F_1 to the path (S1, S3, S4) firstly, the network congestion will happen on the path (S1, S3, S4). However, the extent of network congestion will be greatly reduced, as the total flow rates of F_1 and F_3 is 12 at the beginning of the intermediate network state. This slightly exceeds the link capacity by 2, which means the less possibility of packet loss and fewer chances to trigger the expensive link-level flow control mechanisms (priority flow control). Moreover, when the flows are converged in the intermediate state, the rates of F_1, F_2, and F_3 are 5, 4, and 5, respectively, which is shown in Fig. 3d. Indeed, even the F_1 and F_3 are throttled in the intermediate, the extent of demand violation is smaller than what is planned in Fig. 3c. Thus, a sophisticated network update mechanism should not only alleviate the transient congestion but also minimize the demand violation for all flows. *The key takeaway is that when the congestion is inevitable, we should alleviate the extent of the transient congestion during the network update process.*

2) THE CASE OF THE LINK FAILURE

In motivation example 2, we show the update plan when facing the link failure passively, which is depicted in Fig. 4. In this example, there is one source sending data to two sinks simultaneously, and there are five flows coexisting in the network, i.e., F_1, F_2, F_{31}, F_{32} and F_4. These flows are colored by blue, green, orange, orange and purple respectively, and the demands of these flows are 8, 6, 3, 3, 10. It is noteworthy that there are two same orange flows going through the path (source, S2, sink1), and the capacity of each link is 10. As shown in Fig. 4a, all five flows are satisfied in the initial state. Suddenly, a link (source, S4) is broken, then the network update is triggered to migrate the F_1 to the path (source, S1, sink1) in the target state, which is shown in Fig. 4(b).

If the network operator adopts the one-shot mechanism, serious congestion may happen on the link (source, S1). When the F_1 is migrated to its final path while the F_4 still

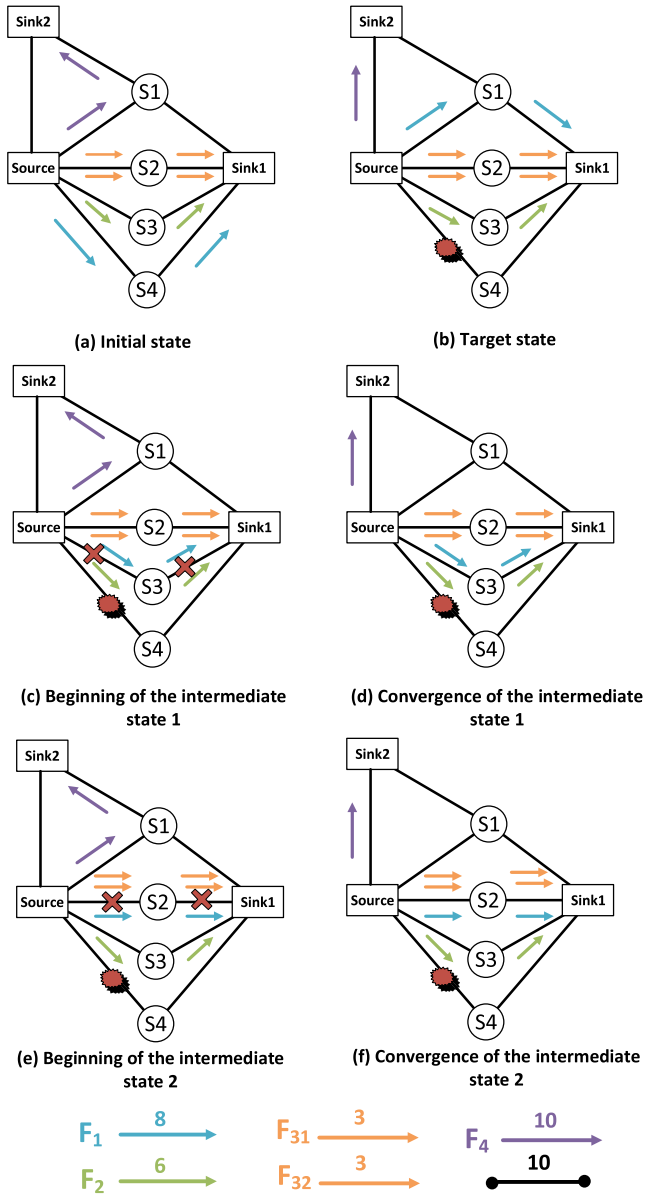


FIGURE 4. Motivation example 2.

