Minimizing Transient Congestion during Network Update in Data Centers

Jiaqi Zheng*†, Hong Xu†, Guihai Chen*‡, Haipeng Dai*
*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210024, China
†Department of Computer Science, City University of Hong Kong, Hong Kong
‡Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China

***Abstract*—To maximize data center network utilization, the SDN control plane needs to frequently update the data plane as the network conditions change. Since each switch updates its flow table independently and asynchronously, the state transition—if done directly from the initial to the final stage—may result in serious flash congestion and packet loss. Prior work strives to find a *congestion-free* update plan with multiple stages, each with the property that there will be no congestion independent of the update order. Yet congestion-free update requires part of the link capacity to be left vacant and decreases utilization of the expensive network infrastructure. Further, it involves solving a series of LP, which is slow and does not scale well.**

In this paper, we study the more general problem of *minimizing* transient congestion during network update, given the number of intermediate stages. This exposes the tradeoff between update speed and transient congestion, and allows an operator to navigate a broader design space for performing network update. We formulate the minimum congestion update problem (MCUP) as an optimization program and prove its hardness. We propose an approximation algorithm and a greedy improvement algorithm to find the update sequence in an efficient and scalable manner. Extensive experiments with Mininet show that our solution reduces update time by 50% and saves control overhead by 30% compared to state of the art.

I. Introduction

In software defined networking (SDN), a logically centralized controller has a global view of the network state, and is responsible for delivering the control decisions to the data plane. The controller enforces policies by installing, modifying, or deleting forwarding rules in switch flow tables through southbound APIs such as Openflow [22]. SDN presents tremendous advantages for data center networks. Google and Microsoft build B4 [12] and SWAN [11], respectively, to interconnect their data centers and achieve higher network utilization, lower delay, and less packet loss.

Despite the centralization of control plane, data plane remains a distributed system. When network conditions change due to for example routing policy reconfiguration, switch upgrade, network failures, or traffic variations, the controller needs to update the data plane by modifying the flow tables so as to optimize performance. This process is not *atomic* [25]: each switch is updated independently and asynchronously. Thus network update may result in serious congestion during the transient period, even though the initial and final configurations are not. For example, congestion may happen when new flows—those that are supposed to be carried by a switch after the update—arrive before old ones that need to be migrated have left. Updates in both intra-datacenter and inter-datacenter networks, if not carefully planned, may disrupt many applications [11], [19].

Existing work, especially SWAN [11] and zUpdate [19], proposes to find a congestion-free update plan to solve this problem. The update plan consists of discrete stages, each of which involves changing flow tables of a set of switches, with the property that there will be no congestion independent of the update order or timing. This approach suffers from several drawbacks, however.

TABLE I
Running time for finding congestion-free update plans

	1K	2K	3K	4K	5K
DCN	0.73 min	1.40 min	2.10 min	2.96 min	4.12 min
WAN	0.60 min	1.01 min	1.57 min	2.43 min	3.12 min

For DCN, the topology is an 8-pod fat-tree, with 16 long-lived flows in the background. For WAN the topology is Microsoft's production network topology from [11]. The initial and final routing is generated randomly. We use the algorithms in [19] and [11] to find a congestion-free update plan for DCN and WAN, respectively, using LINGO as the solver, with different numbers of flows in the network.

First, to guarantee that a congestion-free update plan always exists, a portion (10%–50% [11]) of the network capacity has to be left vacant before update. This leads to reduced utilization of the expensive network infrastructure. Second, calculating a congestion-free update plan requires solving a series of LPs, which is too slow for production scale networks. Table I shows the running time of the algorithms in [11], [19] for data center networks (DCN) and inter-data center wide area networks (WAN) which connect geo-distributed data centers of the same operator. Note that a real DCN have more than 5K flows [16]. However not all of them need to be explicitly managed by the controller; only elephant flows are subject to explicit control of SDN and require controller intervention. In Table I, all flows in DCN refer to elephant flows. When the number of flows is larger than 2K, the running time in both scenarios is well beyond one minute. Thus congestion-free update is infeasible for operators like Google [12] and Microsoft [11] who perform centralized traffic engineering (TE) every five minutes to improve link utilization. In addition, slower update speed limits the controller's ability to react to failures and degrades application performance.

Instead of congestion-free update, in this paper we advocate to find an update plan that *minimizes* the transient congestion, given the number of intermediate stages within which the update needs to be done. We argue that, since switches are deeply buffered [10], and many low-priority data transfers tolerate packet loss especially in inter-data center WAN [15], it is worthwhile to explore solutions with a small extent of transient congestion. Our problem is more general in the sense that it allows the operator to navigate a broader design space, where one may trade off update speed, represented by the number of intermediate stages, for the extent of transient congestion. In previous work, the number of intermediate stages is not known until a update plan is found and cannot be adjusted [11], [19].

We make three novel contributions in this paper. First, we propose a general optimization framework for the minimum congestion update problem (MCUP) both in DCN and WAN. Generally speaking, the optimization program aims to determine routing for all ρ intermediate stages, where ρ is given, such that the maximum link utilization during the transition is minimized. We take into account single-path routing and multipath routing as constraints to meet the different application requirements, since applications such as video do not work well when their flows are split.

Our second contribution is a set of efficient algorithms to solve MCUP. We prove that MCUP is NP-hard, and thus focus on designing approximation algorithms. We first propose a randomized rounding algorithm and prove that it yields a $\mathcal{O}(\log k)$ upper bound of link congestion, where k is the number of switches. This algorithm only requires solving one LP and thus have faster running time than prior work. We further propose a greedy improvement algorithm which improves upon the rounding result by greedily rerouting each flow in each stage. The greedy algorithm has the same approximation ratio $\mathcal{O}(\log k)$ as the rounding algorithm in general topologies, and an approximation ratio of 4 for fat-tree in particular.

Our third contribution is a comprehensive performance evaluation of our algorithms in both DCN and WAN scenarios using production topologies. Simulation results show that our algorithms can reduce average link congestion by 60% and 67%, respectively, in both scenarios with 5 intermediate stages, and save 30% control overhead compared to prior work. We also develop a prototype of our algorithm on Mininet using the Floodlight controller. Experimental results show that our solution is 50% faster than prior work.

The remainder of the paper is organized as follows. We summarize related work in Sec. II. We give a formal definition of MCUP and analyze its hardness in Sec. III. In Sec. IV we present a rounding algorithm and prove its congestion upper bound. Based on the solution of the rounding algorithm, in Sec. V we propose the greedy improvement algorithm. Experimental evaluation and implementation are presented in Sec. VI and Sec. VII, and finally Sec. VIII concludes the paper.

II. Related Work

We briefly review prior art on network update in both traditional networks and SDN.

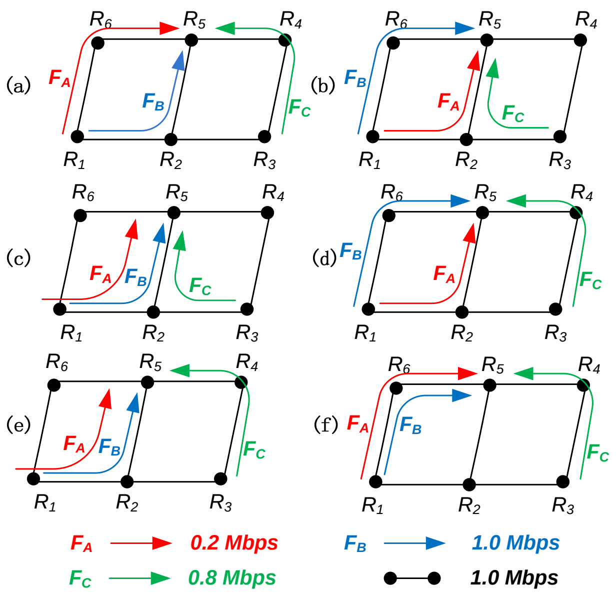


Fig. 1. Illustration of the network flow transition model.

In traditional networks, much work exists on avoiding transient misbehavior during network update. For example, consensus routing [14] considers the problem of eliminating transient state inconsistency. Vanbever et al. [26] focus on lossless migration and modification when moving from one routing protocol to another. Raza et al. [24] propose graceful network state migration, which strives to minimize the overall performance disruption by reassigning and maintaining link weights in the network.

