Real-time Update with Joint Optimization of Route Selection and Update Scheduling for SDNs

Hongli Xu[1] Zhuolong Yu[1] Xiang-Yang Li[1,3] Chen Qian[2] Liusheng Huang[1] Taeho Jung[3]
Email: xuhongli@ustc.edu.cn, yzl123@mail.ustc.edu.cn, xiangyang.li@gmail.com,
cqian12@ucsc.edu, lshuang@ustc.edu.cn, tjung@hawk.iit.edu
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Department of Computer Engineering, University of California at Santa Cruz
[3]Department of Computer Science, Illinois Institute of Technology

***Abstract*—Due to flow dynamics, a software defined network (SDN) may need to frequently update its data plane so as to optimize various performance objectives, such as load balancing. Most previous solutions first determine a new route configuration based on the current flow status, and then update the forwarding paths of existing flows. However, due to slow update operations of Ternary Content Addressable Memory (TCAM) based flow tables, unacceptable update delays may occur, especially in a large or frequently changed network. According to recent studies, most flows have short duration and the workload of the entire network may vary after a long duration. As a result, the new route configuration may be no longer efficient for the workload after the update, if the update duration takes too long. In this paper, we address the real-time route update, which jointly considers the optimization of flow route selection in the control plane and update scheduling in the data plane. We formulate the delay-satisfied route update (DSRU) problem, and prove its NP-Hardness. Two algorithms with bounded approximation factors are designed to solve this problem. We implement the proposed methods on our SDN testbed. The experimental results and extensive simulation results show that our method can reduce the route update delay by about 60% compared with previous route update methods while preserving a similar routing performance (with link load ratio increased less than 3%).**

***Index Terms*—Route Update, Software Defined Networks, Real-time, Load Balancing, Rounding.**

I. Introduction

Software-defined networking (SDN) is a new paradigm that separates the control and data planes on independent devices [1]. The controller provides centralized and flexible control by installing forwarding rules in the data plane, and the switches perform flow forwarding according to the rules. Due to frequent flow dynamics in a network [2], the data plane needs to be timely updated to avoid sub-optimal flow routes that may cause network congestion. The controller should respond to events such as shifts in traffic intensity, and new connection from hosts, by pushing forwarding rules to flow tables on the switches so as to achieve various performance requirements, such as load balancing. Thus, network updates (also called route updates) help to significantly improve network performance and resource utilization [3].

The speed of network updates is an important metric in many application scenarios because it determines the agility of the control loop. The effectiveness of network updates is tied to how quickly they adapt to changing workloads. Slow network updates will make network utilization lower and decrease the route performance, such as imbalanced link utilization [4]. The route update procedure consists of two main components: route selection in the control plane and forwarding table update in the data plane. Generally, the delay for route selection is much less than that for update scheduling [4]. Thus, this paper focuses on minimizing the delay for forwarding table update. Most existing update studies first compute a new route configuration *only* based on the current workload (or the collected flow intensity information) in a wide area network [4] or data center [5]. To minimize the update delay, these methods then schedule the data plane updates from the current route configuration to the new one with a fine-grained manner. For example, Hong *et al.* [4] divided all flows into minimum number of sets, and updated the flows in one set per round while avoiding the transient congestion. Jin *et al.* [3] encoded the consistency-related dependencies among updates at individual switches as a graph, and dynamically scheduled these updates on different switches.

However, the previous solutions [3] [4] do not pay attention to the impact of route selection in the control plane, including how many flows the controller will update and which path a flow will be updated to, on the update delay. Thus, they may still result in a long update duration, especially in a large and dynamic network. For example, in a moderate-size data center network, the volume of flows arriving at a switch can be in the order of $75K$-$100K$ flows/min for a rack consisting of 40 servers and the same would be $1{,}300K$ for the servers hosting around 15 virtual machines per host [6]. At one time instant, assume that it requires to update the routes of $8K$ flows (less than 1%) on one switch. The update delay depends on two main factors: the total number of rules that need to be updated or inserted, and TCAM's speeds for update operations (*e.g.*, insertion or modification). By testing on the today's commodity switches [3], it often takes about 5ms and 10ms for each insertion and modification, respectively, on the TCAM-based flow table. For the above scenario, assume that there needs $4K$ insertion operations and $4K$ modification operations. It will last for $60s$ at least for rule update on this switch.

Unfortunately, *a long update delay hurts the quality of route selection*. The existing studies [2] [6] have presented the traffic characteristics by testing in various datacenters. We can make several observations from existing measurement results [2]: 1)

978-1-5090-3281-5/16/$31.00 ©2016 IEEE

More than 80% flows last less than $10s$. 2) Fewer than 0.1% flows last longer than $200s$. 3) More than 50% bytes are in flows lasting less than $25s$. If a network-wide route update takes a long duration (*e.g.*, $60s$), the selected routes may not be useful because many flows have already terminated and many new flows have arrived [7]. As the optimal route configuration is usually derived by the current workload in a network [4], one route configuration is efficient for the current workload, but may be no longer efficient for another scheme after a long time duration. In other words, route updates with a shorter duration help to enhance the route performance. Therefore, real-time network update is necessary for an SDN.

In fact, both route selection and update scheduling determine the update delay. On the one hand, when the controller updates more flows, though the flow routes may be optimized, the update delay will be increased significantly. On the other hand, if only a few flows are updated, the update delay is smaller. So, there is a trade-off between route update delay and flow path optimization. Different from these previous works, *we will consider the performance trade-off by jointly optimizing the route selection and update scheduling*. To satisfy real-time requirements, we only update routes for a subset of chosen flows, including selecting new routes for these flows and scheduling the update operations. As a result, the final route configuration can still achieve a close-to-optimal route performance, such as load balancing, with update delay constraint. One may say that we only update the routes of those large flows (also called elephant flows) [5] [8]. However, our simulation results show that the update delay of this method is still unacceptable under many network situations, especially with a large number of (elephants) flows.

To address this challenge, we formulate the delay-satisfied route update (DSRU) problem, and prove its NP-Hardness by reduction from the classical multi-commodity flow (MCF) problem [9]. Due to its difficulty, we design two algorithms to solve the real-time route update challenge through *joint optimization of route selection and update scheduling*. One is an approximation algorithm based on the randomized rounding method, the other is a greedy algorithm. We show that the proposed algorithm can achieve the bi-criteria constant approximation performance under most network situations. We also implement our proposed route update algorithm on a real SDN platform. The experimental results on the platform and extensive simulation results show that our algorithm can significantly decrease the route update delay while achieving similar load balance. For example, our method decreases the route update delay by 60% compared with the previous method [3] while preserving a close route performance (with link load ratio increased less than 3%).

The rest of this paper is organized as follows. Section II introduces the preliminaries and problem definition. In Section III, we propose two algorithms to deal with the DSRU problem. The testing results and the simulation results are given in Section IV. Section V discusses the related works on the route update in an SDN. We conclude the paper in Section VI.