remains on its initial path because of the unsynchronized nature of switches. In this case, the transient traffic on link

(source, S1) is $8 + 10 = 18$, which exceeds the link capacity by 8. Thus, the F_4 should be migrated to its final path before the F_1 is moved to the link (source, S1, sink) to avoid the serious network congestion, and the network operator has to firstly move the F_1 to other feasible paths to ensure the F_4 has been migrated, which also introduces intermediate network states to bring more determinacy.

However, the free capacities of the links (source, S2) and (source, S3) are both 4 in the initial state, which means the network congestion will happen when F_1 is migrated to either of these links. Moreover, the extent of network congestion will be the same, because the total transient traffic at the beginning of both intermediate states will be 14, which are shown in Fig. 4c and Fig. 4e. Then with the same transient congestion, there exist two update sequences, i.e., state $(a) \rightarrow (c) \rightarrow (b)$ or $(a) \rightarrow (e) \rightarrow (b)$.

If we adopt the update sequence state $(a) \rightarrow (c) \rightarrow (b)$, the converged rates of all five flows, $F_1, F_2, F_{31}, F_{32}, F_4$ in the intermediate state will be 5, 5, 3, 3, 10, respectively, which is shown in Fig. 4d. Moreover, F_1 and F_2 are throttled in the intermediate state, and the total demand violation ratio of all flows is $(6-5)/6 + (8-5)/8 = 13/24$. On the other hand, if the update sequence is state $(a) \rightarrow (e) \rightarrow (f)$, the converged rates of all five flows in the intermediate state will be 4, 6, 3, 3, 10, which is shown in Fig. 4f. Similarly, only F_1 is throttled, and the total demand violation ratio is $(8-4)/8 = 1/2$ during the whole update process, which is smaller than the previous plan. Therefore, the update sequence state $(a) \rightarrow (e) \rightarrow (f)$ is preferable in this example as it minimizes the demand violation ratio for all flows. *The key takeaway is that with controllable extent of congestion, we prefer the update with minimum demand violation during the whole update process.*

IV. DESIGN DETAILS

In this section, we formulate the MDVP as an integer linear programming (ILP) problem, and prove its hardness. Further, we propose a heuristic algorithm to approximate the optimal solution. The reason why we do not use the dependency graph to model the problem like [7] is that the over-subscription of bandwidth in our assumption makes it impossible for the graph construction.

A. PROBLEM FORMULATION

Based on the network model as described in § III-A, we will formulate the minimum demand violation problem (MDVP) as an integer linear programming problem in this section. Given the maximum congestion network operators allows, we aim to update the network from initial to target state while minimizing the demand violation ratio for all flows to minimize the negative impact.

Objective

$$minimize \quad \sum_{s \in S} \sum_{f \in F} \frac{d_f - r_f^s}{d_f} \tag{1}$$

Constraints

$$(1+\lambda)C_e \geq \sum_{f \in F} r_f^s \sum_{p \in P(f): e \in p} \max(x_{f,p}^s, x_{f,p}^{s+1}) \tag{1a}$$

$$r_f^{s+1} = \min_{p \in P(f): e \in p} \; \min_{e \in p} Alloc(G, F, x_f^s, d_f) \tag{1b}$$

$$r_f^s \geq 0 \tag{1c}$$

$$d_f \geq r_f^s \tag{1d}$$

$$\lambda \geq 0 \tag{1e}$$

$$x_{f,p}^s \in \{0,1\}, \quad \forall f \in F, \; \forall p \in P(f), \; \forall s \in \{2, 3, \ldots, n-1\} \tag{1f}$$

$$1 = \sum_{s=1}^{S} x_{f,p}^s \tag{1g}$$

We assume each flow f has its bandwidth demand d_f. When the flow f encounters network congestion, its performance degrades transiently at the beginning of the update. Further, its performance degrades consistently if its current rate is lower than its demand. The demand violation for a flow f at the state s can be quantified as the difference between its current rate r_f^s and its demand d_f. Therefore, we sum up the *demand violation ratio* for all flows F during the whole update process to quantify the extent of performance degradation, and the optimal objective (1) aims to minimize the total *demand violation ratio* to alleviate the performance degradation during the whole update process.