Recent work starts to study network updates in SDN. Reitblatt et al. [25] propose a two-phase commit protocol that is guaranteed to preserve consistency when transitioning between configurations. Noyes et al. [23] propose a tool used for synthesizing network updates automatically and avoiding errors caused by manual configurations such as forwarding loops and access control violations. WayPoint [20], [21] aims to minimize the number of sequential controller interactions when directly transitioning from the initial to the final stage. SWAN [11] and zUpdate [19] try to find congestion-free update plans in WAN and DCN, respectively. Dionysus [13] employs dependency graphs to find a fast congestion-free update plan according to different runtime conditions of switches. CPC [6] studies a distributed SDN control plane that enables concurrent and robust policy implementation.

III. An Optimization Framework

We introduce our optimization framework for the minimum congestion update problem in this section.

A. A Motivating Example

In a software defined data center network, whenever the topology or traffic matrix changes, the controller needs to recalculate routing in order to optimize performance. Consider the example in Fig. 1, in which there are six switches $R_1, \ldots, R_6$, and the link capacity is 1 Mbps. F_A and F_B

are two flows from R_1 to R_5, whose demands are 0.2 Mbps and 1 Mbps, respectively. F_C is a flow from R_3 to R_5, whose demand is 0.8 Mbps. The initial routing is illustrated in Fig. 1(a). At this point, suppose a new flow appears from R_3 to R_4 with a demand of 1 Mbps. The controller then wants to change routing to Fig. 1(b). Due to the different update order, the three flows may be routed temporarily as in Fig. 1(c) during the transition. In this case congestion occurs at the link from R_2 to R_5, which is overloaded with twice its capacity, and results in severe packet loss.

Introducing intermediate stages can reduce transient congestion [11], [19]. For example, zUpdate [19] takes advantage of vacant capacity to find a congestion-free update plan. It first moves 80% of F_B onto path $\langle R_1, R_6, R_5\rangle$ and keeps the remaining 20%. Then F_A is moved from path $\langle R_1, R_2, R_5\rangle$ to $\langle R_1, R_6, R_5\rangle$. Finally the remaining 20% of F_B and all F_C is moved to their final paths. The whole process has three intermediate stages and involves solving three LPs, one for each stage.

Yet a congestion-free update plan may not always exist especially when the number of flows is large. Thus one has to set aside some capacity on each link to guarantee its existence, as proved in SWAN [11], and resource utilization is reduced. In this case, F_B with 1 Mbps demand cannot be completely satisfied through path $\langle R_1, R_2, R_5\rangle$. It must be split onto different paths. Flow splitting may not be feasible for certain applications that are sensitive to TCP packet reordering, such as video applications.

In contrast, we wish to find an update plan that minimizes the transient congestion. Assume F_A, F_B and F_C are three unsplittable flows. They should be routed only through a single path during update. If we consider the update plan in which routing is first changed from Fig. 1(a) to Fig. 1(d), and then to Fig. 1(b), transient congestion can be reduced significantly. Specifically, transitioning from Fig. 1(a) to Fig. 1(d) only causes the link capacity to exceed by 0.2, with two possible transient states shown in Fig. 1(e) and Fig. 1(f), and the transition from Fig. 1(d) to Fig. 1(b) is congestion-free. So the overall transient congestion is 0.2. This update plan may be acceptable in practice because switches have buffers to accommodate traffic bursts, and data center transports such as DCTCP can detect congestion early on with ECN to adjust sending rate in a fine granularity [5]. Further many applications, such as data processing frameworks, are elastic to bandwidth and can tolerate temporary rate reduction [18].

Moreover, the problem of finding an update plan that minimizes transient congestion given the number of intermediate stages is more general than finding a congestion-free plan. We expose the tradeoff between update speed and congestion, reducing the number of LPs that need to be solved and allowing the operator to speed up the update process. In the motivating example, there is only one intermediate stage which requires solving just one LP. We design fast heuristics to further improve the scalability of our solutions for large-scale networks.

TABLE II
KEY NOTATIONS IN THIS PAPER.

F_{sp}	The set of flows routed with single path routing
F_{mp}	The set of flows routed with multipath routing
F	The set of flows $F = F_{sp} \cup F_{mp}$
V	The set of switches v
E	The set of links e
G	The directed network graph $G = (V, E)$
S	The set of stages the routing update is performed
R_v	The capacity of flow table in switch v
C_e	The capacity of link e
$P(f)$	The set of possible paths for flow f
d^f	The demand of flow f
n	The number of update stages. $n = \lvert S\rvert$
ρ	The number of intermediate stages. $\rho = \lvert S\rvert - 2$
k	The number of switches in the network. $k = \lvert V\rvert$

B. Network Model

Before formulating the problem, we first present our network model. A network is a directed graph $G = (V, E)$, where V is the set of switches and E the set of links with capacities C_e for each link $e \in E$. According to application requirements, we divided flows in the network into two categories. F_{sp} represents the set of unsplittable flows that must use single path routing during update; F_{mp} represents the set of flows that can use multipath routing. Each flow f is associated with a demand d^f, routed through a possible path $p \in P(f)$ between its source and destination. The set of stages is $S = \{1, 2, \dots, n-1, n\}$, in which stage 1 and stage n are initial and final stage, respectively. Routing in stage 1 and stage n are known while routing in stages $2, 3, \dots, n-1$ need to be determined. The number of intermediate stages ρ is specified by the network operator. The operator may obtain this based on the history of update data and its global view of the network state, which however is beyond the scope of this paper. For convenience, we summarize important notations in Table II.

C. Problem Formulation

Based on the above network model, we formulate the minimum congestion update problem (MCUP) as a mixed integer linear program (1). Given the number of intermediate stages, we wish to find the optimal routing for all intermediate stages that minimizes the transient congestion from the initial stage to the final stage.

$$\text{minimize} \quad \max_{e \in E, s \in \{1,2,\dots,n-1\}} \mu_e^s \tag{1}$$

$$\text{subject to} \quad \sum_{f \in F_{sp} \cup F_{mp}} d^f \sum_{p \in P(f): e \in p} \max(x_{f,p}^s, x_{f,p}^{s+1}) \le \mu_e^s C_e,$$

$$\forall e \in E, \forall s \in \{1, 2, \dots, n-1\}, \tag{1a}$$

$$\sum_{p \in P(f)} x_{f,p}^s = 1,$$

$$\forall f \in F_{sp} \cup F_{mp}, \forall s \in \{2, 3, \dots, n-1\}, \tag{1b}$$

$$x_{f,p}^s \in \{0, 1\},$$

$$\forall f \in F_{sp}, \forall p \in P(f), \forall s \in \{2,3,\ldots,n-1\}, \quad \text{(1c)}$$

$$x^s_{f,p} \geq 0,$$
$$\forall f \in F_{mp}, \forall p \in P(f), \forall s \in \{2,3,\ldots,n-1\}, \quad \text{(1d)}$$

$$\mu^s_e > 0, \forall e \in E, \forall s \in \{1,2,\ldots,n-1\}. \quad \text{(1e)}$$

We define transient congestion as the maximum link congestion relative to its capacity μ^s_e during update across the network, as shown in the objective of MCUP (1). The optimization variables $x^s_{f,p}$ indicate whether flow f is routed through path p in stage s. Constraint (1a) characterizes transient congestion for individual links e during transition. For example, as illustrated in Fig. 1, during the transition from Fig. 1(a) to Fig. 1(b), *i.e.*, from stage 1 to stage 2, the maximum load of the link (R_2, R_5) is $0.2 \times \max(0,1) + 1.0 \times \max(1,0) + 0.8 \times \max(0,1) = 2$, which describes the case shown in Fig. 1(c). Constraint (1b) is the flow demand conservation constraint. Constraint (1c) represents the single path routing constraint for unsplittable flows, and (1d) represents the multipath routing constraint for splittable flows.