II. Preliminaries and Problem Formulation

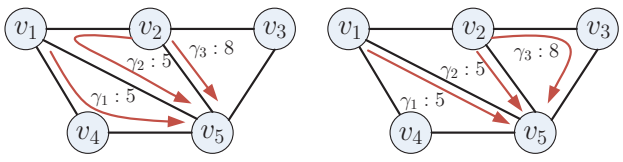
In this section, we will introduce the network and TCAM update models in an SDN, describe two update requirements for congestion freedom and route consistency, define the delay-satisfied route update problem, and prove its NP-Hardness.

A. Network Model

An SDN typically consists of two device sets: a controller, and a set of switches, $V = \{v_1, ..., v_n\}$, with $n = |V|$. Thus, The network topology can be modeled by $G = (V, E)$, where E is a set of links connecting switches. When a flow arrives at a switch, if there is a matched flow entry for the header packet, the switch takes the action specified in the entry, such as forwarding packets to a certain port. Otherwise, the switch reports the header packet to the controller. Then, the controller determines the route path for this flow, and deploys flow entries on all switches through this path. Same to many previous works [3] [10], we also adopt the unsplittable flow mode for its simplicity in this paper.

B. Delay Model for TCAM Updates

We introduce the delay model of TCAM updates, including insertion and modification, on a switch. Jin *et al.* [3] have shown that the delay for inserting/modifying flow entries is almost linear with the number of being inserted/modified flow entries if all flow entries have the same priority. In fact, most of flows have the unique priority in many practical applications [11]. Even though in some applications with microflow and macroflow rule schemes [12], there have two different priorities, the higher one for macroflows (or elephant flows) and the lower one for microflows. Since we only update some selective elephant flows, all of them have the unique priority. Thus, it is reasonable to assume that the operation delay for insertion/modification of each flow entry is a constant under the unique priority. Let t_i and t_m denote the required delays for the insertion and modification operations of a flow entry, respectively. For example, by testing on the practical commodity switches [3], t_m may be 10ms or more on some switches due to low-speed of TCAM updates. To be more practical, we will discuss how to deal with the various latency of TCAM updates in Section III-E.



(a) Current Route Configuration (b) Target Route Configuration

Fig. 1: An example of route update. Each link has 10 units of capacity. To avoid transient congestion, v_2 should apply the update for moving γ_3 before v_2 moves γ_2. Otherwise, link v_2v_5 will be congested.

C. Congestion-free Route updates [4]

We illustrate the downside of static ordering of rule updates with the example of Fig. 1. Each link has a capacity of 10 units and each flow size is marked. The controller wants to update

the network from (a) to (b) in Fig. 1. If we update all switches in one shot (*i.e.*, send all update commands simultaneously), since different switches will apply the updates at different times, such a strategy may cause congestion on some links. For instance, if v_2 applies the update for moving flow γ_3 after v_2 moves γ_2, link v_2v_5 will be congested. Thus, it requires us to carefully schedule the updates for congestion freedom.

D. Consistent Route updates [10]

When network updates occur, the consistency guarantee persists: each packet (or flow) is forwarded 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 [10]. This strong requirement is important for some applications such as HTTP load balancers, which need to ensure that all packets in the same TCP connection reach the same server replica to avoid breaking connections. Among many previous algorithms, the two-phase update mechanism [3] [10] has been widely used, because it can provide a simple, consistent and efficient route update way. Thus, our route update is also built on this method. To guarantee consistency, the two-phase update mechanism should satisfy the following constraint.

Definition 1 (Consistent Update Order): Given a flow, assume that its final path after the update is $v_0...v_m$, with $m \geq 1$. v_0 is the ingress switch, and others are the internal switches. The controller should start the route update on the ingress switch of this flow after the route updates are finished on all the internal switches. We call this as *consistent update order*.

Due to limited space, we omit the detailed description of this update mechanism here. The reader can refer [3] [10] for the detailed procedure of the two-phase update method.

E. Definition of Delay-Satisfied Route Update (DSRU)

This section defines the delay-satisfied route update (DSRU) problem. In an SDN, since the header packet of each new-arrival flow will be reported to the controller, the controller saves the information of each flow. Thus, it is reasonable to assume that we know the current flow set, denoted by $\Gamma = \{\gamma_1, ..., \gamma_r\}$ with $r = |\Gamma|$, in the network. After a flow entry is setup for flow γ on one switch, this switch can count the traffic size of this flow. By collecting the flow statistics information from switches, the controller knows the size (or intensity) of each flow γ as $s(\gamma)$. Note that, in some scenarios, the intensity of each flow may vary dynamically [11]. Thus, it is difficult to master the accurate intensity of each flow. We will discuss how to deal with the more practical case without accurate intensity information of each flow in Section III-E. Each flow γ will be assigned a set $\mathcal{P}_\gamma$ of feasible paths, and be routed through one feasible path in $\mathcal{P}_\gamma$. We will further discuss $\mathcal{P}_\gamma$ in the next section when we present our route update algorithm.

The route update procedure can be divided into route selection and update scheduling. To obtain the trade-off optimization among route performance and update delay, we should consider the joint optimization of route selection and update scheduling. Assume that the current route configuration is denoted by $\mathcal{R}^c$, in which the route of flow γ under this route configuration is denoted by $\mathcal{R}^c(\gamma)$. To support real-time update, we will determine a subset of flows, denoted by Γ^u, and select a feasible path as the target route of each flow. That is, the controller just updates the routes of flows in Γ^u. Assume that the target route configuration is denoted by $\mathcal{R}^f$, in which the route of flow γ is denoted by $\mathcal{R}^f(\gamma)$. The route update from $\mathcal{R}^c$ to $\mathcal{R}^f$ should satisfy the following three constraints:

- *The congestion-free constraint*: During the route update, there is no transient congestion in a network, illustrated in Fig. 1, by proper update scheduling.
- *The route-consistency constraint*: For each flow $\gamma \in \Gamma^u$, the consistent route update should be guaranteed. That is, the flow entry modification on the ingress switch should start after flow entries have been setup at all internal switches on the final route of this flow.
- *The real-time constraint*: The maximum delay of route update on all the switches should not exceed T_0, where T_0 is the tolerated delay.

After route update, we measure the traffic load on each link e as $l(e) = \sum_{\gamma \in \Gamma, e \in \mathcal{R}^f(\gamma)} s(\gamma) \leq \lambda \cdot c(e)$, where λ is the maximum link load factor. To provide more flexible routes for new-arrival flows, our objective is to achieve the load balancing in a network, by minimizing λ.

Theorem 1: The DSRU problem is NP-hard.

We can prove the NP-hardness by showing that the unsplittable multi-commodity flow (MCF) with minimum congestion problem [9] is a special case of DSRU. Due to limited space, we omit the detailed proof here.

III. Real-time Route Update Algorithm

Due to NP-hardness, it is difficult to optimally solve the DSRU problem. This section first gives a relaxed version of the DSRU problem, and explores the quantitative relationship between DSRU and its relaxed version (Section III-A). Then, we present an approximation algorithm, called RRSU, for the DSRU problem (Section III-B), and analyze the approximation performance of the proposed algorithm (Section III-C). We modify the RRSU algorithm so as to satisfy both the update delay and link capacity constraints (Section III-D). Finally, we give some discussion on our algorithm (Section III-E).