The constraint (1a) limits the maximum transient congestion a link e can accommodate at the state s. Due to the asynchronous nature of switches, we cannot determine the update sequences of different flows in the adjacent stages. Therefore, we use the sum of all possible flows walking through a link in two adjacent stages to represent the maximum traffic load that can be in a link, i.e., $r_f^s * max(x_f^s, x_f^{s+1})$. Note that the phrase, $r_f^s * max(x_f^s, x_f^{s+1})$, is not linear, which cannot be solved by the linear programming method, and we will discuss it in § IV-B. The parameter λ represents the maximum congestion extent the whole network allows, and by setting different λ the network operators can make a trade-off between update speed and network congestion. A larger λ means the flows can tolerate a larger scale network congestion, and this frees more space for designing the update plan, which is beneficial for introducing less intermediate states. For instance, if the λ is $+\infty$, this means the flows can ignore any congestion happening in the network, so the network operator can adopt the plan which introduces zero intermediate states, i.e., one-shot update plan.

The constraint (1b) will estimate the converged rate of the flow f at the state s based on its demand and the current network condition. Note that the converged rate of the flow f at the state s is the real-time flow rate at the beginning of state $s+1$, i.e., r_f^{s+1}. Because the inevitable congestion may occur in the network, the flow demand and rate mismatch, and it is necessary to estimate flow rate dynamically based on current network state rather than its original demand exclusively. Hence, the function $Alloc(G, F, r_f^s, d_f)$, assumes

the congestion control algorithm at end-hosts will drive the flow rates to converge to the ones conforming to the max-min fairness at the end of any network states. However, to compute the values which conform to the max-min fairness across flows is a classic and complex problem, and it involves a lot of non-linear computation. Thus, we need to approximate and relax the problem, and we will discuss it in § IV-B.

In this paper, we assume that there is only one kind of flows (unsplittable) in the network. If the flows are unsplittable, the variable, x_f^s, is binary, representing the flow f can only choose one of the feasible paths. Note that even if the splittable characteristic benefits from multi-path routing, the splittable flows introduce a number of out-of-order packets, which can hurt the network performance greatly. Thus we only assume single path routing in this paper, which is represented in the constraint (1f). The constraint (1g) means there is only one feasible routing for flow f, thus the $\sum_{s=1}^{S} x_{f,p}^s = 1$.

Finally, the constraint (1c) means any flow rate should be greater than 0. The constraint (1d) represents that a flow cannot grab more bandwidth than its demand. It is obvious that the congestion extent λ is greater than 0, which is shown in the constraint (1e).

B. TRANSFORMATION TO LINEARIZATION

Because of the *max* function, the constraint (1a) is not linear. By introducing the auxiliary variable $y_{f,p}^s$, we can transform the constraint (1a) to the following linear constraints. The auxiliary variable $y_{f,p}^s$ equals 1 when flow f is routed through path p either in state s or $s+1$, and equals 0 otherwise.

$$(1+\lambda)C_e \geq \sum_{f\in F} r_f^s \sum_{p\in P(f):e\in p} y_{f,p}^s \tag{2a}$$

$$y_{f,p}^s \geq x_{f,p}^s, \quad \forall f \in F \tag{2b}$$

$$y_{f,p}^s \geq x_{f,p}^{s+1}, \quad \forall f \in F \tag{2c}$$

The other non-linear constraint is the *Alloc* function in the constraint (1b). The *Alloc* function is responsible for allocating bandwidth among flows in a max-min fairness manner. When the set of flows are routed through the network without congestion, their demands are satisfied automatically, and equal their real-time flow rates. On the other hand, when a flow f routes through path p at stage s, transient congestion can occur along the path, and the demand will not be satisfied. In this situation, the rate of the flow f at state s will converge quickly in this stage.