Because of the max function, constraint (1a) is not linear. By introducing auxiliary variables $\{y^s_{f,p}\}$ and $\{z^s_{f,p}\}$, we can transform (1) to the following mixed integer program with linear constraints. The auxiliary variable $y^s_{f,p}$ ($z^s_{f,p}$) equals one when unsplittable (splittable) flow f is routed through path p either in stage s or $s+1$, and equals zero otherwise.

$$\text{minimize} \quad \max_{e \in E, s \in \{1,2,\ldots,n-1\}} \mu^s_e \quad \text{(2)}$$

$$\text{subject to} \quad \sum_{f \in F_{sp}} d^f \sum_{p \in P(f): e \in p} y^s_{f,p} + \sum_{f \in F_{mp}} d^f \sum_{p \in P(f): e \in p} z^s_{f,p} \leq \mu^s_e C_e,$$
$$\forall e \in E, \forall s \in \{1,2,\ldots,n-1\}, \quad \text{(2a)}$$
$$y^s_{f,p} \geq x^s_{f,p}, \quad \forall f \in F_{sp}, \quad \text{(2b)}$$
$$y^s_{f,p} \geq x^{s+1}_{f,p}, \quad \forall f \in F_{sp}, \quad \text{(2c)}$$
$$z^s_{f,p} \geq x^s_{f,p}, \quad \forall f \in F_{mp}, \quad \text{(2d)}$$
$$z^s_{f,p} \geq x^{s+1}_{f,p}, \quad \forall f \in F_{mp}, \quad \text{(2e)}$$
$$\text{(1b)}, \text{(1c)}, \text{(1d)}, \text{(1e)}.$$

D. Hardness Analysis

We establish the hardness of MCUP below.

***Theorem** 1:* MCUP is NP-hard, even for a network consisting of two switches and $m+1$ parallel links.

Proof: Consider a special case of MCUP with only one intermediate stage. We construct a polynomial reduction from the set partition problem [7] to it. Consider a partition instance consisting of m items, each with a value s_i, $s_i \in \mathbb{R}, i \in \{1,2,\ldots,m\}$. For each item we introduce two flows F_i and F'_i with demands $d_{F_i} = d_{F'_i} = s_i$ and the instance of MCUP is constructed as shown in Fig. 2. There are $2m$ flows from source s to destination t in the initial stage, in which flows $F_1, F_2, ..., F_m$ are routed through links $e_1, e_2, \ldots, e_m$, respectively, and flows $F'_1, F'_2, \ldots, F'_m$ are routed through a single link e_0. The final stage is that flows $F'_1, F'_2, \ldots, F'_m$ are routed through links $e_1, e_2, ..., e_m$ and flows $F_1, F_2, ..., F_m$ are routed through link e_0. They cannot be split during update. The link capacities are $C_{e_i} = s_i$, and $C_{e_0} = \sum_{i=1}^m s_i$ for all $i \in \{1,2,\ldots,m\}$.

Therefore, any feasible partition of the items corresponds to MCUP with only one intermediate stage, and vice versa. The change of routing from the initial stage to the intermediate stage forms one set of the partition, and that from the intermediate stage to the final stage forms the other. ■

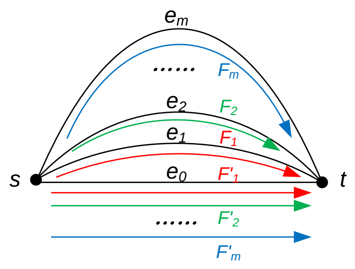


Fig. 2. Reduction from Partition to MCUP.

IV. A Rounding Algorithm

We now design a rounding based approximation algorithm [27] to tackle the NP-hard MCUP (2).

Algorithm 1 Randomized Rounding

Input: The optimal fractional solution $\{\tilde{x}^s_{f,p}\}$ to the relaxed LP of (2).
Output: A solution $\{\hat{x}^s_{f,p}\}$ to (1).

```
1:  for $s = 2$ to $n-1$ do
2:      for each $f \in F_{mp}$ do
3:          for each $p \in P(f)$ do
4:              $\hat{x}^s_{f,p} = \tilde{x}^s_{f,p}$
5:          end for
6:      end for
7:      for each $f \in F_{sp}$ do
8:          $P'(f) = \emptyset$
9:          for each $p \in P(f)$ and $p \notin P'(f)$ do
10:             $\hat{x}^s_{f,p} = 0$
11:             $P'(f) = P'(f) \cup p$
12:             $l^s_{f,p} = \sum_{p' \in P'(f)} \tilde{x}^s_{f,p'}$
13:         end for
14:         Generate a number $r$ in (0,1] uniformly at random
15:         Find $\hat{p}$ such that $r \leq l^s_{f,\hat{p}}$ and $l^s_{f,\hat{p}} - r$ is minimum
16:         $\hat{x}^s_{f,\hat{p}} = 1$
17:     end for
18: end for
```

The mixed integer program (2) can be relaxed to a linear program by replacing the constraint (1c) $x^s_{f,p} \in \{0,1\}$ with $x^s_{f,p} \geq 0$. Since constraint (1b) holds, $\{x^s_{f,p}\}$ are in fact real numbers between 0 to 1. The optimal fractional solutions $\{\tilde{x}^s_{f,p}\}$ of the relaxed LP of (2) can be obtained in polynomial time using standard solvers.

As shown in Algorithm 1, for $f \in F_{mp}$, $\{\tilde{x}^s_{f,p}\}$ is already the feasible solution (lines 2-6). For $f \in F_{sp}$, we apply randomized rounding to obtain an integer solution $\{\hat{x}^s_{f,p}\}$ (lines 7-17). We do not show the process of rounding auxiliary variable $\{\tilde{y}^s_{f,p|f\in F_{sp}}\}$, which can be readily obtained from the integer solutions $\{\hat{x}^s_{f,p|f\in F_{sp}}\}$. To ensure that only one path is chosen for a flow $f \in F_{sp}$ in stage s, the optimal fractional solution can be viewed as partitioning the interval $[0,1]$ to intervals of lengths $\{\tilde{x}^s_{f,p|f\in F_{sp}}\}$ (lines 9-13). A real number is generated uniformly at random in $(0,1]$ and the interval in which it lies determines the path (lines 14-16).

Before analyzing the performance of Algorithm 1, we introduce the following definition.

***Definition** 1:* Let μ^* be the optimal solution to (1), which gives a lower bound of transient congestion.

***Theorem** 2:* If $\mu^* > 1, \forall e \in E$, Algorithm 1 outputs a feasible solution with transient congestion bounded by $\mathcal{O}(\log k)\mu^*$ from any stage s to $s+1$ with probability $1-\frac{1}{k^2}$, where k is the number of switches in the network.

The proof can be found in Appendix A.

V. A Greedy Improvement Algorithm

In this section we develop a greedy improvement algorithm to improve the solution of the rounding algorithm. In spite of its guaranteed approximation ratio, the randomized algorithm is still not efficient as it may occasionally produce a bad solution. The greedy algorithm improves upon the solution of rounding by greedily rerouting each flow to a better path. It has the same approximation ratio as the rounding algorithm in general topologies, and has a constant approximation ratio of 4 in fat-tree topology.

A. Algorithm Design

Let us introduce two related notations first.

***Definition** 2:* **The** $\vee$ **operator**: Let $\{\alpha^{s_1}_{f,p}\}$ and $\{\beta^{s_2}_{f,p}\}$ be two routing configurations in stages s_1 and s_2. The result of $\{\alpha^{s_1}_{f,p}\} \vee \{\beta^{s_2}_{f,p}\}$ is a flow distribution $\{D_{f,e}\}$ in network G, where $D_{f,e} = \max(\{\alpha^{s_1}_{f,p}\}, \{\beta^{s_2}_{f,p}\})$. If flow $f \in F_{sp}$, $D_{f,e} \in \{0,1\}$. If flow $f \in F_{mp}$, $D_{f,e} \in [0,1]$.

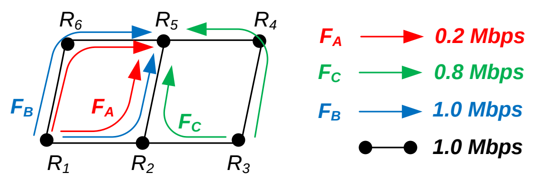


Fig. 3. The result of $\vee$ operator over routing configurations in Fig. 1(a) and Fig. 1(b).

The $\vee$ operator maps routing configurations of different stages onto the same network, which is convenient for further optimization. Fig. 3 shows the $\vee$ operator applied over routing configurations in Fig. 1(a) and Fig. 1(b).