A. Preliminaries

Since the DSRU problem requires to ensure the congestion-free and consistent update constraints for all the flows, it makes problem formulation and algorithm design more difficult. Thus, we first consider a relaxed version of the DSRU problem, in which the controller sends all update commands and all switches execute the route updates of all flows simultaneously. This is also called Oneshot in [5]. Next, we formulate the relaxed version, called R-DSRU, into an integer linear program as follows. Let variable $y_\gamma^p \in \{0, 1\}$ denote whether the flow γ selects the feasible path $p \in \mathcal{P}_\gamma$ or not in the target route configuration $\mathcal{R}^f$. We use $t(v, \gamma, p)$ to express the necessary delay of flow-entry operation on switch v as the route of flow γ is updated to the target path p. Note that, if the route of flow γ does not change from the start configuration to the target one, *i.e.*, $\mathcal{R}^c(\gamma) = p$, $t(v, \gamma, p) = 0$. According to the two-phase update procedure [10], the constant $t(v, \gamma, p)$ is expressed as

follows: If switch v is the ingress switch of path p, v will take the modification operation for route update of flow γ, and $t(v,\gamma,p) = t_m$. If v is the internal switch of path p, this switch will take an insertion operation. So, $t(v,\gamma,p) = t_i$. Otherwise, $t(v,\gamma,p) = 0$. We have

$$t(v,\gamma,p) = \begin{cases} t_m, & v \text{ is the ingress switch of path } p(\neq \mathcal{R}^c(\gamma)) \\ t_i, & v \text{ is one of internal switches on } p(\neq \mathcal{R}^c(\gamma)) \\ 0, & otherwise. \end{cases} \tag{1}$$

Let variable λ be the maximum link load factor. R-DSRU solves the following problem:

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{p\in\mathcal{P}_\gamma} y_\gamma^p = 1, & \forall\gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{v\in p:p\in\mathcal{P}_\gamma} y_\gamma^p \cdot t(v,\gamma,p) \le T_0, & \forall v \in V \\ \sum_{\gamma\in\Gamma}\sum_{e\in p:p\in\mathcal{P}_\gamma} y_\gamma^p \cdot s(\gamma) \le \lambda \cdot c(e), & \forall e \in E \\ y_\gamma^p \in \{0,1\}, & \forall p,\gamma \end{cases} \tag{2}$$

The first set of equations means that each flow will be forwarded through one path from a source to a destination. The second set of inequalities denotes that the route update delay on each switch should not exceed a threshold, T_0 for the R-DSRU problem. That is, the real-time feature is pursued. The third set of inequalities expresses that the traffic load on each link e after update does not exceed $\lambda \cdot c(e)$, where λ is the maximum link load factor. Our objective is to achieve the load balance, *i.e.*, $\min\ \lambda$.

We use $\lambda^D(T)$ and $\lambda^R(T)$ to denote the optimal load-balance factors for the DSRU and R-DSRU problems under the delay constraint T, respectively. We have:

Lemma 2: For any delay constraint T, $\lambda^R(T) \le \lambda^D(T)$.

Proof: Assume that a set of flows Γ^u can be updated within a delay constraint T so as to achieve the minimum load-balance factor for the DSRU problem. Since it permits to update all the flows simultaneously, the total delay on each switch for R-DSRU should not exceed T if the routes for the same flow set are updated. In other words, we can at least update a flow set Γ^u within delay constraint T for the R-DSRU problem. As a result, the optimal load balance factor for the R-DSRU problem will not be worse than that for the DSRU problem under the same delay constraint. That is, $\lambda^R(T) \le \lambda^D(T)$, $\forall T$. ■

B. Rounding-based Route Selection and Update (RRSU)

We describe a rounding-based algorithm for real-time route update with consistency and congestion-free guarantee in an SDN. Due to difficulty of the DSRU problem, the first step obtains the fractional solution for the relaxed DSRU problem. In the second step, we choose one feasible path for each flow using the randomized rounding method, and obtain the target route configuration. Finally, we schedule the update operations of all the flows on different switches so as to guarantee the congestion-free and consistency. There might be an exponential number of feasible paths between a source and a destination for each flow. Following [4] [13], we assume that the controller has pre-computed a set of feasible paths between each pair of switches. Given a flow γ, we use $\mathcal{P}_\gamma$ to be the set of feasible paths between the two corresponding switches which connect to the source and the destination. These feasible paths may simply be the shortest paths, which can be found by depth-first search, between two switches.

Algorithm 1 RRSU: Rounding-based Route Selection-Update

1: **Step 1: Solving the Relaxed R-DSRU Problem**
2: Construct a linear program in Eq. (3) as Relaxed S-DSRU
3: Obtain the optimal solution $\widetilde{y_\gamma^p}$
4: **Step 2: Selecting Routes for Load-balance**
5: Derive an integer solution $\widehat{y_\gamma^p}$ by randomized rounding
6: **for** each flow $\gamma \in \Gamma$ **do**
7: **for** each feasible path $p \in \mathcal{P}_\gamma$ **do**
8: **if** $\widehat{y_\gamma^p} = 1$ **then**
9: Appoint a feasible path p for flow γ
10: **Step 3: Route Update Scheduling**
11: Apply the previous Dionysus method [3] for route update

To solve the problem formalized in Eq. (2), the algorithm constructs a linear program as a relaxation of the R-DSRU problem. More specifically, R-DSRU assumes that the traffic of each flow should be forwarded only through one feasible path. By relaxing this assumption, traffic of each flow γ is permitted to be splittable and forwarded through a path set $\mathcal{P}_\gamma$. We formulate the following linear program LP_1.

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{p\in\mathcal{P}_\gamma} y_\gamma^p = 1, & \forall\gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{v\in p:p\in\mathcal{P}_\gamma} y_\gamma^p \cdot t(v,\gamma,p) \le T_0, & \forall v \in V \\ \sum_{\gamma\in\Gamma}\sum_{e\in p:p\in\mathcal{P}_\gamma} y_\gamma^p \cdot s(\gamma) \le \lambda \cdot c(e), & \forall e \in E \\ y_\gamma^p \ge 0, & \forall p,\gamma \end{cases} \tag{3}$$

Note that, variable y_γ^p is fractional in Eq. (3). Since LP_1 is a linear program, we solve it in polynomial time with a linear program solver. Assume that the optimal solution for LP_1 is denoted by $\widetilde{y_\gamma^p}$, and the optimal result is denoted by $\widetilde{\lambda}$. As LP_1 is a relaxation of the R-DSRU problem, $\widetilde{\lambda}$ is a lower-bound result for R-DSRU. Using the randomized rounding method [14], variable $\widehat{y_\gamma^p}$, with $p \in \mathcal{P}_\gamma$, is set as 1 with the probability of $\widetilde{y_\gamma^p}$ while satisfying $\sum_{p\in\mathcal{P}_\gamma} \widehat{y_\gamma^p} = 1$, $\forall\gamma \in \Gamma$. If $\widehat{y_\gamma^p} = 1$, $\exists p \in \mathcal{P}_\gamma$, this means that flow γ selects $p \in \mathcal{P}_\gamma$ as its route.