Indeed, the computation of the *Alloc* function is similar to the well-known Max-Min Fair (MMF) multi-commodity flow problem [22], and there exist two kinds of approaches to solve this problem. The first one is the *progressive filling algorithm* [23]. The algorithm works as follows: 1) initially, all flow rates are set to 0; 2) then a small amount of bandwidth ϵ is added to all flows in a round-robin manner; 3) the algorithm continues to grow the rates of all flows until some flows are satisfied, and the algorithm will skip these flows in the next round bandwidth allocation; 4) finally, the algorithm keeps bandwidth allocation until all the bandwidth capacity is used or all flows are satisfied. However, this algorithm involves multiple rounds of bandwidth allocation, in each stage, it has to allocate bandwidth among flows, and it needs to exclude the flows who have already been satisfied. These functions make the *progressive filling algorithm* too complex to be modeled as the linear programming problem.

The other kind of approaches is to solve the Max-Min Fair (MMF) multi-commodity flow problem by a series of linear programming [3], [24]–[27]. However, this linear programming approach requires to solve a number of commodity linear programs, which becomes too time-consuming as the size of the problem increases. Moreover, we cannot replace the constraint (1b) with a series of linear programs, otherwise, it becomes a nested linear programming in the original problem, which is also too complex for the linear programming solver.

Therefore, we need to replace the constraint (1b) with an easy and standard one, which is shown in the following equation.

$$r_f^{s+1} = \min_{e\in E}\left(\frac{C_e}{\sum_{p\in P(f):e\in p} x_{f,p}^s}\right) \tag{3}$$

This simple equation means that the converged flow rate at state s is just determined by the number of flows coexisting on the same link, then its final rate is the minimum value out of all the links it goes through. Note that the *min* function can be translated to linear constraints similar to the way the *max* function does. The simple flow estimation is obviously not accurate, and we need to design an algorithm to approach the precise solution, which will be discussed in § IV-D.

C. HARDNESS ANALYSIS

Even after approximating the original constraints in the previous section, the MDVP problem is still a very hard problem. In this section, we will prove the hardness of MDVP problem by constructing the following case in Fig. 5.

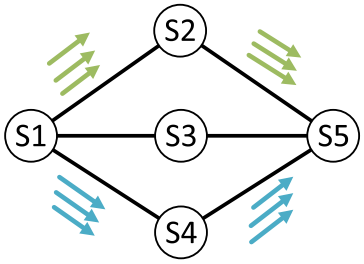


FIGURE 5. Reduction from set partition to MDVP problem.

Proof: Consider there are I green flows routed through the path $S1 \rightarrow S2 \rightarrow S5$, and each flow has the demand d_i, $i \in \{1, 2, 3, \ldots, I\}$. Similarly, there are J blue flows routed through the path $S1 \rightarrow S4 \rightarrow S5$, and each flow has the demand d_j, $j \in \{1, 2, 3, \ldots, J\}$.

The capacities of links, $(S1, S2)$, $(S2, S5)$, $(S1, S4)$, and $(S4, S5)$, are the same, and they equal the sum of the demands of the green flows, i.e., $C_{S1,S2} = C_{S2,S5} = C_{S1,S4} = C_{S4,S5} = \sum_{i=1}^{I} d_i = \sum_{j=1}^{J} d_j$. On the other hand, the capacities of links $(S1, S3)$ and $(S3, S5)$ are half of other links, i.e., $C_{S1,S3} = C_{S3,S5} = \sum_{i=1}^{I} d_i/2$.

Now we need to swap the paths where green and blue flows go through, and in the final state the blue flows go through the path $(S1, S2, S5)$, and the green flows go through the path $(S1, S4, S5)$. If all flows are sensitive to packet losses, the congestion tolerance λ in the constraint (2a) should be set to 0, then the network update from initial to final state is congestion-free. To minimize the intermediate states and speed up the update process, the update plan should make full use of the path $(S1, S3, S5)$. Therefore, the controller should firstly find the subset of green flows whose total demand approaches the capacity of the path $(S1, S3, S5)$ as closely as possible. The optimal situation is that finding a subset of green flows whose total demands equal the capacity of the path $(S1, S3, S5)$, then place these flows onto the path $(S1, S3, S5)$. After that, finding a subset of blue flows whose total demands equal the free capacity on the path $(S1, S2, S5)$, then swap the paths of these flows from the path $(S1, S4, S5)$ to the path $(S1, S2, S5)$. Later the update plan can continue in a similar logic.