The congestion calculation function ϕ is rigorously described in Algorithm 2. It takes the result of $\vee$ operator $\{D_{f,e}\}$ as input and calculates the maximal link congestion λ. η_e denotes the load and δ_e the load relative to its capacity in link e (lines 3-8). When Algorithm 2 stops, λ represents the maximal link congestion. Take flow distribution in Fig. 3 as

Algorithm 2 Congestion Calculation Function ϕ

```
Input: Flow distribution $\{D_{f,e}\}$.
Output: Maximal congestion $\lambda$.
 1: for each $e \in E$ do
 2:     $\eta_e, \delta_e = 0$
 3:     for each $f \in F$ do
 4:         if $D_{f,e} > 0$ then
 5:             $\eta_e = \eta_e + d^f \cdot D_{f,e}$
 6:         end if
 7:     end for
 8:     $\delta_e = \frac{\eta_e}{C_e}$
 9: end for
10: $\lambda = \arg\max_{e\in E} \delta_e$
```

the input of function ϕ, the result is 2.0, which represents the transient congestion transitioning from Fig. 1(a) to Fig. 1(b).

***Property** 1:* For any routing configuration $\{a_{f,p}\}$ in the network, $\phi(\{a_{f,p}\} \vee \{a_{f,p}\}) = \phi(\{a_{f,p}\})$.

***Property** 2:* Let $\{a_{f,p}\}$, $\{b_{f,p}\}$ and $\{c_{f,p}\}$ be three routing configurations in the same network. If $\phi(\{a_{f,p}\}) \geq \phi(\{b_{f,p}\})$, then $\phi(\{a_{f,p}\} \vee \{c_{f,p}\}) \geq \phi(\{b_{f,p}\} \vee \{c_{f,p}\})$.

***Property** 3:* Let $\{a_{f,p}\}$, $\{b_{f,p}\}$, $\{c_{f,p}\}$ and $\{d_{f,p}\}$ be four routing configurations in the same network. If $\phi(\{a_{f,p}\}) \geq \phi(\{b_{f,p}\})$ and $\phi(\{c_{f,p}\}) \geq \phi(\{d_{f,p}\})$, then $\phi(\{a_{f,p}\} \vee \{c_{f,p}\}) \geq \phi(\{b_{f,p}\} \vee \{d_{f,p}\})$.

***Theorem** 3:* Let $\{\alpha^1_{f,p}\}$ and $\{\gamma^n_{f,p}\}$ be the initial and final routing configurations. If there exists intermediate routing configurations $\{\beta^2_{f,p}\}, \{\beta^3_{f,p}\}, \ldots, \{\beta^{n-1}_{f,p}\}$ such that the value of $\phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\} \vee \ldots \vee \{\beta^{n-1}_{f,p}\} \vee \{\gamma^n_{f,p}\})$ is minimum, $\{\beta^2_{f,p}\}, \{\beta^3_{f,p}\}, \ldots, \{\beta^{n-1}_{f,p}\}$ are the optimal intermediate stages with the least transient congestion.

The proof can be found in Appendix B.

Algorithm 3 Greedy Improvement Algorithm

```
Input: The optimal fractional solution $\{\tilde{x}^s_{f,p}\}$ to the relaxed LP of
       (2).
Output: An optimized solution $\{\hat{x}^s_{f,p}\}$ to (1).
 1: Run Algorithm 1 and obtain a solution $\{\hat{x}^s_{f,p}\}$
 2: $\lambda = \phi\left(\vee_{s\in S}\{\hat{x}^s_{f,p}\}\right)$
 3: for $s^* = n-1$ to 2 do
 4:     $\{\beta^{s^*}_{f,p}\} = \{\hat{x}^{s^*}_{f,p}\}$
 5:     $\{D_{f,e}\} = \vee_{s\in S-\{s^*\}}\{\hat{x}^s_{f,p}\}$
 6:     for each $f^* \in F_{sp}$ do
 7:         $\{D^*_{f,e}\} = \{D_{f,e}\} \vee \{\beta^{s^*}_{F-\{f^*\},p}\}$
 8:         for each $p \in P(f^*)$ do
 9:             $\beta^{s^*}_{f^*,p} = 1$
10:             if $\phi\left(\{D^*_{f,e}\} \vee \beta^{s^*}_{f^*,p}\right) < \lambda$ then
11:                 $\{\beta^{s^*}_{f,p}\} = \{\beta^{s^*}_{F-\{f^*\},p}\} \vee \beta^{s^*}_{f^*,p}$
12:                 $\lambda = \phi\left(\{D^*_{f,e}\} \vee \beta^{s^*}_{f^*,p}\right)$
13:             end if
14:             if $\phi\left(\{D^*_{f,e}\} \vee \beta^{s^*}_{f^*,p}\right) = \lambda$ and $\beta^{s^*}_{f^*,p} = \hat{x}^{s^*+1}_{f^*,p}$ then
15:                 $\{\beta^{s^*}_{f,p}\} = \{\beta^{s^*}_{F-\{f^*\},p}\} \vee \beta^{s^*}_{f^*,p}$
16:             end if
17:         end for
18:     end for
19:     $\{\hat{x}^{s^*}_{f,p}\} = \{\beta^{s^*}_{f,p}\}$
20: end for
```

We are now ready to describe our greedy algorithm shown in Algorithm 3. We first run Algorithm 1 and obtain an initial solution $\{\hat{x}^{s}_{f,p}\}$ (line 1), which serves as input to the congestion calculation function ϕ (line 2). For simplicity, $\phi(\vee_{s\in S}\{\hat{x}^{s}_{f,p}\})$ represents $\phi(\{\hat{x}^{1}_{f,p}\} \vee \{\hat{x}^{2}_{f,p}\} \vee \ldots \vee \{\hat{x}^{n-1}_{f,p}\} \vee \{\hat{x}^{n}_{f,p}\})$, where $S = \{1, 2, \ldots, n\}$ (line 2). We consider intermediate stages $n-1$ to 2 and greedily change the routing configuration $\{\beta^{s^*}_{f,p}\}$ flow by flow to improve transient congestion from the initial stage $\{\hat{x}^{1}_{f,p}\}$ to the final stage $\{\hat{x}^{n}_{f,p}\}$ (lines 3-20). $\{\beta^{s^*}_{f,p}\}$ represent the routing of intermediate stage s^* (line 4). $\{D_{f,e}\}$ represent the result of $\vee$ operator applied over routing of intermediate stages $S - \{s^*\}$ (line 5). For each flow f^*, we first calculate $\{D^*_{f,e}\}$, which is the result of $\vee$ operator over routing in all stages except f^* in stage s^* (line 7). Then we move f^* onto a different potential path p in order to find a better routing (line 9). If the new routing $\beta^{s^*}_{f^*,p}$ for f^* results in less congestion, we update $\{\beta^{s^*}_{f,p}\}$ and λ (lines 10-13). Further, if f^* is routed through the final path p in stage s^*+1 and does not increase congestion, we update $\{\beta^{s^*}_{f,p}\}$ as well (lines 14-16). When all flows are rerouted in stage s^*, we update $\{\hat{x}^{s^*}_{f,p}\}$ and enter the next stage (line 19).

Note that the congestion upper bound of the rounding algorithm is $\mathcal{O}(\log k)$. The greedy algorithm improves upon it whenever possible. Thus its performance is at least as good as that of rounding. We have the following.

***Theorem** 4:* Algorithm 3 achieves an approximation ratio no more than that of Algorithm 1 in a general topology.

B. *Approximation Ratio for Fat-tree*

We now consider a particular DCN topology, fat-tree [4], an example of which is shown in Fig. 4(a). We analyze the approximation ratio of Algorithm 3 in a fat-tree.

***Definition** 3:* Let $\{\alpha^{1}_{f,p}\}$ and $\{\gamma^{n}_{f,p}\}$ be the initial and final routing. $\hat{\mu} = \phi(\{\alpha^{1}_{f,p}\} \vee \{\hat{\beta}^{2}_{f,p}\} \vee \ldots \vee \{\hat{\beta}^{n-1}_{f,p}\} \vee \{\gamma^{n}_{f,p}\})$, where $\{\hat{\beta}^{2}_{f,p}\}, \{\hat{\beta}^{3}_{f,p}\}, \ldots, \{\hat{\beta}^{n-1}_{f,p}\}$ are the optimal intermediate stages for a fat-tree.

