URBM: User-Rank-Based Management of Flows in Data Center Networks through SDN

You-Chiun Wang
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
ycwang@cse.nsysu.edu.tw

Ting-Jui Hsiao
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
ss910034@gmail.com

***Abstract*—A *data center network (DCN)* is composed of many servers and switches, and how to manage flows makes a great impact on its performance. The paper applies the *software-defined networking* technology to flow management in a fat-tree DCN with hybrid switches, where electrical switches form the network backbone, while optical switches provide fast links. Considering that flows have different ranks, we propose a *user-rank-based management (URBM)* scheme to improve DCN throughput and offer differentiated QoS supports to flows based on their ranks. Low-rank flows have limits on using fast links, and their users should pay extra fees. If some links become busy, URBM finds substitute paths to reroute their flows. Alternatively, it restricts bandwidth usage of flows, where high-rank flows can use more bandwidth to satisfy QoS demands. With the *price elasticity of demand* model, we develop a pricing method to compute extra fees for low-rank users to use fast links. Simulation results show that URBM can improve flow throughput, provide better QoS supports for high-rank users, and raise the operator's revenue.**

***Keywords*—data center network, flow management, OpenFlow, pricing, software-defined networking.**

I. Introduction

Data center networks (DCNs) have been widely deployed to deliver various services such as cloud computing. A DCN contains many servers interconnected by switches, usually in a regular topology (e.g., fat tree and jellyfish) [1]. Since the amount of traffic could be large, it plays a key role in DCN performance to efficiently route flows between servers. Traditional switches forward packets based on a given routing protocol [2]. If congestion occurs, it usually relies on administrators to reconfigure the involved switches to alter their packet routing, which incurs a high cost [3].

To conquer this problem, the *software-defined networking (SDN)* technique separates the control plane from each switch and hands it over to a controller [4]. Thus, the controller can query switches about their statuses and also issue commands to them. In this way, it becomes easy to monitor network situation and change behavior of switches through the controller, thereby facilitating network management.

To save the hardware cost, many operators adopt electrical switches in their DCNs, which transmit data through *Ethernet cables (ECs)*. However, ECs are vulnerable to the attenuation of electrical signals, which limits their transmission speeds. Optical switches send light signals via *optical fibers (OFs)*, which overcomes this difficulty. They can also communicate with electrical switches by transferring light signals to electrical signals [5]. However, optical switches are more expensive than electrical switches. An economical solution is to add a few optical switches and make them cooperate with electrical switches [6]. Our work takes the above *hybrid-switch* solution, where electrical switches serve as the network backbone, while optical switches provides fast links.

This paper aims to exploit SDN to manage flows in a DCN with hybrid switches, whose topology is a common fat tree. Besides, users can choose to pay different unit prices and have different priorities and QoS supports accordingly. In particular, we divide users into three ranks (i.e., $R_{\mathbf{A}}$, $R_{\mathbf{B}}$, $R_{\mathbf{C}}$ ranks) [7], and propose a *user-rank-based management (URBM)* scheme. $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows can directly use fast links, but the latter has a limited quota. With extra frees, $R_{\mathbf{C}}$ flows are allowed to use fast links to improve throughput. The controller monitors network status and checks if some links are congested. If so, it seeks for substitute links or limits bandwidth usage of flows based on their ranks. As low-rank users need to pay extra fees for using fast links, we propose a pricing method to calculate reasonable fees, which considers the *price elasticity of demand (PED)* model [8]. Through simulations, we verify that URBM can improve flow throughput, offer different QoS supports to users, and increase the operator's revenue.

II. Related Work

Some studies manage flows in non-SDN-based DCNs. The study [9] first meets the demands of urgent flows by using the minimum bandwidth, and schedules other flows to shorten the competition time. In [10], each switch can differentiate flows by using multiple queues to store their packets. The work [11] lets switches record next-hop candidates for each destination in their routing tables. Then, each switch finds routes for flows to balance workloads of its ports. The study [12] assumes that each top-of-rack switch can connect with other switches in the DCN. If a top-of-rack switch is busy, it reroutes some flows to other switches to mitigate congestion. Wang et al. [13] apply machine learning to flow management in a multi-tenant DCN.