After the second step, we have determined the target route configuration. The third step can just apply the previous Dionysus method [3] for consistent and congestion-free route update. The RRSU algorithm is given in Alg. 1.

C. Approximation Performance Analysis

We analyze the approximate performance of the proposed RRSU algorithm. Assume that the minimum capacity of all the links is denoted by $c_{\min}$. We define a variable α as follows:

$$\alpha = \min\{\min\{\frac{\widetilde{\lambda} c_{\min}}{s(f)}, f \in \Gamma\}, \frac{T_0}{t_m}\} \tag{4}$$

In most practical situations, since the flow intensity is usually much less than the link capacity, for example, $c_{\min}$ =100Mbps,

and $s(f)$ =4Mbps for high definition video, and t_m is usually much less than T_0, it follows that $\alpha \gg 1$. Since RRSU is a randomized algorithm, we compute the expected traffic load on links and the expected update delay on switches. We give two famous lemmas for probability analysis.

Lemma 3 (Chernoff Bound): Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then, $\mathbf{Pr}\left[\sum_{i=1}^{n} x_i \geq (1+\epsilon)\mu\right] \leq e^{\frac{-\epsilon^2\mu}{2+\epsilon}}$, where ϵ is an arbitrarily positive value.

Lemma 4 (Union Bound): Given a countable set of n events: $A_1, A_2, ..., A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup ... \cup A_n) \leq \sum_{i=1}^{n} \mathbf{Pr}(A_i)$.

Link Capacity Constraints. We first bound the probability with which the capacity of each link will be violated after route update. The first step of the RRSU algorithm will derive a fractional solution $\widetilde{y_\gamma^p}$ and an optimal result $\widetilde{\lambda}$ for the relaxed R-DSRU problem by the linear program. Using the randomized rounding method, for each flow $\gamma \in \Gamma$, only one path in $\mathcal{P}_\gamma$ will be chosen as its target route. Thus, the traffic load of link e from flow γ is defined as a random variable $x_{e,\gamma}$ as follows:

Definition 2: For each link $e \in E$ and each flow $\gamma \in \Gamma$, a random variable $x_{e,\gamma}$ is defined as:

$$x_{e,\gamma} = \begin{cases} s(\gamma), & \textit{with probability of } \sum_{e\in p: p\in\mathcal{P}_\gamma} \widetilde{y_\gamma^p} \\ 0, & otherwise. \end{cases} \tag{5}$$

According to the definition, x_{e,γ_1}, x_{e,γ_2}... are mutually independent. The expected traffic load on link e is:

$$\mathbb{E}\left[\sum_{\gamma\in\Gamma} x_{e,\gamma}\right] = \sum_{\gamma\in\Gamma} \mathbb{E}[x_{e,\gamma}] = \sum_{\gamma\in\Gamma} \sum_{e\in p: p\in\mathcal{P}_\gamma} \widetilde{y_\gamma^p} \cdot s(\gamma) \leq \widetilde{\lambda} c(e) \tag{6}$$

Combining Eq. (6) and the definition of α in Eq. (4), we have

$$\begin{cases} \frac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda} c(e)} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\right] \leq \alpha. \end{cases} \tag{7}$$

Then, by applying Lemma 3, assume that ρ is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\alpha\right] \leq e^{\frac{-\rho^2\alpha}{2+\rho}} \tag{8}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\right] \leq e^{\frac{-\rho^2\alpha}{2+\rho}} \leq \frac{\mathcal{F}}{n^2} \tag{9}$$

where $\mathcal{F}$ is the function of network-related variables (such as the number of switches n, *etc.*) and $\mathcal{F} \to 0$ when the network size grows.

The solution for Eq. (9) is expressed as:

$$\rho \geq \frac{\log\frac{n^2}{\mathcal{F}} + \sqrt{\log^2\frac{n^2}{\mathcal{F}} + 8\alpha\log\frac{n^2}{\mathcal{F}}}}{2\alpha}, \quad n \geq 2 \tag{10}$$

We give the approximation performance as follows.

Theorem 5: The proposed RRSU algorithm achieves the approximation factor of $\frac{4\log n}{\alpha}+3$ for link capacity constraints.

Proof: Set $\mathcal{F} = \frac{1}{n^2}$. Eq. (9) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\right] \leq \frac{1}{n^4}, \quad \text{where} \quad \rho \geq \frac{4\log n}{\alpha} + 2 \tag{11}$$

By applying Lemma 4, we have,

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{e\in E}\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\right] \\ &\leq \sum_{e\in E} \mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\right] \\ &\leq n^2 \cdot \frac{1}{n^4} = \frac{1}{n^2}, \quad \rho \geq \frac{4\log n}{\alpha} + 2 \end{aligned} \tag{12}$$

Note that the third inequality holds, because there are at most n^2 links in a network with n switches. The approximation factor of our algorithm is $\rho + 1 = \frac{4\log n}{\alpha} + 3$. ■

Route Update Delay Constraints. Next, we analyze the route update delay performance on most situations. Similar to definition 2, we define another random variable $z_{v,\gamma}$ to formulate the route update delay on a switch.

Definition 3: For each switch $v \in V$ and each flow $\gamma \in \Gamma^u$, random variable $z_{v,\gamma}$ is defined as:

$$z_{v,\gamma} = \begin{cases} t_m, & v \textit{ is the ingress switch of flow } \gamma \\ t_i, & \textit{with probability } \sum_{v\in p: p\in\mathcal{P}_\gamma} \widetilde{y_\gamma^p} \\ 0, & otherwise \end{cases} \tag{13}$$

By the definition, $z_{v,\gamma_1}, z_{v,\gamma_2}$... are mutually independent. Thus, we have:

$$\begin{cases} \frac{z_{v,\gamma}\cdot\alpha}{T_0} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma^u} \frac{z_{v,\gamma}\cdot\alpha}{T_0}\right] \leq \alpha. \end{cases} \tag{14}$$

Similar to the proof for the link capacity constraint, we have a general form of proving: the total delay for route update on any switch v will not exceed the delay constraint T_0 by a factor of $1+x$, if and only if there exists $\mathcal{F}$ satisfying

$$\mathbf{Pr}\left[\bigvee_{v\in V}\sum_{\gamma\in\Gamma^u} \frac{z_{v,\gamma}\cdot\alpha}{T_0} \geq (1+x)\cdot\alpha\right] \leq \mathcal{F}, \tag{15}$$

where x is the function of network-related variables (such as n, *etc.*), and $\mathcal{F}$ is defined same as that in Eq. (9). By applying Theorem 4, Eq. (15) is relaxed to:

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{v\in V}\sum_{\gamma\in\Gamma^u} \frac{z_{v,\gamma}\cdot\alpha}{T_0} \geq (1+x)\cdot\alpha\right] \\ &\leq \sum_{v\in V} \mathbf{Pr}\left[\sum_{\gamma\in\Gamma^u} \frac{z_{\gamma,v}\cdot\alpha}{T_0} \geq (1+x)\cdot\alpha\right] \leq \mathcal{F} \end{aligned} \tag{16}$$

Let n be the number of switches. By applying Theorem 3 and Eq. (14), we can assume:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma^u}\frac{z_{v,\gamma}\cdot\alpha}{T_0}\geq(1+x)\cdot\alpha\right]\leq e^{-\frac{x^2\alpha}{2+x}}\leq\frac{\mathcal{F}}{n},\quad n\geq 2 \tag{17}$$

Then, we get the result:

$$x\geq\frac{\log\frac{n}{\mathcal{F}}+\sqrt{\log^2\frac{n}{\mathcal{F}}+8\cdot\alpha\log\frac{n}{\mathcal{F}}}}{2\cdot\alpha},\quad n\geq 2 \tag{18}$$

By setting suitable values of parameters x and $\mathcal{F}$, we can derive the approximation performance on the route update delay constraint.

Lemma 6: After the rounding process, the total route update delay on any switch v will not exceed the constraint T_0 by a factor of $\frac{3\log n}{\alpha}+3$ for the R-DSRU problem.

Proof: Set $\mathcal{F}(n)=\frac{1}{n^2}$. Apparently $\mathcal{F}\to 0$ as $n\to\infty$. With respect to Eq. (18), we set

$$x=\frac{\log\frac{n}{\mathcal{F}}+\log\frac{n}{\mathcal{F}}+4\cdot\alpha}{2\cdot\alpha}=\frac{6\log n+4\cdot\alpha}{2\cdot\alpha}=\frac{3\log n}{\alpha}+2 \tag{19}$$

Then Eq. (15) is guaranteed with $1+x=\frac{3\log n}{\alpha}+3$ and $\mathcal{F}=\frac{1}{n^2}$, which concludes the proof. ■

Approximation Factor: To forward all the flows on chosen paths, the above analysis shows that, the link capacity will hardly be violated by a factor of $\frac{4\log n}{\alpha}+3$, and the route update delay constraint will not be violated by a factor of $\frac{3\log n}{\alpha}+3$ for the R-DSRU problem. For simplicity, we use F_v to denote the set of updated flows whose ingress switches are v. By Eq. (3), we have the following lemma:

Lemma 7: $\sum_{\gamma\in F_v}t_m\leq T_0,\ \forall v\in V$.

According to Lemma 7, we conclude that:

Theorem 8: If we omit the congestion-free constraint during update, the the RRSU algorithm can guarantee that, the link capacity will hardly be violated by a factor of $\frac{4\log n}{\alpha}+3$, and the route update delay constraint will not be violated by a factor of $\frac{3\log n}{\alpha}+4$ for the DSRU problem.

Note that, the previous Dionysus method [3] has shown that, the congestion-free and consistent constraints will not bring significant route update delay compared with simultaneously updating (*i.e.*, the Oneshot method) through efficient scheduling. As our RRSU algorithm only updates a smaller number of flows, the congestion-free constraint will also not bring significant update delay increase. In most practical situations, the RRSU algorithm can reach almost the constant bi-criteria approximation. For example, let $\widetilde{\lambda}$ be 0.4. The link capacity of today's networks will be 1Gbps. Observing the practical flow traces, the maximum intensity of a flow may reach 1Mbps or 10Mbps. Under two cases, $\frac{c^e_{\min}}{s(f)}$ will be 10^3 and 10^2, respectively. In a larger network with 1000 switches, $\log n=10$. The approximation factor for the link capacity constraint is 3.04 and 3.4, respectively. Since $\frac{T_0}{t_m}$ is usually 10^2 at least, the approximation factor for the route update delay constraint is 4.3. In other words, our RRSU algorithm can achieve the constant bi-criteria approximation for the DSRU problem in many situations.

Algorithm 2 GRSU: Greedy Route Selection and Update

```
Step 1: the same as that in RRSU
Step 2: Route Selection with Link Capacity Constraint
for each flow γ ∈ Γ do
    z_γ = Σ_{p∈P_γ−{R^c(γ)}} ỹ_γ^p
for each flow γ ∈ Γ in the decreasing order of z_γ do
    for each path p ∈ P_γ in the decreasing order of ỹ_γ^p do
        if all the links on path p can contain this flow then
            Assign p as its target route
            Update the remaining capacity of each link on p
            break
Step 3: Update Scheduling with Delay Constraint
Sort all the flows γ ∈ Γ^u in the decreasing order of z_γ
The current updated flow set is Γ'
repeat
    if Γ' = Φ or A flow has been updated then
        repeat
            Catch the header flow γ from the queue
            if Eq. (20) is satisfied for each link e ∈ R^f(γ) then
                Schedule Update of this flow, Γ' = Γ' ∪ {γ}
                Remove flow γ from queue
        until (Such a flow is not found)
until (The update delay is running out)
```

D. Greedy Route Selection and Update (GRSU) Algorithm

Though the RRSU algorithm almost achieves the bi-criteria approximation performance for the DSRU problem, the randomized rounding mechanism cannot fully guarantee that both the route update delay and link capacity constraints are always met. Below we propose a greedy route selection and update (GRSU) algorithm which satisfies both two constraints. The GRSU algorithm is formally described in Alg. 2.

The GRSU algorithm mainly consists of three steps. Same as RRSU, the first step of GRSU constructs a linear program by Eq. (3) as a relaxation of the R-DSRU problem. Assume that the optimal solution for Eq. (3) is denoted by $\widetilde{y^p_\gamma}$.

In the second step, we will choose a subset of flows for route update while satisfying the link capacity constraint. Let variable z_γ denote the probability with which flow γ is selected for route update. For each flow $\gamma\in\Gamma$, we compute z_γ as $z_\gamma = \sum_{p\in\mathcal{P}_\gamma-\{\mathcal{R}^c(\gamma)\}}\widetilde{y^p_\gamma}$. The algorithm sorts all the flows by the decreasing order of z_γ, and checks these flows one by one. For each flow $\gamma\in\Gamma$, we sort all the feasible paths in P_γ by the decreasing order of $\widetilde{y^p_\gamma}$. For a feasible path p, if all the links on path p can contain this flow, that is, the remaining capacity of each link on path p is not less than $s(\gamma)$, we find a path p as the target route of flow γ. Moreover, we update the remaining capacity of each link in the network. By this way, we have determined the target route configuration $\mathcal{R}^f(\gamma)$.

In the third step, the algorithm schedules the update operations on switches while satisfying several constraints. We sort all the flows by the decreasing order of z_γ, and put them into a queue. The set of being updated flows is denoted by

Γ'. When Γ' is null (*i.e.*, Φ) or the route update of a flow has been finished, we choose several flows from the queue for simultaneous update without congestion, which will be described in the next paragraph. To guarantee route consistency, we require that the update on the ingress switch should start after the updates on the internal switches are finished. Due to link capacity constraint, we may not find such a flow for update. Under this situation, a natural way is to reduce the flow rate so as to satisfy the link capacity constraint. The algorithm will be terminated until the update delay is running out.