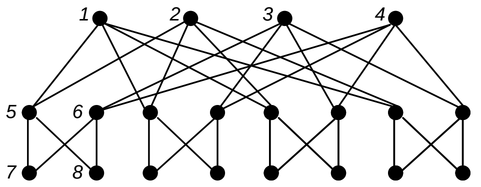
***Lemma** 1:* Independent of the rerouting order, when rerouting any flow f^* in stage s^* in Algorithm 2, if $\phi(\{D^*_{f,e|e\in p}\} \vee \beta^{s^*}_{f^*,p}) \geq 4\hat{\mu}$, there must exist another path p' such that $\phi(\{D^*_{f,e|e\in p'}\} \vee \beta^{s^*}_{f^*,p'}) < 4\hat{\mu}$, where $p, p' \in P(f^*)$ and $p \neq p'$.

The proof can be found in Appendix C. Now we can show that the greedy algorithm has a constant approximation ratio in fat-tree networks.

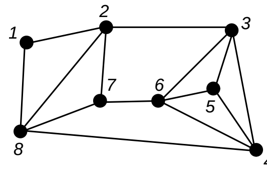
***Theorem** 5:* Algorithm 3 approximates MCUP in fat-tree networks with a factor of 4.

Proof: By Lemma 1, for any flow f^*, if $\phi(\{D^*_{f,e|e\in p}\} \vee \beta^{s^*}_{f^*,p}) \geq 4\hat{\mu}$, there must exist another path $p' \in P(f^*)$ such that $\phi(\{D^*_{f,e|e\in p'}\} \vee \beta^{s^*}_{f^*,p'}) < 4\hat{\mu}$. When all flows have been rerouted in all stages, there must exist intermediate routing configurations $\{\beta^{2}_{f,p}\}, \{\beta^{3}_{f,p}\}, \ldots, \{\beta^{n-1}_{f,p}\}$ such that $\phi(\{\alpha^{1}_{f,p}\} \vee \{\beta^{2}_{f,p}\} \vee \ldots \vee \{\beta^{n-1}_{f,p}\} \vee \{\gamma^{n}_{f,p}\}) < 4\hat{\mu} = 4 \cdot \phi(\{\alpha^{1}_{f,p}\} \vee \{\hat{\beta}^{2}_{f,p}\} \vee \ldots \vee \{\hat{\beta}^{n-1}_{f,p}\} \vee \{\gamma^{n}_{f,p}\})$. According to Property 2, Property 3 and Theorem 3, $\max(\phi(\{\alpha^{1}_{f,p}\} \vee \{\beta^{2}_{f,p}\}), \phi(\{\beta^{2}_{f,p}\} \vee \{\beta^{3}_{f,p}\}), \ldots, \phi(\{\beta^{n-1}_{f,p}\} \vee \{\gamma^{n}_{f,p}\})) \leq 4 \cdot \max(\phi(\{\alpha^{1}_{f,p}\} \vee \{\hat{\beta}^{2}_{f,p}\}), \phi(\{\hat{\beta}^{2}_{f,p}\} \vee \{\hat{\beta}^{3}_{f,p}\}), \ldots, \phi(\{\hat{\beta}^{n-1}_{f,p}\} \vee \{\gamma^{n}_{f,p}\})) = 4\mu^*$. Hence, when Algorithm 3 stops, the maximal transient congestion is less than or equal to $4\mu^*$.

■



(a) A 4-pod fat-tree DCN topology.



(b) Microsoft's inter-data center WAN topology.

Fig. 4. Realistic network topologies used in our evaluation.

VI. Experimental Evaluation

We conduct extensive experiments to evaluate our algorithms in this section.

A. *Setup*

We consider two realistic topologies.

- A 4-pod fat-tree for the DCN scenario as shown in Fig. 4(a). The edge and aggregation layer has 4 switches in each pod. Each edge switch connects to 2 hosts. The network has 16 hosts and 4 core switches. Each switch has 4 10 Gbps ports, resulting in a full bisection bandwidth network.
- A realistic WAN topology for interconnecting Microsoft's data centers [13], which is illustrated in Fig. 4(b). There are 8 switches and 14 10 Gbps links.

We consider both single path and multipath routing in the DCN scenario. For the WAN scenario, we consider tunnel based multipath routing [11]. For both settings, we leave 10% link capacity vacant on each link for SWAN. Flows in the network are generated randomly [2], and we change the flow demand to simulate traffic variations. We calculate the initial routing before the flow demand changes and final routing after the flow demand changes to maximize link utilization [8].

B. *Benchmark Schemes*

We evaluate the following schemes.

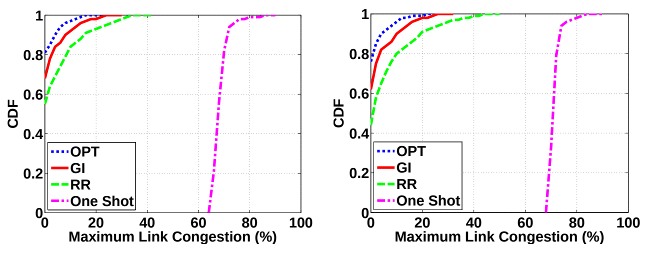
One Shot: Transition directly from the initial to the final stage.

RR: Our randomized routing algorithm as in Algorithm 1.

GI: Our greedy improvement algorithm as in Algorithm 3.

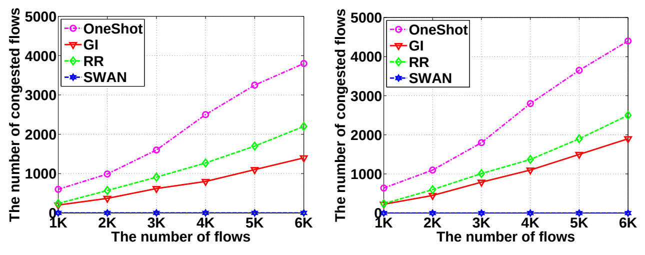
OPT: The optimal solution of the integer program (2) obtained using the branch and bound method.

SWAN: State-of-the-art congestion-free update algorithm [11]. As discussed in Sec. I, this heuristic algorithm



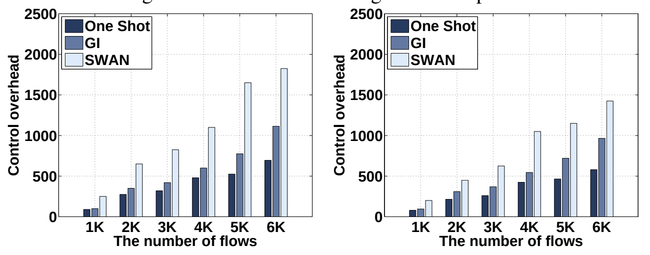
(a) DCN scenario (b) WAN scenario

Fig. 5. Maximum link congestion comparison.



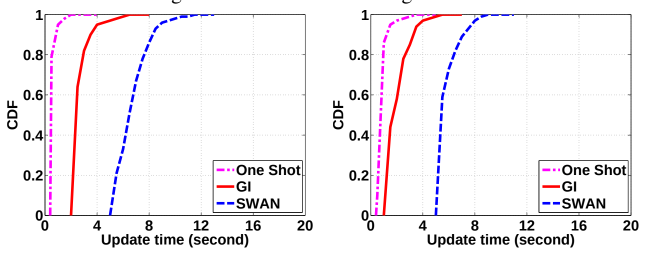
(a) DCN scenario (b) WAN scenario

Fig. 6. The number of congested flows.



(a) DCN scenario (b) WAN scenario

Fig. 7. Control overhead during update.



(a) DCN scenario (b) WAN scenario

Fig. 8. Update time.

works by iteratively solving a series of LPs until a congestion-free update plan is found, and cannot take the number of intermediate stages as input. Thus it cannot be used to solve (2), and we only include it for comparing the maximum number of rules and update time.

C. Basic Performance

We study the maximum link congestion generated by One Shot and the performance of our algorithms—RR and GI—in minimizing congestion comparing to OPT. Fig. 5(a) and Fig. 5(b) show the measured maximum link congestion, where the results of RR, GI and OPT are produced with 5 intermediate stages. Both GI and RR can effectively decrease congestion: GI and RR decrease link congestion by 67% and 60% respectively compared to One Shot. Furthermore, GI consistently outperforms RR by up to 12%, and provides near-optimal performance compared to OPT.