Various issues relevant to flow management in SDN-based DCNs are discussed. The study [14] finds working and recovery paths for flows. A low-rank flow will use the recovery path only if its working path fails. A high-rank flow can use both

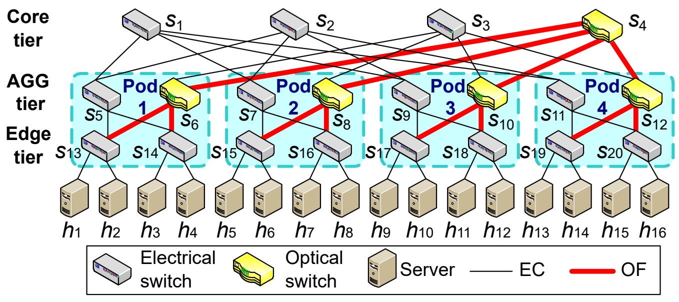


Fig. 1. Fat-tree DCN with electrical and optical switches, where $m = 4$.

TABLE I
SPECIFICATION OF FLOWS BASED ON USER RANKS.

Rank	*Price*	*QoS support*	*Using OF links*
$R_{\mathbf{A}}$	$P_{\mathbf{A}}$	$(Q_{\mathbf{A}})$% bandwidth reserved	No limit
$R_{\mathbf{B}}$	$P_{\mathbf{B}}$	$(Q_{\mathbf{B}})$% bandwidth reserved	With quota $\Lambda_{\mathbf{B}}$
$R_{\mathbf{C}}$	$P_{\mathbf{C}}$	No bandwidth reserved	Extra fees

TABLE II
SUMMARY OF NOTATIONS.

Notation	*Definition*
$R_{\mathbf{A}}$, $R_{\mathbf{B}}$, $R_{\mathbf{C}}$	User ranks (priority: $R_{\mathbf{A}} > R_{\mathbf{B}} > R_{\mathbf{C}}$)
$P_{\mathbf{A}}$, $P_{\mathbf{B}}$, $P_{\mathbf{C}}$	Unit prices for $R_{\mathbf{A}}$, $R_{\mathbf{B}}$, and $R_{\mathbf{C}}$ users
$Q_{\mathbf{A}}$, $Q_{\mathbf{B}}$	QoS supports for $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ users
$\Lambda_{\mathbf{B}}$	Quota for $R_{\mathbf{B}}$ flows to use OF links
$\Gamma_{\mathbf{P}}$, $\Gamma_{\mathbf{R}}$, $\Gamma_{\mathbf{D}}$	Port, route, and delay tables
$b_k^{\mathtt{max}}$, $b_k^{\mathtt{use}}$	Maximum and consumed bandwidth of the k-th port
$\hat{\mathcal{L}}_{\mathtt{EC}}^{\mathtt{top}}$, $\hat{\mathcal{L}}_{\mathtt{OF}}^{\mathtt{top}}$	Sets of EC and OF links in the top layer
$\hat{\mathcal{L}}_n^{\mathtt{pod}}$	Set of links in the n-th pod
$\delta_{\mathtt{low}}$, $\delta_{\mathtt{high}}$	Thresholds used in the congestion eliminating module

paths to raise throughput. Pang et al. [15] combine multipath TCP and segment routing to improve the DCN's performance while saving the deployment cost. The work [16] proposes a detection mechanism to find elephant flows that carry volumes of data. Zaher et al. [17] propose a flow scheduling method to maintain throughput of elephant flows. Evidently, these studies have different objectives with ours.

A number of flow management methods based on SDN are developed for fat-tree DCNs. Long et al. [18] find alternative paths for flows on busy links. If the network utilization exceeds a threshold, the controller chooses the flow which spends the maximum bandwidth from the busiest link, and reroutes it to a lower-load path. The *dynamic load-balanced path optimization (DLPO)* method [19] adopts the simple moving average of the variance of link loads to check whether link congestion occurs. If so, DLPO selects top 10% of the busiest links and reroutes their flows to other links with smaller utilization. The study [20] finds the shortest paths from each server to others and computes every link's cost. When congestion occurs in a path, it replaces this path with a substitute path whose link cost is the minimum. The *low-cost, load-balanced route management (L2RM)* method [21] checks if some links are busy by a load-deviation factor. Then, L2RM uses a group table to split flows by sending their packets along multiple paths, so as to mitigate congestion. As can be seen, none of these methods addresses user differentiation. This motivates us to propose the URBM scheme that can flexibly adjust routing paths and bandwidth usage of flows according to their ranks and demands, so as to provide different QoS supports to users.