We introduce the efficient way for choose several flows for simultaneous update without congestion. We catch the header flow from the queue, and schedule the route update for this flow if this flow and all flows in set Γ' can be updated without transient congestion. This procedure is terminated until we cannot find such a flow from the queue. Assume that the current route configuration is denoted by $\mathcal{R}^1$. For the header flow γ, the transient congestion can be avoided if the following constraint is satisfied on each link e of its target route:

$$\sum_{\gamma' \in \Gamma - \Gamma'} \sum_{e \in \mathcal{R}^1(\gamma')} s(\gamma') + \sum_{\gamma' \in \Gamma'} \sum_{e \in \mathcal{R}^1(\gamma') \cup \mathcal{R}^f(\gamma')} s(\gamma') \leq c(e) - s(\gamma) \quad (20)$$

Eq. (20) shows that the traffic load on link e consists of two parts: 1) For each $\gamma' \in \Gamma - \Gamma'$, the controller will not change its route, *i.e.*, $\mathcal{R}^1(\gamma')$. 2) If the controller is updating the route of flow $\gamma' \in \Gamma'$, due to asynchronous operations on different switches, we cannot determine its route, either the route before the update ($\mathcal{R}^1(\gamma')$) or the route after the update ($\mathcal{R}^f(\gamma')$).

E. Discussion

1) Jin *et al.* [3] have shown that the per-rule update latency may be varied for flow entries with different priorities by testing on the switches. To solve this case, we first estimate the expected delay for each update operation, and then apply the GRSU algorithm for real-time route update. However, due to various latency of per-rule update, it may be difficult to guarantee the route approximation bound.
2) In some application scenarios, the network traffic matrix may tremendously vary over space and time [11]. Thus, the flow intensity is unknown for the controller. Some previous works [11] use the number of burdened flows as the traffic load of each link. Accordingly, the link capacity $c(e)$ is defined as the maximum number of flows that one link e can pass through. Our GRSU algorithm can solve this case by setting $s(\gamma)$ as 1. Due to limited space, we omit the detailed description here.
3) The occasional topology change also triggers the route updates. For example, migrating virtual machines will make the routes of some flows be disrupted [15]. If we directly update the routes of all the affected flows by topology change, some flows may be blocked for a longer delay, due to route interruption. To reduce the delay, we first deploy some default paths using OSPF for those affected flows, similar as [5], so that they can recover data forwarding within a small delay. Then, we adopt our GRSU algorithm for real-time route update to achieve the load balancing.

IV. Simulation Results

This section first introduces the metrics and benchmarks for performance comparison (*Section IV-A*). Then, we describe the implementation of delay-satisfied route update algorithms on our SDN platform, and present testing results (*Section IV-B*). We evaluate our algorithm by comparing with the previous methods through extensive simulations (*Section IV-C*).

A. Performance Metrics and Benchmarks

Since this paper cares for real-time route update by joint optimization of route selection and update scheduling, we adopt two main performance metrics to measure the update efficiency. The first metric is link load ratio (LLR), which can be obtained by measuring the traffic load $l(e)$ of each link e. Then, LLR is defined as: $LLR = \max\{l(e)/c(e),\ e \in E\}$. The second one is the route update delay, which refers the delay for the update procedure from the current route configuration to the target one, by different algorithms.

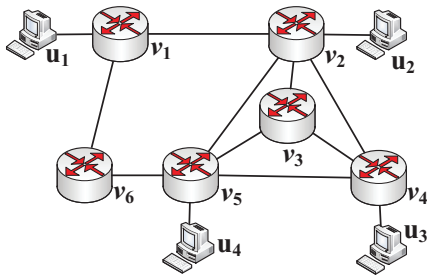
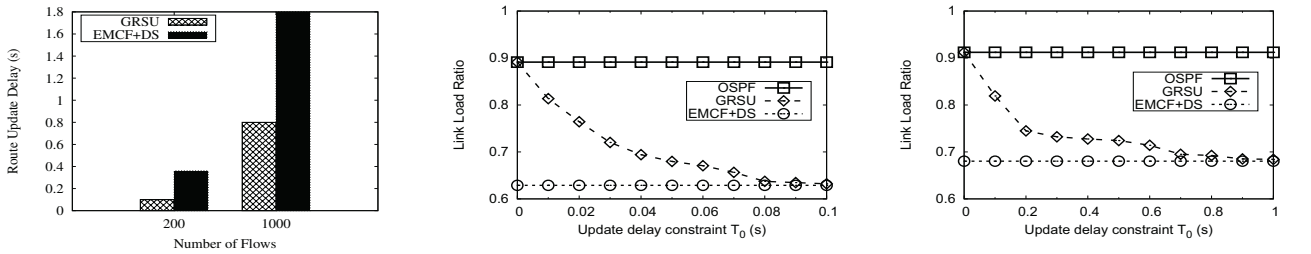


Fig. 2: Topology of the SDN Platform. Our platform is mainly composed of three parts: a controller, six OpenFlow enabled switches $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and four terminals $\{u_1, u_2, u_3, u_4\}$.

We implement the delay-satisfied route update algorithm on both the SDN platform and Mininet [16], which is a widely-used simulator for an SDN. To show update efficiency of our GRSU algorithm, we compare it with some other benchmarks. First, the controller often determines the target route configuration based on the current workload using the different routing algorithms, *e.g.*, the multi-commodity flow (MCF) algorithm [9], and executes route updates from the current route configuration to the target one using the update scheduling algorithm, *e.g.*, Dionysus [3]. Since the controller may update all flows, including elephant flows and mice flows, by the MCF algorithm, the update delay may likely be larger. For example, our simulation results show that, when there are $40K$ flows in topology (b), the update delay by joint MCF and Dionysus methods may reach $65s$, which is unacceptable for many applications. An improved version is that we *only* update the routes of those elephant flows [11], denoted by EMCF, and also adopt the Dionysus method [3] for update scheduling. The combined method is denoted by EMCF+DS. The second one is the OSPF protocol, which only chooses the shortest path for route selection, and does not apply route updates. This benchmark is adopted for comparing the route performance of our proposed algorithm. The third one is the optimal result of the linear program LP_1 in Eq. (3), denoted by OPT. Since LP_1 is the relaxed version of the R-DSRU problem, and R-DSRU is the relaxed version of DSRU, OPT



(a) Route Update Delay vs. Number of Flows

(b) Link Load Ratio vs. Update Delay Constraint with 200 flows

(c) Link Load Ratio vs. Update Delay Constraint with 1000 flows

Fig. 3: Testing Results Through the SDN Platform.

is a lower-bound for both R-DSRU and DSRU. We mainly observe the impact of two parameters, *i.e.*, number of flows and route update delay constraint T_0, on the route update performance. Intuitively, when parameter T_0 increases, since the routes of more flows can be updated, the link load ratio will be reduced. Due to limited capacity, our commodity switch cannot support the delay measurement of insertion and modification operations on the flow table. According to the testing results on the HP ProCurve 5406zl switch [17], the delays for insertion and modification operations are set as 5ms and 10ms, respectively. We also take these results in our platform testing and simulations.