However, finding a subset of flows whose total demands equal a specific value is the well-known set partition problem. The set partition problem has been proven that finding a subset of values that the sum of them equals a specific value is NP-Complete. Therefore, the MDVP problem is NP-Complete, and with the integer $x^s_{f,p}$, it is very difficult to find the appropriate routing plan to minimize the update process while controlling the congestion extent.

D. THE HEURISTIC APPROXIMATION ALGORITHM

We now design a heuristic approximation algorithm to tackle the original NP-Complete MDVP problem.

As we prove in § IV-C, the binary variable $x^s_{f,p}$ makes the MDVP problem NP-Complete. Thus, we can introduce the continuous variable $\widetilde{x^s_{f,p}}$ to replace $x^s_{f,p}$ to relax the original problem, and the continuous variable $\widetilde{x^s_{f,p}} \in [0, 1]$. Moreover, it is also noteworthy that flows can be throttled during the update process, thus we should change the constraint (1g) to $1 \geq \sum_{s=1}^{S} \widetilde{x^s_{f,p}}$. However, when we solve the relaxed MDVP problem, there will be multiple $\widetilde{x^s_{f,p}}$ that are greater than 0 in one stage, which means the flow f splits its traffic into multiple subflows, then these subflows are transmitted on different paths. Note that we do not allow multi-path transmission, and a flow f can only transmit data through one of its feasible paths. Therefore, if we have multiple $\widetilde{x^s_{f,p}}$ whose values are greater than 0 in a stage, we should select one of the feasible paths among these $\widetilde{x^s_{f,p}}$.

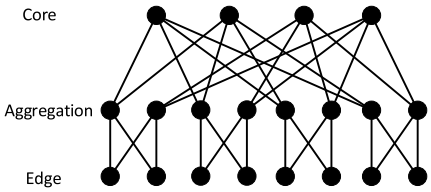
Algorithm 1 The Heuristic Approximation Algorithm

1: **Input:** the optimal fractional result $\widetilde{x^s_{f,p}}$, the estimated flow rate at each state $r^s_{f,p}$
2: **output:** the integer result $x^s_{f,p}$
3: **for** $s \leftarrow 1, 2, \ldots, S$ **do** ▷ S is the number of planned network states
4: **for** $f \leftarrow 1, 2, \ldots, N$ **do** ▷ N is the number of flows
5: **if** $\widetilde{x^s_{f,p}} == 1$ **then**
6: $x^s_{f,p} \leftarrow 1$
7: **else if** $\widetilde{x^s_{f,p}} > 0$ **then**
8: $\theta \leftarrow \infty$
9: $k \leftarrow p$
10: **for all** $r^s_{f,p}$ **do**
11: **if** $\theta > |r^s_{f,p} - d^s_f|$ **then**
12: $\theta \leftarrow |r^s_{f,p} - d^s_f|$
13: $k \leftarrow p$
14: **end if**
15: **end for**
16: $x^s_{f,k} \leftarrow 1$
17: **end if**
18: **end for**
19: **end for**

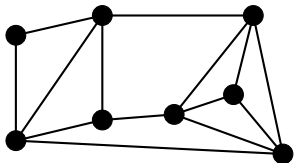
Now we will introduce our heuristic approximation algorithm, shown as Algorithm 1, to determine the routing for every flow in each network state. First, we should solve the relaxed MDVP problem introduced before, and use the multiple $\widetilde{x^s_{f,p}}$ whose values are greater than 0 as the input of our heuristic algorithm. The goal of this algorithm is to select one of the possible paths for each flow to approximate the optimal result which minimizes the demand violation ratio. As shown in Algorithm 1, for $\widetilde{x^s_{f,p}}$ is already integer (lines 5-6), we directly adopt this result as the final routing, because the integer means this value is already the feasible solution for flow f at state s. Otherwise, we use our heuristic algorithm to decide the final solution (lines 7-16). Recall that in § IV-A, we ignore the flow demand and set the converged flow rate equally across multiple flows, i.e., each flow equally shares the bandwidth capacity. This estimation may violate the max-min fairness due to the ignorance of flow demand, e.g., for a flow with small demand, the estimated flow rate is larger than what it actually consumes. Therefore, we should find the flow routing $x^s_{f,p}$ which makes the estimated flow rate $r^s_{f,p}$ as close as possible to its demand. This alleviates the overestimation or underestimation of flow at each state, and achieves a rather good performance which will be shown in § V.