III. System Model

We consider an SDN-based DCN, as shown in Fig. 1. The topology is an m-ary fat tree, where each switch has m ports. It arranges switches into three tiers: *edge*, *aggregation (AGG)*, and *core*. Edge and AGG switches are divided into m groups (known as *pods*). Each pod has $\frac{1}{2}m$ edge switches and $\frac{1}{2}m$ AGG switches, so every edge switch can connect with $\frac{1}{2}m$ AGG switches and $\frac{1}{2}m$ servers. Furthermore, each core switch connects with an AGG switch in every pod. There are at most $\frac{1}{4}m^2$ core switches. In theory, the fat tree can employ $\frac{5}{4}m^2$ switches to interconnect $\frac{1}{4}m^3$ servers. This topology has two benefits [1]. First, it becomes easy to add switches and broaden the DCN's scale. Second, we can find multiple shortest paths between any two servers in the DCN.

The DCN comprises hybrid switches. There is one optical AGG switch using OFs to connect with every edge switch in each pod. Besides, there is also an optical switch in the core tier used to connect with these optical switches in the AGG tier via OFs. All other switches are electrical, which connect with neighbors via ECs. The transmission speed of an OF is dozens or even hundreds of times of that of an EC, so optical switches can support high-speed communications in the DCN.

Let $R_{\mathbf{x}}$ be the rank of a user. Without loss of generality, we consider three ranks of users (i.e., $\mathbf{x} \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$). $R_{\mathbf{A}}$, $R_{\mathbf{B}}$, and $R_{\mathbf{C}}$ users are charged with unit prices $P_{\mathbf{A}}$, $P_{\mathbf{B}}$, and $P_{\mathbf{C}}$ for bandwidth usage, where $P_{\mathbf{A}} > P_{\mathbf{B}} > P_{\mathbf{C}}$, and have high, medium, and low priorities, respectively. The priority decides not only the degree of QoS support, but also the restriction on using OF links, as shown in Table I. For QoS support, if a link is busy, at least $Q_{\mathbf{A}}$ and $Q_{\mathbf{B}}$ percents of the link's bandwidth should be reserved for $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows, respectively, where $Q_{\mathbf{A}} > Q_{\mathbf{B}}$ (e.g., $Q_{\mathbf{A}} = 80$ and $Q_{\mathbf{B}} = 60$). When both $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows are sent via a busy link, the $Q_{\mathbf{A}}$ requirement should be satisfied first (as $R_{\mathbf{A}}$ flows have a higher priority than $R_{\mathbf{B}}$ flows). For $R_{\mathbf{C}}$ flows, we adopt the best-effort policy, which means that no bandwidth is reserved for them. Moreover, $R_{\mathbf{A}}$ users can use OF links at the basic fee. Each $R_{\mathbf{B}}$ flow is given with a quota $\Lambda_{\mathbf{B}}$ on using OF links. Once the flow uses up the quota, its user has to pay extra fees. $R_{\mathbf{C}}$ flows are only allowed to use OF links with extra fees. $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ users can choose not to pay extra fees. In this case, their flows may not use OF links (the detail will be discussed in Section IV-B).

Our objective is to manage flows based on their ranks and demands, so as to mitigate congestion and improve throughput in the DCN. Moreover, we also develop a pricing method by considering the PED model to compute reasonable user fees. Table II summarizes the notations.

IV. The URBM Scheme

The URBM scheme has five modules. The *table maintaining module* helps the controller keep abreast of the network state.

TABLE III
EXAMPLES OF STATUS TABLES.

(a) Port table $\Gamma_{\mathbf{P}}$:

Switch	*Maximum and consumed bandwidth of each port*
s_1	(1, 10^4, 3430), (2, 10^4, 7217), (3, 10^4, 2105), (4, 10^4, 4323)
s_6	(1, 10^4, 5811), (2, 10^6, 281327), (3, 10^6, 93873), (4, -1, -1)

(b) Route table $\Gamma_{\mathbf{R}}$:

Server pair	*Edge switches*	*AGG and core switches*
$h_1 \Leftrightarrow h_2$	$\{s_{13}\}$	
$h_1 \Leftrightarrow h_4$	$\{s_{13}, s_{14}\}$	$\{s_5\}$, $\{s_6^*\}$
$h_1 \Leftrightarrow h_6$	$\{s_{13}, s_{15}\}$	$\{s_5, [s_1, s_2], s_7\}$, $\{s_6^*, [s_3, s_4^*], s_8^*\}$

(c) Delay table $\Gamma_{\mathbf{D}}$:

Edge switch	*AGG switch*	*Packet delay*
s_{13}	s_5	15.34
s_{14}	s_6	3.96

The *route building module* finds routing paths for new flows. The *two-layer checking module* judges whether some links are busy, and then the *congestion eliminating module* adjusts flows sent by busy links based on their ranks and demands. Finally, the *price evaluating module* calculates user fees.