B. Test-bed Evaluation

1) Implementation On the Platform: We implement the OSPF, EMCF+DS and GRSU algorithms on a real test-bed. Our SDN platform is mainly composed of three parts: a server installed with the controller's software, a set of OpenFlow enabled switches and some terminals. Specifically, we choose Opendaylight, which is an open source project supported by multiple enterprises, as the controller's software. The Opendaylight controller is running on a server with a core i5-3470 processor and 4GB of RAM. The topology of our SDN platform is illustrated in Fig. 2. The forwarding plane of an SDN comprises of 6 H3C S5120-28SC-HI switches, which support the OpenFlow v1.3 standard. During the platform implementation, each flow is identified by three elements, source IP, destination IP and TCP port, so that each terminal is able to generate different numbers of flows to other terminals.

2) Testing results: We mainly observe the impact of update delay constraint on the performance of link load ratio. Two sets of experiments are run on the platform by generating different numbers of flows. The first experiment contains 200 flows, the other contains 1000 flows. In each experiment, there are 20% elephant flows and 80% mice flows. Fig. 3(a) shows that it takes about $0.36s$ and $1.80s$ for the update procedure by the EMCF+DS algorithm when there are 200 and 1000 flows in a network. Figs. 3(b) and 3(c) show that, the link load ratio is improved with the increase of the route update delay constraint by our GRSU algorithm. However, the improvement is much slower with the increasing route update delay. Note that, the route performance of EMCF+DS will not change with update delay constraint. Fig. 3(b) shows that our algorithm can reduce the route update delay by 77% compared with the EMCF+DS method while preserving a similar routing performance (with link load ratio increased about 1%). Fig. 3(c) shows that, our GRSU algorithm can achieve the close route performance as EMCF+DS (with link load ratio increased about 2%) while it reduces the update delay about 61%. From these testing results, our GRSU algorithm achieves the better trade-off performance between route performance and route update delay.

C. Simulation Evaluation

1) Simulation Setting: In the simulations, we choose two practical and typical topologies with different network sizes [18]. The first topology, denoted by (a), contains 20 switches and 74 links. The second topology, denoted by (b), contains 100 switches and 397 links. For the both topologies, each link has a uniform capacity, 100Mbps. We execute each simulation 100 times, and give the average simulation results. The authors of [5] have shown that less than 20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we generate different numbers of flows, and the intensity of each flow obeys this 2-8 distribution.

2) Simulation Results: We run three groups of experiments to check the effectiveness of our algorithm. The first group of two simulations shows the route update delay by varying the number of flows in an SDN. We execute two algorithms, EMCF+DS and GRSU, on two different topologies. Fig. 4 shows that the required route update delay by the EMCF+DS algorithm is almost linearly increasing with the number of flows in a network. In a large network with $40K$ flows, it requires more than $19s$ by the right plot of Fig. 4. In the following simulations, we limit the route update delay constraint no more than $3.5s$ (and $2s$ by default), and mainly compare the route performance with the EMCF+DS algorithm for fairness. Obviously, even though the controller only updates the routes of those elephant flows, the required update delay by EMCF+DS is still much more than the update delay constraint.

The second group of simulations mainly shows how the route update delay constraint affects the link load ratio on two topologies. Given a fixed number of flows in a network, we change the route update delay constraints, and the route performance is shown in Figs. 5 and 6. Two figures show that, the link load ratio is reduced when the route update delay becomes larger by our proposed GRSU algorithm. However, the route update delay constraint does not affect the link load ratio of the EMCF+DS algorithm, which always updates the routes of those elephant flows in a network. For the small topology, Fig. 5 shows that our GRSU algorithm can reduce

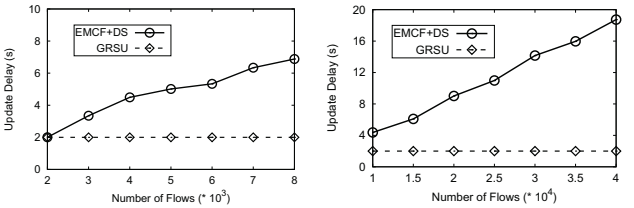


Fig. 4: Route Update Delay vs. Number of Flows. *Left plot*: Topology (a); *right plot*: Topology (b).

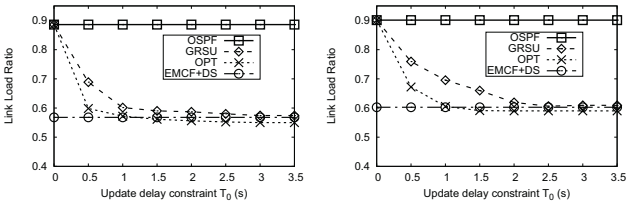


Fig. 5: Link Load Ratio vs. Update Delay Constraint for Topology (a). *Left plot*: 4000 flows; *right plot*: 8000 flows.

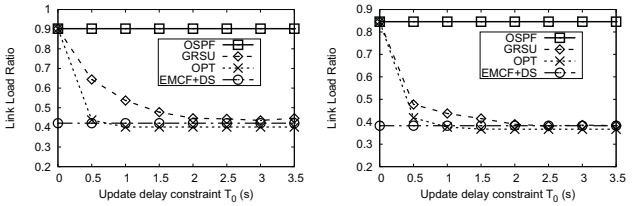


Fig. 6: Link Load Ratio vs. Update Delay Constraint for Topology (b). *Left plot*: 20000 flows; *right plot*: 40000 flows.

the route update delay by 60% compared with the EMCF+DS method while preserving a close route performance (with link load ratio increased less than 3%). For example, when there are 6000 flows in the network, the EMCF+DS method needs about $7s$ for route update by the left plot of Fig. 4. The left plot of Fig. 5 shows that the GRSU algorithm can achieve the similar route performance with EMCF+DS only with a route update delay of $2s$. For the large topology, we find that the GRSU algorithm can reduce the route update delay about 75% compared with EMCF+DS while still achieving the similar link load ratio performance by Fig. 6.

The third group of simulations shows how the number of flows affects the link load ratio performance on two topologies. Figs. 7 and 8 show that, the route performance of our GRSU algorithm is much closer to that of EMCF+DS with the increase of the route update delay constraint. More specifically, the GRSU algorithm with route update delay of $2s$ can achieve the similar route performance as EMCF+DS, which should take a route update delay of 5-8s in topology (a). In topology (b), Fig. 8 shows that our GRSU algorithm just takes about $2s$ for route update, so as to achieve the similar route performance as EMCF+DS, which will take a route update delay of 10-16s.