Fig. 6 shows the number of congested flows during the entire update process. We can see that, as the number of flows increases, One Shot yields significantly more congested flows compared to GI and RR. Specifically, in Fig. 6(a), the number of congested flows for One Shot, RR and GI is 1600, 910 and 620, respectively, in the DCN scenario when the number of flows is 3000. Looking more closely into Fig. 6(a) and Fig. 6(b), the improvement for GI in Fig. 6(a) is more significant: it decreases the number of congested flows by 20% from RR on average. This demonstrates that GI takes full advantage of the richly connected fat-tree topology and significantly mitigates congestion by rerouting flows onto less congested paths.

Fig. 7 shows the comparison of control overhead during update. We define control overhead as the number of rules that needs to be touched (added/removed/modified) during the update. Essentially this measures the number of operations, as well as the number of flow table entries required to perform the update. One Shot does not introduce intermediate stages and needs the least update operations. We also observe that SWAN induces more control overhead than GI. In Fig. 7(a), when the number of flows is 6000, the control overhead of SWAN is almost twice as that of GI. The reason is that SWAN usually takes more stages to transition the state without any congestion. In contrast, GI uses less intermediate stages with a small extent of congestion and saves a lot of update operations. Note that these results become inaccurate for switches that apply longest prefix matching or wild-card rules. However, such rules are increasingly being substituted with exact match rules in SDN [13].

VII. Implementation

Besides simulation, we develop a prototype of our algorithms using Mininet 2.0 [17]. We use Floodlight 0.9 controller [1] running on a PC with an Intel i5-2400 quad-core processor. Switches are emulated by Mininet and run Openflow v1.0. The forwarding rules are installed and updated via Floodlight's static flow pusher API.

We now describe how to perform network update using our algorithms in our implementation. The procedure is shown in Algorithm 4. We first obtain a solution to (1) using the greedy algorithm (line 1). Next we sequentially examine every stage s of the solution and determine what forwarding rules should be added to which switches by comparing the routing in stage s to its adjacent stage $s+1$. To ensure consistency, we adopt the two-phase commit protocol proposed in [25], which uses VLAN ID in packet headers to index stages. In the first phase of transition from s to $s+1$, new rules whose matching fields use the new VLAN ID corresponding to stage $s+1$ (lines 7-10) are added. During this phase, flows are still forwarded according to existing rules as packets are still stamped with the VLAN ID of stage s. Once the update is done for all switches, the protocol enters the second phase when we stamp

Algorithm 4 Performing Minimum Congestion Update

```
Input: Network topology $G = (V, E)$; the number of intermediate
    stage; initial routing $\{x^1_{f,p}\}$ and final routing $\{x^n_{f,p}\}$.
Output: Update sequence of switch rules.
 1: Apply Algorithm 3 and obtain solutions $\{\hat{x}^s_{f,p}\}$ to (1)
 2: for $s = 1$ to $n - 1$ do
 3:   $V' = \emptyset$
 4:   for each $f \in F$ do
 5:     for each $p \in P(f)$ do
 6:       if $\hat{x}^{s+1}_{f,p} \neq \hat{x}^s_{f,p}$ then
 7:         for each switch $v$ in path $p$ do
 8:           Add new rules to forwarding table or group table
              corresponding to flow $f$ in switch $v$. The new rules
              use new VLAN tag corresponding to stage $s + 1$
              to match packets.
 9:           $V' = V' \cup v$
10:         end for
11:       end if
12:     end for
13:   end for
14:   for each $v \in V'$ do
15:     if switch $v$ is connected by host then
16:       Modify the rules in switch $v$ such that it can stamp every
          incoming packet with a new VLAN tag corresponding
          to stage $s + 1$.
17:     end if
18:   end for
19: end for
```

every incoming packet with the new VLAN ID (lines 14-18). At this point the new rules become functional, and old rules are removed by the controller.

We conduct experiments of network update to handle device failures in both DCN and WAN. The topologies used here are the same as illustrated in Fig. 4(a) and Fig. 4(b). Link bandwidth is set to 10 Mb with 1 ms delay in Mininet. Port buffer size is 1 Mb. We use `iperf` to generate flows with an average size of 5 Mb. A flow's source and destination switches are chosen randomly. We use the `link down` command in Mininet to simulate link failures. We run our algorithms and SWAN, respectively, in the controller to update routing, and measure the total update time T. It includes two parts: time for generating an update plan T_{gen} and time for updating forwarding rules T_{update}. We measure T_{update} using OpenFlow arrier messages [3]. Specifically, from stages s to $s{+}1$, we first record the starting time T_s, then send the update messages, and finally send the barrier request message. Upon receiving the barrier response message, we obtain the finish time T_{s+1}. In addition, we measure the average delay between the controller and the switch T_{delay} using the `hello` messages [3], which is subtracted from the calculation.

$$T = T_{gen} + T_{update} = T_{gen} + \sum_{s=1}^{n-1} (T_{s+1} - T_s - T_{delay})$$

Fig. 8 shows the update time results in response to failures. In DCN, most updates using GI finish within three seconds while SWAN takes eight seconds. In WAN, GI uses three seconds while SWAN takes seven. One Shot, as the lower bound of update time, is also implemented in our experiments. T_{gen} is equal to zero for One Shot since it does not require solving LP. We observe that the update time of GI is close to One Shot especially in WAN scenario.

VIII. Conclusion

In this paper, we studied the problem of minimizing transient congestion during network update in data center networks. We formulated it as an integer linear program, and proposed two algorithms to solve the NP-hard problem. Experimental and simulation results show that our algorithms mitigates transient congestion and reduces update time significantly.

Acknowledgment

We thank the anonymous reviewers and our shepherd Lachlan Andrew for their helpful comments on drafts of this paper. The work is partly supported by HKRGC-ECS 21201714, CityU HK Start-up grant 7200366, China 973 projects (2014CB340303) and China NSF grants (61472252, 61133006, 61321491).

References

[1] Floodlight. http://floodlight.openflowhub.org/.

[2] Fnss. http://fnss.github.io/.

[3] Openflow switch specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, pages 63–74, 2008.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[6] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. A distributed and robust sdn control plane for transactional network updates. In *INFOCOM*, 2015.

[7] S. Chopra and M. R. Rao. The partition problem. *Math. Program.*, 59:87–115, 1993.

[8] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz. On the effect of forwarding table size on sdn network utilization. In *INFOCOM*, 2014.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[10] J. Gettys and K. M. Nichols. Bufferbloat: dark buffers in the internet. *Communication of the ACM*, 55(1):57–65, 2012.

[11] 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*, pages 15–26, 2013.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. In *SIGCOMM*, pages 3–14, 2013.

[13] X. Jin, H. H. Liu, X. Wu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *SIGCOMM*, pages 539–550, 2014.

[14] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. E. Anderson, and A. Venkataramani. Consensus routing: The internet as a distributed system. (best paper). In *NSDI*, pages 351–364, 2008.

[15] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula. Calendaring for wide area networks. In *SIGCOMM*, pages 515–526, 2014.

[16] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC*, pages 202–208, 2009.

[17] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets*, page 19, 2010.

[18] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *SIGCOMM*, 2011.

[19] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz. zupdate: updating data center networks with zero loss. In *SIGCOMM*, pages 411–422, 2013.

[20] A. Ludwig, J. Marcinkowski, and S. Schmid. Scheduling loop-free network updates: It's good to relax! In *PODC*, pages 13–22, 2015.
[21] A. Ludwig, M. Rost, D. Foucard, and S. Schmid. Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies. In *HotNets*, pages 1–7, 2014.
[22] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner. Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008.
[23] A. Noyes, T. Warszawski, P. Cerný, and N. Foster. Toward synthesis of network updates. In *SYNT*, pages 8–23, 2014.
[24] S. Raza, Y. Zhu, and C.-N. Chuah. Graceful network state migrations. *IEEE/ACM Trans. Netw.*, 19(4):1097–1110, 2011.
[25] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, pages 323–334, 2012.
[26] L. Vanbever, S. Vissicchio, C. Pelsser, P. François, and O. Bonaventure. Lossless migrations of link-state igps. *IEEE/ACM Trans. Netw.*, 20(6):1842–1855, 2012.
[27] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Appendix A
Proof of Theorem 2

Proof: To begin with, we present some related properties and definitions to facilitate the proof.

Property 4: If X and Y are nonnegative random variables, then $E[\max(X,Y)] \le E[X]+E[Y]$ holds [9].