A. Table Maintaining Module

The controller regularly queries each switch about its status and takes down the collected information in three *status tables*. In this way, it can check if congestion occurs, and then adjust routes or bandwidth usage of flows by consulting these tables.

First, the *port table* $\Gamma_{\mathbf{P}}$ records the bandwidth consumption of switches. The format of each entry is "switch, (1, $b_1^{\max}$, b_1^{use}), $\cdots$, (m, $b_m^{\max}$, b_m^{use})". Here, $b_k^{\max}$ and b_k^{use} indicate the maximum bandwidth and the consumed bandwidth of the k-th port in the switch, respectively ($k = 1..m$), where the unit is kbps. To do so, the controller queries a switch about the number φ_k^{Tx} of bits sent and the number φ_k^{Rx} of bits received by its k-th port in the last period. Then, b_i^{use} is calculated by

$$b_k^{\text{use}} = \frac{\varphi_k^{\text{Tx}} + \varphi_k^{\text{Rx}}}{1000 \times \Delta_t}, \qquad (1)$$

where Δ_t is the period length (in seconds). Table III(a) gives an example, where the capacity of ECs and OFs is 10Mbps and 1Gbps, respectively. Note that if a port is unused or broken, both $b_k^{\max}$ and b_k^{use} are set to -1 (e.g., s_6's 4th port). Then, Theorem 1 analyzes the size of table $\Gamma_{\mathbf{P}}$.

Theorem 1. *The port table $\Gamma_{\mathbf{P}}$ contains $\frac{5}{4}m^2(3m+1)$ items for an m-ary fat-tree DCN in the worst case.*

Proof. An m-ary fat tree contains at most $\frac{5}{4}m^2$ switches. In $\Gamma_{\mathbf{P}}$, we record the identification of every switch and also three items for each of its m ports (i.e., port #, $b_i^{\max}$, b_i^{use}). So, $\Gamma_{\mathbf{P}}$ has $\frac{5}{4}m^2(3m+1)$ items in the worst case. □

Second, the *route table* $\Gamma_{\mathbf{R}}$ records the shortest routing paths between any two servers. As mentioned in Section III, servers connect to edge switches, while AGG and core switches offer multiple choices of routes. Based on this structure, we propose an efficient storage method for $\Gamma_{\mathbf{R}}$, as referred to Fig. 1 and Table III(b). Let h_x and h_y be the source and the destination, respectively. The storage method has the following three rules:

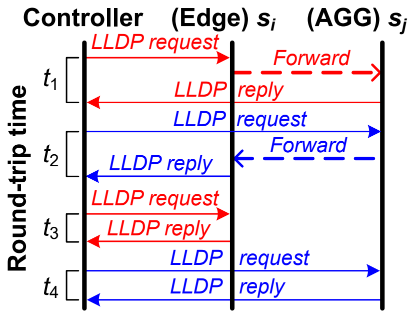


Fig. 2. Four-step method to estimate delay $\tau_{i,j}$ of a link (s_i, s_j) by LLDP.

1-1. Both h_x and h_y connect with an edge switch s_i. In this case, we can record just s_i in $\Gamma_{\mathbf{R}}$. For example, the path from h_1 to h_2 includes only s_{13}.

1-2. h_x and h_y are in the same pod, but they are not linked by the same edge switch. The shortest path will start from the edge switch linking to h_x and end at the edge switch linking to h_y. Moreover, there are $\frac{1}{2}m$ choices of AGG switches. Take h_1 and h_4 as an example. There are two paths: $\langle s_{13}, s_5, s_{14}\rangle$ and $\langle s_{13}, s_6, s_{14}\rangle$ (we omit h_1 and h_4). The notation '$*$' means that a switch is optical (i.e., the path has OF links).

1-3. h_x and h_y are in different pods. Thus, there may exist m paths, each including a core switch. We use square brackets (i.e., '[]') to indicate alternative switches in the core tier. Take h_1 and h_6 as an example. There are four paths from h_1 to h_6: $\langle s_{13}, s_5, s_1, s_7, s_{15}\rangle$, $\langle s_{13}, s_5, s_2, s_7, s_{15}\rangle$, $\langle s_{13}, s_6, s_3, s_8, s_{15}\rangle$, and $\langle s_{13}, s_6, s_4, s_8, s_{15}\rangle$.