V. EXPERIMENTAL EVALUATION

In this section, we first introduce the metrics and benchmarks for performance comparison. Then we will conduct



(a) A 4-pod fat-tree topology



(b) Microsoft's inter-DC topology

FIGURE 6. Network topologies used in our simulation.

extensive experiments to evaluate the overall performance and properties of our algorithm.

A. SIMULATION SETTINGS

We conduct our comparison experiments in two realistic topologies, and topologies are built using the FNSS [28].

- A 4-pod fat-tree for the DCN scenario as shown in Fig. 6. This topology has 3 layers and both the edge and aggregation layer have 4 switches in each pod. Each switch in the edge layer connects to hosts, and the number of hosts connecting to an edge switch is 4. The links in this topology have a bandwidth capacity of 10 Gbps, and each link latency is 1 us.
- A realistic inter-DC topology for interconnecting Microsoft's data centers as illustrated in Fig. 6b. The links have a capacity of 10 Gbps, and the link delay is 1 ms.

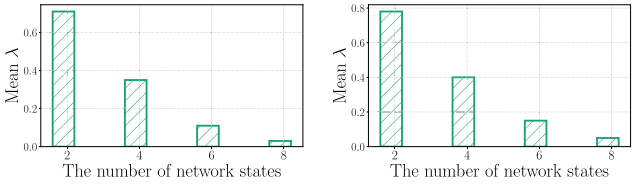
Now we introduce the benchmarks that will be compared and evaluated in our simulations.

- **One Shot:** Transition directly from the initial to the final state with no intermediate stages.
- **SWAN [3]:** State-of-the-art congestion-free update algorithm. This algorithm assigns the maximum number of states, then computes a congestion-free plan by solving a series of LP problems. We modify the basic SWAN algorithm to remove the remaining capacity margin to fully utilize link bandwidth.
- **MCUP [14]:** State-of-the-art update algorithm to minimize transient congestion instead of computing a congestion-free plan. This algorithm can guarantee an update plan in a given number of intermediate stages.
- **Heuristic:** The heuristic approximation algorithm as described in Algorithm 1.

B. PERFORMANCE EVALUATION

1) THE IMPACT OF PARAMETER λ

We show the impact of parameter λ in § IV-A and by changing the value of λ we can trade off the network congestion and the update speed.



(a) Fat-tree scenario (b) Microsoft's inter-DC scenario

FIGURE 7. The impact of parameter λ.

Setup: We generate 3000 flows in both the scenarios of fat-tree and inter-DC network. Specifically, the source and destination of flows are chosen arbitrarily, and their demands which are also randomly set should be guaranteed in the initial and final network states.

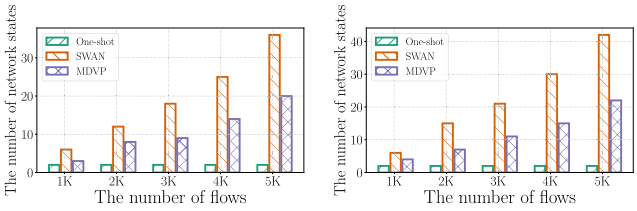
Results: Fig. 7a shows the values of the congestion extent, λ, can be set differently with a varied number of network states MDVP introduces. In Fig. 7a, when the number of network states is set to 2, which is the one-shot update, it is inevitable to cause serious network congestion. Thus, we have to set λ to 0.71, only then MDVP can produce a feasible update plan. As we introduce more intermediate network states, MDVP can compute the plan which alleviates network congestion greatly. For instance, in Fig.7a, when the number of states is 6, the maximum congestion extent λ can be greatly reduced to 0.11. Essentially, by introducing more intermediate states, MDVP can move more traffic to redundant paths in both scenarios temporarily to free more space for updating. This feature gives flexibility to network operators to compute the desired update plan based on the requirements of different kinds of network traffic. Moreover, Fig. 7b shows the similar trend as Fig. 7a, but introduces slightly more congestion. It is because there are fewer redundant paths in the inter-DC scenario compared to the fat-tree topology.