From these simulation results, we can make some conclusions. First, with the increase of the route update delay constraint, our algorithm can update more flows, and reduce the link load ratio. Second, Figs. 7 and 8 show that, with the increasing number of flows, the link load ratio will be increased under the same route update delay constraint by our GRSU algorithm. Third, Figs. 5-8 show that, the proposed algorithm almost achieves the similar performance as OPT,

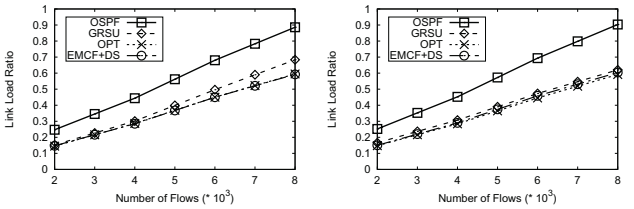


Fig. 7: Link Load Ratio vs. Number of Flows for Topology (a). *Left plot*: $T_0 = 1.0s$; *right plot*: $T_0 = 2.0s$.

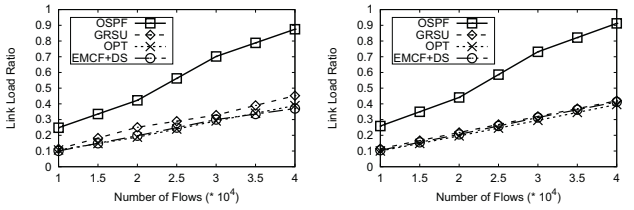


Fig. 8: Link Load Ratio vs. Number of Flows for Topology (b). *Left plot*: $T_0 = 1.0s$; *right plot*: $T_0 = 2.0s$.

which is the lower bound for the DSRU problem, provided that the route update delay constraint is not too small. Fourth, our proposed algorithm decreases the route update delay about 60-80% compared with the EMCF+DS algorithm. However, it can reach almost the similar route performance, such as load balancing, as EMCF+DS. Therefore, our proposed GRSU algorithm can achieve the better trade-off between the route performance and update delay by joint optimization of route selection and update scheduling.

V. Related Works

Due to network dynamics, the controller should frequently update its data plane so as to provide high resource utilization. The comprehensive survey of route update can be found in [19]. Almost all the previous methods usually compute the target route configuration based on the current workload, and design different algorithms for route update from the current route configuration to the target one. These studies can be divided into several categories by their optimization objectives, such as consistency-guarantee, and low update delay, etc.

The first category is to ensure the packet/flow consistency during the route update. Reitblatt *et al.* [10] introduced two abstractions for network updates: per-packet and per-flow consistency. These two abstractions guaranteed that a packet or a flow were handled either by the current route configuration before an update or by the target route configuration after an update, but never by both. Katta *et al.* [20] introduced a generic algorithm for implementing consistent updates that traded update time for rule-space overhead. They divided a global policy into a set of consistent slices and updated to the new policy of one slice at a time. By increasing the number of slices, the rule-space overhead on the switches could be reduced, and the route update delay could be increased. Mahajan *et al.* [21] highlighted the inherent trade-off between the strength of the consistency property and dependencies it imposed among rules at different switches. zUpdate [15] provided a primitive to manage the network-wide traffic migration for all the datacenter network updates. Given the end requirements of a specific datacenter network

update, zUpdate would automatically handle all the details, including computing a lossless migration plan and coordinating the changes to different switches. Canini *et al.* [22] studied a distributed control plane that enabled concurrent and robust policy implementation.

The second category is to minimize the update delay while satisfying other performance requirements, such as congestion-free and consistency-conservation, etc. Hong *et al.* [4] tried to minimize the number of rounds for congestion-free update through flow splitting. They formulated the route update problem into a linear program, solved it in polynomial time, and analyzed the possibly maximum round (or delay) for route update. Jin *et al.* [3] described Dionysus, a system for fast, consistent network updates in SDNs. Dionysus encoded as a graph the consistency-related dependencies among updates at individual switches, and it then dynamically scheduled these updates based on runtime differences in the update speeds of different switches. The authors of [23] presented a practical method for implementing accurate time-based updates, TIMEFLIPs, which could be used to implement atomic bundle updates, and to coordinate network updates with high accuracy.

Almost all the previous works update the network from the old route configuration to a new one, which is derived only based on the current workload. Though some works [3] have designed different algorithms to decrease the route update delay with consistency-guarantee, due to low-speed of TCAM operations, it may still result in a longer delay for route updates, especially in a large-scale or dynamically changed network. In most situations, the workload in a network has changed significantly after a certain period, *e.g.*, 20-60s [6]. If the route update takes a longer delay, the final route configuration may be inefficient for the workload after update. So, we need a real-time route update for an SDN, so as to achieve the trade-off between update delay and route performance.

VI. Conclusion

In this paper, we have studied the real-time route update while considering the current workload, the speed of TCAM updates, and the delay requirement on each switch. We have designed a rounding-based algorithm and a greedy method for the DSRU problem. The testing results on the SDN platform and the extensive simulation results have shown the high efficiency of our proposed algorithm.

Acknowledgement

This search of Dr. Xu, Dr. Yu and prof. Huang is supported by the NSFC under Grant No. U1301256, 61472383 and 61472385, the Natural Science Foundation of Anhui Province in China under No. 1408085MKL08, and the Natural Science Foundation of Jiangsu Province in China under No. BK20161257. The research of Prof. Li is partially supported by NSF ECCS-1247944, NSF CMMI 1436786, NSF CNS 1526638, National Natural Science Foundation of China under Grant No. 61520106007. Chen Qian is supported by UC Santa Cruz Startup Grant and NSF grant CNS-1464335.

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 Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[3] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 539–550.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM*, 2013, pp. 15–26.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[6] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *Distributed Computing and Networking*. Springer, 2013, pp. 439–444.

[7] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive packet sampling for flow volume measurement," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 9–9, 2002.

[8] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM, 2011, p. 8.

[9] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*. IEEE, 1975, pp. 184–193.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 323–334.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[12] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane," in *Software Defined Networking, 2012 European Workshop on*. IEEE, 2012, pp. 79–84.

[13] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *IEEE INFOCOM*, 2013, pp. 2211–2219.

[14] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[15] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 411–422, 2013.

[16] "The mininet platform," http://mininet.org/.

[17] "Hp procurve 5400 zl switch series," http://h17007.www1.hp.com/us/en/products/switches/HP_E5400_zl_Switch_Series/index.aspx.

[18] "The network topology from the monash university," http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.

[19] S. Wang, D. Li, and S. Xia, "The problems and solutions of network update in sdn: a survey," in *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*. IEEE, 2015, pp. 474–479.

[20] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 49–54.

[21] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, 2013, p. 20.

[22] M. Canini, P. Kuznetsov, D. Levin, S. Schmid *et al.*, "A distributed and robust sdn control plane for transactional network updates," in *The 34th Annual IEEE International Conference on Computer Communications (INFOCOM 2015)*, 2015.

[23] T. Mizrahi, O. Rottenstreich, and Y. Moses, "Timeflip: Scheduling network updates with timestamp-based tcam ranges," in *IEEE Infocom*, 2015.