As the fat tree has a symmetrical structure, the shortest paths from h_y to h_x must be identical to those from h_x to h_y (in a reverse sequence). Thus, we need not store the shortest paths from h_y to h_x in $\Gamma_{\mathbf{R}}$. Theorem 2 analyzes the size of $\Gamma_{\mathbf{R}}$.

Theorem 2. *Given an m-ary fat tree, there are at most $\frac{1}{8}(m^3 - m^4 - \frac{1}{4}m^5 + \frac{1}{8}m^6 + \frac{1}{2}m^7)$ switches recorded in table $\Gamma_{\mathbf{R}}$ by using our storage method.*

Proof. In rule 1-1, each edge switch can connect with at most $\frac{1}{2}m$ servers, so there are $\frac{1}{4}m$ possible combinations of server pairs. Since there are $\frac{1}{2}m^2$ edge switches in the fat tree, we record no more than $\frac{1}{4}m \times \frac{1}{2}m^2$ switches by rule 1-1.

In rule 1-2, we have $\frac{1}{2}m \times \frac{1}{4}m^2(\frac{1}{4}m^2 - \frac{1}{2}m)$ combinations. From Table III(b), each path for a server pair includes 2 edge switches and $\frac{1}{2}m$ AGG switches. Therefore, we record at most $\frac{1}{2}m \times \frac{1}{4}m^2(\frac{1}{4}m^2 - \frac{1}{2}m) \times (2 + \frac{1}{2}m)$ switches by rule 1-2.

In rule 1-3, there exist $\frac{1}{8}m^3(\frac{1}{4}m^3 - \frac{1}{4}m^2)$ combinations. Similarly, each path for a server pair contains 2 edge switches, m AGG switches, and m core switches, so we record at most $\frac{1}{8}m^3(\frac{1}{4}m^3 - \frac{1}{4}m^2) \times (2 + 2m)$ switches by rule 1-3.

By summing the maximum number of switches recorded in each rule, we can thus verify this theorem. □

Third, the *delay table* $\Gamma_{\mathbf{D}}$ records the average packet delay of each link in pods. The format of each entry is "$s_i, s_j, \tau_{i,j}$", where s_i and s_j are the edge and AGG switches that connect with each other, and $\tau_{i,j}$ gives the packet delay of link (s_i, s_j)

(measured in ms). An example is presented in Table III(c). To estimate $\tau_{i,j}$, we propose a four-step method by using the *link layer discovery protocol (LLDP)*, as shown in Fig. 2. In step 1, the controller sends s_i an LLDP request that queries s_j. In this case, s_i forwards the request to s_j, and s_j sends an LLDP reply to the controller. In step 2, the controller sends s_j an LLDP request that queries s_i. In step 3, the controller sends s_i one LLDP request which queries s_i. In step 4, the controller sends s_j an LLDP request that queries s_j. Let t_x be the round-trip time in step x ($x = 1, 2, 3, 4$). Then, the average packet delay of link (s_i, s_j) can be calculated by $\tau_{i,j} = \frac{1}{2}(t_1 + t_2 - t_3 - t_4)$. In Theorem 3, we analyze the size of table $\Gamma_{\mathbf{D}}$ and also the message cost to update it.

Theorem 3. *Given an m-ary fat tree, table $\Gamma_{\mathbf{D}}$ has no more than $\frac{1}{4}m^3$ entries. To update $\Gamma_{\mathbf{D}}$, the controller needs to send at most m^3 LLDP requests to edge and AGG switches.*

Proof. The number of $\Gamma_{\mathbf{D}}$'s entries will be equal to the number of links between edge and AGG switches. From Fig. 1, each edge switch connects with all AGG switches in a pod. Since the fat tree has m pods, where each pod has $\frac{1}{2}m$ edge switches and $\frac{1}{2}m$ AGG switches, there are no more than $(m \times \frac{1}{2}m \times \frac{1}{2}m)$ links between edge and AGG switches. In other words, $\Gamma_{\mathbf{D}}$ has $\frac{1}{4}m^3$ entries in the worst case. Because it requires four LLDP requests to estimate $\tau_{i,j}$ for every link, the message cost (in terms of LLDP requests) to update $\Gamma_{\mathbf{D}}$ is $\frac{1}{4}m^3 \times 4 = m^3$. $\square$

B. Route Building Module

Whenever a flow is generated, the controller determines its routing path based on the route table $\Gamma_{\mathbf{R}}$. If there are multiple choices (i.e., rules 1-2 and 1-3), we select the path such that the average load of its links is the minimum. Specifically, the load of a link l_i is calculated by $b_i^{\text{use}}/b_i^{\text{max}}$, where b_i^{use} and b_i^{max} are the consumed and maximum bandwidths of l_i (these two parameters can be found in the port table $\Gamma_{\mathbf{P}}$).