2) THE REDUCED NETWORK STATES

By increasing the number of flows, we manually exacerbate the congestion in the network. The MDVP shows its ability to speed up the update by introducing controllable network congestion.

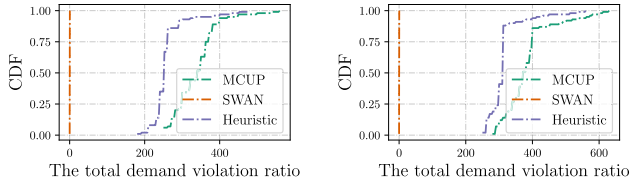
Setup: The source and destination of generated flows are chosen arbitrarily, and their demands are guaranteed in the initial and final network states. We set the parameter λ to 0.05, which introduces small congestion (5 % bandwidth oversubscription) in the network. Because MCUP should assign the number of intermediate states before network updates, in this experiment where we need to compare the varied number of network states, we omit the MCUP for a clear explanation.

Results: Fig. 8a shows with the increasing number of flows, MDVP can greatly reduce the number of network states, compared to SWAN. Specifically, when the number of flows is 3000 in the network, SWAN introduces 18 network states to compute an update plan. By contrast, MDVP only introduces 9 network states, which greatly reduce the



(a) Fat-tree scenario (b) Microsoft's inter-DC scenario

FIGURE 8. The number of states each method introduces.



(a) Fat-tree scenario (b) Microsoft's inter-DC scenario

FIGURE 9. The total demand violation ratio.

update time. This is because SWAN cannot tolerate network congestion at all, and it involves a large amount of computation to explore the possible paths to get a congestion-free update plan. On the contrary, by introducing slight congestion, MDVP does not need to explore much to find a congestion-free update plan, and can obtain a feasible plan in a short period. In addition, both Fig. 8a and Fig. 8b show a similar trend, and the gap between SWAN and MDVP becomes larger when the number of flow grows. Note that the one-shot method always consists of two network states, i.e., the initial and final states.

3) DEMAND VIOLATION RATIO

One of the main goals of this paper is to minimize the demand violation ratio during the network process. In this experiment, MDVP shows its ability to reduce the demand violation ratio for flows, thus disturbs as few flows as possible.

Setup: In this experiment, we generate 3000 flows, and choose their source and destination randomly as before. We study the demand violation ratio by deploying our algorithm, heuristic, and other benchmark algorithms, MCUP, SWAN, and we will evaluate all algorithms in both fat-tree topology and inter-DC topology. We set λ in MDVP to 0.08, and set the update states of MCUP introduces to 6 for a fair comparison. The one-shot mechanism is skipped because it only consists of the initial and final states, and introduces no throttled flows in the intermediate states.

Results: As Fig. 9a depicts, the demand violation ratio of our heuristic algorithm is much smaller than that of MCUP. Specifically, the mean value of total demand violation of our heuristic algorithm is 251. By contrast, the mean value of total demand violation of MCUP is 343, which is 36.7% larger than that of ours. This is because MCUP only concerns the maximum network congestion in the network, but ignores the demand violation of throttled flows in the intermediate states. As illustrated in § III, even the two update plans causing the same network congestion, the resulting demand violation ratio can be different. Our heuristic algorithm takes this into account, and computes the update plan that minimizes the demand violation ratio. A similar result is also shown in Fig. 9b. Note that SWAN will compute a congestion-free update plan, there is be no demand violation during the update process, which is shown in Fig. 9.

VI. CONCLUSION

In this paper, we present the minimum demand violation problem (MDVP), and use it to model the network update process. It firstly considers traffic variation in the current ultra-low latency data-center network, and it uses the max-min fairness across flows to estimate the converged rate of each flow at intermediate states. Our method brings flexibility for network operators to do a tradeoff between the network congestion and update speed. Finally, it improves the overall update performance by reducing the demand violation ratio by 36.7 % and halving the introduced intermediate states.

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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 323–334.

[3] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013.

[4] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "ZUpdate: Updating data center networks with zero loss," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 411–422, Aug. 2013.