Definition 4: Let $X^s_{f,e}$ be a binary random variable that indicates whether flow $f \in F_{sp}$ is routed through link e in stage s. If f passes through link e in stage s, $X^s_{f,e}=1$, otherwise $X^s_{f,e}=0$.

Definition 5: Let $z_e = \sum_{f\in F_{mp}} d^f \max(\tilde{x}^s_{f,e}, \tilde{x}^{s+1}_{f,e})$, which represents the maximum load on link e for $f \in F_{mp}$ during the transition from stage s to stage $s+1$.

Definition 6: Let $X_e = \sum_{f\in F_{sp}} d^f \max(X^s_{f,e}, X^{s+1}_{f,e}) + z_e$ be a random variable that indicates the maximum load on link e during the transition from stage s to stage $s+1$.

According to Property 4, Definition 6, and constraint (1a),

$$\begin{aligned}
E[X_e] &= \sum_{f\in F_{sp}} d^f E\left[\max\left(X^s_{f,e}, X^{s+1}_{f,e}\right)\right] + z_e \\
&\le \sum_{f\in F_{sp}} d^f \left(E\left[X^s_{f,e}\right] + E\left[X^{s+1}_{f,e}\right]\right) + z_e \\
&= \sum_{f\in F_{sp}} d^f \sum_{p\in P(f):e\in p} \left(E\left[X^s_{f,p}\right] + E\left[X^{s+1}_{f,p}\right]\right) + z_e \\
&= \sum_{f\in F_{sp}} d^f \sum_{p\in P(f):e\in p} \left(\tilde{x}^s_{f,p} + \tilde{x}^{s+1}_{f,p}\right) + z_e \\
&\le 2\mu^* C_e
\end{aligned}$$

Let $Y_e = \frac{X_e}{C_e} = \sum_{f\in F_{sp}} \frac{d^f}{C_e}\max(X^s_{f,e}, X^{s+1}_{f,e}) + \frac{z_e}{C_e}$. From above, $E[Y_e] \le 2\mu^*$. The random variables $\{X^s_{f,e}\}$ are mutually independent since link e for flow f is chosen independently in Algorithm 1. Therefore, Y_e, the sum of random variables $\{X^s_{f,e}\}$, is independent. Choose δ such that $(1+\delta) = \frac{8\ln k}{\ln\ln k}$ and apply Chernoff bound [27],

$$\Pr\left[Y_e \ge \frac{8\ln k}{\ln\ln k}2\mu^*\right] \le \left(\frac{8\ln k}{e\ln\ln k}\right)^{-8\frac{\ln k}{\ln\ln k}} \le \frac{1}{k^4}.$$

There are k switches in the network, so the number of links between nodes is at most k^2. By union bound,

$$\Pr\left[\max_{e\in E} Y_e \ge \frac{8\ln k}{\ln\ln k}2\mu^*\right] \le \sum_{e\in E}\Pr\left[Y_e \ge \frac{8\ln k}{\ln\ln k}2\mu^*\right] \le \frac{1}{k^2}.$$

■

Appendix B
Proof of Theorem 3

Proof: Suppose $\phi(\{\alpha^1_{f,p}\} \vee \{\hat\beta^2_{f,p}\} \vee \ldots \vee \{\hat\beta^{n-1}_{f,p}\} \vee \{\gamma^n_{f,p}\}) > \phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\} \vee \ldots \vee \{\beta^{n-1}_{f,p}\} \vee \{\gamma^n_{f,p}\})$ and $\{\hat\beta^2_{f,p}\}, \{\hat\beta^3_{f,p}\}, \ldots, \{\hat\beta^{n-1}_{f,p}\}$ is the optimal intermediate stage with the least transient congestion. Hence, we obtain $\max(\phi(\{\alpha^1_{f,p}\}\vee\{\beta^2_{f,p}\}), \phi(\{\beta^2_{f,p}\}\vee\{\beta^3_{f,p}\}), \ldots, \phi(\{\beta^{n-1}_{f,p}\}\vee\{\gamma^n_{f,p}\})) \ge \max(\phi(\{\alpha^1_{f,p}\} \vee \{\hat\beta^2_{f,p}\}), \phi(\{\hat\beta^2_{f,p}\} \vee \{\hat\beta^3_{f,p}\}), \ldots, \phi(\{\hat\beta^{n-1}_{f,p}\}\vee\{\gamma^n_{f,p}\}))$. Without loss of generality, we assume $\phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\})$ is the maximum value in the left side. We thus have $\phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\}) \ge \phi(\{\alpha^1_{f,p}\} \vee \{\hat\beta^2_{f,p}\}), \phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\}) \ge \phi(\{\hat\beta^2_{f,p}\} \vee \{\hat\beta^3_{f,p}\}), \ldots, \phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\}) \ge \phi(\{\hat\beta^{n-1}_{f,p}\} \vee \{\gamma^n_{f,p}\})$. From Property 1 and Property 3, $\phi(\{\alpha^1_{f,p}\} \vee \{\beta^2_{f,p}\}) \ge \phi(\{\alpha^1_{f,p}\} \vee \{\hat\beta^2_{f,p}\} \vee \ldots \vee \{\hat\beta^{n-1}_{f,p}\} \vee \{\gamma^n_{f,p}\}) > \phi(\{\alpha^1_{f,p}\}\vee\{\beta^2_{f,p}\}\vee\ldots\vee\{\beta^{n-1}_{f,p}\}\vee\{\gamma^n_{f,p}\})$. This contradiction concludes the proof, since we know that $\phi(\{\alpha^1_{f,p}\}\vee\{\beta^2_{f,p}\}) \le \phi(\{\alpha^1_{f,p}\}\vee\{\beta^2_{f,p}\}\vee\ldots\vee\{\beta^{n-1}_{f,p}\}\vee\{\gamma^n_{f,p}\})$ must hold. ■

Appendix C
Proof of Lemma 1

Proof: In a τ-pod fat-tree, each switch has τ ports. There are $\tau/2$ aggregation switches and $\tau/2$ edge switches in each pod. Core switches connect aggregation switches across pods. A 4-pod fat-tree topology is illustrated in Fig. 9. Aggregation and edge switches in pod $j \in \{1,2,\ldots,\tau\}$ are denoted by Agg^j_i and Tor^j_i, respectively, where $i \in \{1,2,\ldots,\tau/2\}$. Core switches, which connect aggregation switches, are denoted by Cor^i_u, where $u \in \{1,2,\ldots,\tau/2\}$.

Suppose there does not exist path p' such that $\phi(\{D^*_{f,e|e\in p'}\} \vee \beta^{s^*}_{f^*,p'}) < 4\hat\mu$. We denote source and destination switch of flow f^* by $Tor^{pod_i}_h$ and $Tor^{pod_j}_g$. Let E_h (E_g) be the set of links incident to switch $Tor^{pod_i}_h$ ($Tor^{pod_j}_g$). Let l_h (l_g) be the number of links in set E_h (E_g) such that $\phi(\{D^*_{f,e}\}) > 4\hat\mu$, where $e \in E_h(E_g)$. Let l'_h (l'_g) be the number of links in set E_h (E_g) such that $4\hat\mu - d^* \le \phi(\{D^*_{f,e}\}) \le 4\hat\mu$, where $e \in E_h(E_g)$ and $d^* = d^{f^*}/C_e$. We denote by F_h the total amount of flow in all links incident to switch $Tor^{pod_i}_h$, and thus $F_h > l_h\cdot 4\hat\mu\cdot C_e + l'_h(4\hat\mu - d^*)\cdot C_e \ge l_h \cdot 4\hat\mu\cdot C_e + l'_h(4\hat\mu - \hat\mu)\cdot C_e$. Hence,

$$F_h > 4l_h\hat\mu C_e + 3l'_h\hat\mu C_e \tag{3}$$

Now we consider the lower bound of $\hat\mu$, which is the case that the total amount of flow F_h are evenly splitted in $\tau/2$ links incident to $Tor^{pod_i}_h$.