[5] T. Mizrahi and Y. Moses, "Software defined networks: It's about time," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[6] G. Li, Y. Qian, C. Zhao, Y. R. Yang, and T. Yang, "DDP: Distributed network updates in SDN," in *Proc. 2018 IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1468–1473.

[7] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 539–550, Aug. 2014.

[8] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in SDN," in *Proc. Symp. SDN Res. (SOSR)*, 2017, pp. 21–33, doi: 10.1145/3050220.3050224.

[9] R. Gandhi, O. Rottenstreich, and X. Jin, "Catalyst: Unlocking the power of choice to speed up network updates," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2017, pp. 276–282, doi: 10.1145/3143361.3143397.

[10] S. Vissicchio and L. Cittadini, "FLIP the (flow) table: Fast lightweight policy-preserving SDN updates," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[11] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

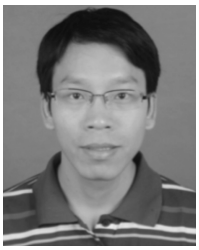
[12] S. Brandt, K.-T. Forster, and R. Wattenhofer, "On consistent migration of flows in SDNs," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[13] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10.

[14] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 1–10.
[15] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity Ethernet at scale," in *Proc. Conf. ACM SIGCOMM Conf.*, 2016, pp. 202–215, doi: 10.1145/2934872.2934908.
[16] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2019, pp. 44–58.
[17] D. Popescu, N. Zilberman, and A. Moore, "Characterizing the impact of network latency on cloud-based applications' performance," Tech. Rep. UCAM-CL-TR-914, 2017, doi: 10.17863/CAM.17588.
[18] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 523–536. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787484
[19] D. Firestone *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *Proc. 15th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 51–64. [Online]. Available: http://dl.acm.org/citation.cfm?id=3307441.3307446
[20] S. Wang, D. Li, and S. Xia, "The problems and solutions of network update in SDN: A survey," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2015, pp. 474–479.
[21] K. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1435–1461, 2nd Quart., 2019.
[22] D. Nace, L. Nhat Doan, O. Klopfenstein, and A. Bashllari, "Max-min fairness in multi-commodity flows," *Comput. Oper. Res.*, vol. 35, no. 2, pp. 557–573, Feb. 2008.
[23] D. Nace, M. Pioro, and L. Doan, "A tutorial on max-min fairness and its applications to routing, load-balancing and network design," in *Proc. 4th IEEE Int. Conf. Comput. Sci. Res., Innov. Vis. Future (RIVF)*, May 2006.
[24] D. Nace, N.-L. Doan, E. Gourdin, and B. Liau, "Computing optimal max–min fair resource allocation for elastic flows," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1272–1281, Dec. 2006.
[25] D. Nace and M. Pioro, "Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 5–17, 4th Quart., 2008.
[26] M. Pioro, P. Nilsson, E. Kubilinskas, and G. Fodor, "On efficient max-min fair routing algorithms," in *Proc. 8th IEEE Symp. Comput. Commun. (ISCC)*, Mar. 2004, pp. 365–372, vol. 1.
[27] E. Danna, S. Mandal, and A. Singh, "A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 846–854.
[28] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *Proc. 6th Int. Conf. Simulation Tools Techn.*, Brussels, Belgium, 2013.

CHENGYUAN HUANG received the B.Eng. degree in electronics engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2015, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology. His research interests are in the areas of data center networks and software-defined networking.

JIAO ZHANG received the B.S. degree from the School of Computer Science and Technology, Beijing University of Posts and Telecommunications (BUPT), in July 2008, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in July 2014. From August 2012 to August 2013, she was a Visiting Student with the Networking Group of ICSI, UC Berkeley. She is currently an Associate Professor with the School of Information and Communication Engineering and the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has (co)authored more than 20 international journal and conference papers. Her research interests include traffic management in data center networks, software-defined networking, network function virtualization, the future Internet architecture, and routing in wireless sensor networks.



TAO HUANG received the B.S. degree in communication engineering from Nankai University, Tianjin, China, in 2002, and the M.S. and Ph.D. degrees in communication and information systems from the Beijing University of Posts and Telecommunications, Beijing, China, in 2004 and 2007, respectively. He is currently a Professor with the Beijing University of Posts and Telecommunications. His current research interests include network architecture, routing and forwarding, and network virtualization.