$$\hat\mu C_e \ge \frac{2F_h}{\tau} \tag{4}$$

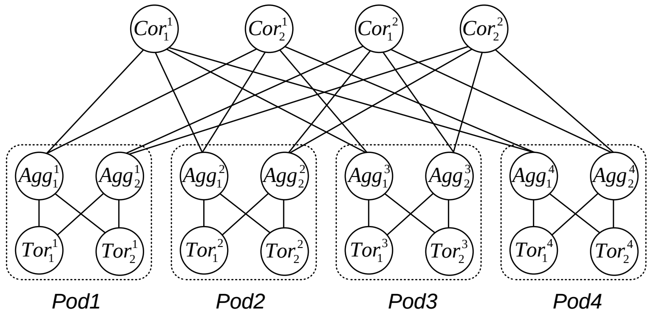


Fig. 9. 4-pod fat-tree topology

Combining (3) and (4), $\tau/2 > 4l_h + 3l'_h$, we obtain

$$\frac{\tau}{6} > \frac{4}{3}l_h + l'_h \tag{5}$$

We denote by l''_h (l''_g) the number of links in set E_h (E_g) such that $\phi(\{D^*_{f,e}\}) < 4\hat{\mu} - d^*$, where $e \in E_h(E_g)$. By inequality (5), we have

$$l''_h = \frac{\tau}{2} - l_h - l'_h = \frac{\tau}{2} - \frac{4}{3}l_h - l'_h + \frac{l_h}{3} > \frac{\tau}{3} + \frac{l_h}{3} \tag{6}$$

For the same reason, $\tau/6 > (4/3)l_g + l'_g$. Hence,

$$l''_g = \frac{\tau}{2} - l_g - l'_g > \frac{\tau}{3} + \frac{l_g}{3} \tag{7}$$

According to whether switch $Tor_h^{pod_i}$ and $Tor_g^{pod_j}$ are in the same pod, we prove Lemma 1 in cases.

Case 1: source switch $Tor_h^{pod_i}$ and destination switch $Tor_g^{pod_j}$ are in the same pod, *i.e.*, $i = j$.

From inequality (6) and (7), we obviously obtain,

$$l''_h > \frac{2}{3} \cdot \frac{\tau}{2}, \quad l''_g > \frac{2}{3} \cdot \frac{\tau}{2}$$

We observe that more than 2/3 links incident to $Tor_h^{pod_i}$ have load relative to its capacity less than $4\hat{\mu} - d^*$. And so also is $Tor_g^{pod_i}$. Moreover, The network topology in pod_i is a bipartite graph. Hence, there must exist a switch set A in aggregation layer and constitute a nonemply path set $P_{agg} = \{\langle Tor_h^{pod_i} Agg_j^{pod_i} Tor_g^{pod_i}\rangle\}$, where $Agg_j^{pod_i} \in A$ and $|A| > \tau/6$. Obviously, $\phi(\{D^*_{f,e|e\in p'}\}) < 4\hat{\mu} - d^*$, where $p' \in P_{agg}$. We further induce that there exists path $p' \in P_{agg}$ such that $\phi(\{D^*_{f,e|e\in p'}\} \vee \beta^{s_2}_{f^*,p'}) < 4\hat{\mu}$, which is contrary to the assumption.

Case 2: source switch $Tor_h^{pod_i}$ and destination switch $Tor_g^{pod_j}$ are in the different pods, *i.e.*, $i \neq j$.

Let E_c be the set of links incident to core switch Cor_u^γ. Let l_{cor} (l'_{cor}) be the number of links in set E_c such that $\phi(\{D^*_{f,e}\}) > 4\hat{\mu}$ ($4\hat{\mu} - d^* \leq \phi(\{D^*_{f,e}\}) \leq 4\hat{\mu}$), where $e \in E_c$. Let B be the set of pairs $(Agg_\gamma^{pod_i}, Agg_\gamma^{pod_j})$ in aggregation layer between pod_i and pod_j which are interconnected by core switch Cor_u^γ, such that both $\phi(\{D^*_{f,e'}\}) < 4\hat{\mu} - d^*$ and $\phi(\{D^*_{f,e''}\}) < 4\hat{\mu} - d^*$, where $e' \in \{\langle Tor_h^{pod_i} Agg_\gamma^{pod_i}\rangle\}$ and $e'' \in \{\langle Agg_\gamma^{pod_j} Tor_g^{pod_j}\rangle\}$. Let B' be the set of pairs $(Agg_\gamma^{pod_i}, Agg_\gamma^{pod_j})$ in aggregation layer between pod_i and pod_j such that $\phi(\{D^*_{f,e'}\}) \geq 4\hat{\mu} - d^*$ or $\phi(\{D^*_{f,e''}\}) \geq 4\hat{\mu} - d^*$, where $e' \in \{\langle Tor_h^{pod_i} Agg_\gamma^{pod_i}\rangle\}$ and $e'' \in \{\langle Agg_\gamma^{pod_j} Tor_g^{pod_j}\rangle\}$. Let $l = (l_h + l_g)/2$ and $l' = (l'_h + l'_g)/2$, we obtain

$$|B'| \leq \frac{l_h + l'_h + l_g + l'_g}{2} = l + l'$$

Note that if the assumption holds, the inequality $\phi(\{D^*_{f,e|e\in p}\}) \geq 4\hat{\mu} - d^*$ must be satisfied, where path $p \in \{\langle Agg_\gamma^{pod_i} Cor_u^\gamma Agg_\gamma^{pod_j}\rangle\}$ and aggregation switch pairs $(Agg_\gamma^{pod_i}, Agg_\gamma^{pod_j}) \in B$. Otherwise flow f^* can be routed through any path in set $P_{cor} = \{\langle Tor_h^{pod_i} Agg_\gamma^{pod_i} Cor_u^\gamma Agg_\gamma^{pod_j} Tor_h^{pod_j}\rangle\}$ and $\phi(\{D^*_{f,e|e\in P_{cor}}\}) < 4\hat{\mu} - d^*$. This is a contradiction. Hence, we must have

$$l_{cor} + l'_{cor} \geq \frac{\tau}{2} \cdot |B| = \frac{\tau}{2} \cdot \left(\frac{\tau}{2} - |B'|\right) \geq \frac{\tau}{2} \cdot \left(\frac{\tau}{2} - l - l'\right) \tag{8}$$

We denote by F_{pod_i} and F_{pod_j} the amount of flow originated from pod_i and directed to pod_j respectively. $F_{pod_i} + F_{pod_j} > l_{cor} \cdot 4\hat{\mu}C_e + l'_{cor}(4\hat{\mu} - d^*)C_e \geq l_{cor} \cdot 4\hat{\mu}C_e + l'_{cor}(4\hat{\mu} - \hat{\mu})C_e$, we obtain

$$F_{pod_i} + F_{pod_j} > 4l_{cor}\hat{\mu}C_e + 3l'_{cor}\hat{\mu}C_e \tag{9}$$

Consider the following lower bound for $\hat{\mu}$, where the amount of flow originated from pod_i and directed to pod_j are evenly splitted by $(\tau/2)(\tau/2)$ links respectively.

$$\hat{\mu}C_e \geq \frac{4F_{pod_i}}{\tau^2}, \hat{\mu}C_e \geq \frac{4F_{pod_j}}{\tau^2} \tag{10}$$

Combining (9) and (10), we have

$$\frac{\tau^2}{2} > 4 \cdot l_{cor} + 3 \cdot l'_{cor} \tag{11}$$

From (8) and (11), we have that

$$\frac{\tau^2}{2} > 3 \cdot (l_{cor} + l'_{cor}) + l_{cor} \geq 3 \cdot \frac{\tau}{2} \cdot (\frac{\tau}{2} - l - l') + l_{cor}$$

By (6) and (7), we derive that

$$\frac{\tau^2}{2} > 3 \cdot \frac{\tau}{2} \cdot (\frac{\tau}{3} + \frac{l}{3}) + l_{cor} > \frac{\tau^2}{2} + \frac{\tau \cdot l}{2} + l_{cor}$$

Hence,

$$0 > \frac{\tau \cdot l}{2} + l_{cor} \tag{12}$$

τ denotes the number of pods in fat-tree topology and thus $\tau > 0$. Recall the condition that $l \geq 0$ and $l_{cor} \geq 0$, we obtain $(\tau \cdot l)/2 + l_{cor} \geq 0$, which is a contradiction to (12). Hence, there exists path p' in fat-tree topology such that $\phi(\{D^*_{f,e|e\in p'}\} \vee \beta^{s_2}_{f^*,p'}) < 4\hat{\mu}$.

■