As mentioned in Section III, $R_{\mathbf{A}}$ users need not pay extra fees for using OF links. On the other hand, $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ users can indicate their volition on whether to pay extra fees in the contracts. If a user prefers not to pay extra fees, the controller puts a constraint on the route selection for the user's flows. More concretely, for $R_{\mathbf{B}}$ users, their flows can be sent via OF links up to a limit. Once an $R_{\mathbf{B}}$ flow uses up quota $\Lambda_{\mathbf{B}}$, the controller replaces its path by another path without OF links. For $R_{\mathbf{C}}$ users, the controller avoids selecting paths that include OF links to send their flows. This can be done by removing the AGG and core switches with the '*' mark from the choices of routes, as shown in Table III(b).

C. Two-Layer Checking Module

This module checks whether some links are busy. Let us observe the topology in Fig. 1, all links between switches can be divided into two layers. The *top layer* includes the links between core and AGG switches, and the *pod layer* contains the links between AGG and edge switches. These two layers of links have different traffic patterns, so we propose different checking methods for them.

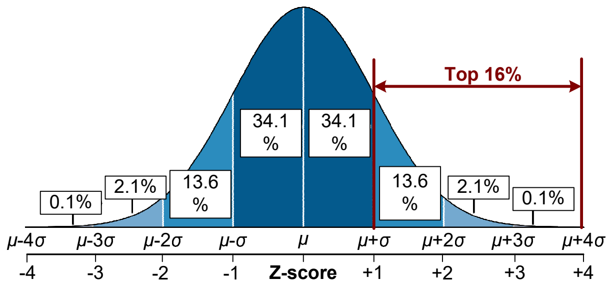


Fig. 3. Normal distribution and z-scores.

1) Top-Layer Checking Method: Top-layer links are used to route cross-pod packets, so their traffic may be more diverse. To find busy links, we adopt the *z-score* solution. As OFs can carry far more traffic than ECs, they are handled separately. Let $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$ denote the set of EC links in the top layer. Then, the z-score of each EC link $l_i \in \hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$ is estimated by

$$z_i = (b_i^{\text{use}}/b_i^{\text{max}} - \mu_{\text{EC}}^{\text{top}})/\sigma_{\text{EC}}^{\text{top}}, \quad \text{if } \sigma_{\text{EC}}^{\text{top}} \neq 0, \tag{2}$$

where $\mu_{\text{EC}}^{\text{top}}$ is the average load of EC links in $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$:

$$\mu_{\text{EC}}^{\text{top}} = \left(\sum\nolimits_{\forall l_j \in \hat{\mathcal{L}}_{\text{EC}}^{\text{top}}} b_j^{\text{use}}/b_j^{\text{max}}\right)/|\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}|, \tag{3}$$

and $\sigma_{\text{EC}}^{\text{top}}$ is the standard deviation of EC-link loads in $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$:

$$\sigma_{\text{EC}}^{\text{top}} = \sqrt{\left(\sum\nolimits_{\forall l_j \in \hat{\mathcal{L}}_{\text{EC}}^{\text{top}}} (b_j^{\text{use}}/b_j^{\text{max}} - \mu_{\text{EC}}^{\text{top}})^2\right)/|\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}|}. \tag{4}$$

Suppose that the loads of EC links in $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$ follow a normal distribution, as shown in Fig. 3. When $z_i > 1$, it means that l_i's load exceeds $(\mu_{\text{EC}}^{\text{top}} + \sigma_{\text{EC}}^{\text{top}})$, and l_i belongs to the top 16% of the busiest links in $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$. In view of this, we treat l_i as a busy EC link if its z-score is larger than one.

Similarly, let $\hat{\mathcal{L}}_{\text{OF}}^{\text{top}}$ be the set of OF links in the top layer. we can find the average $\mu_{\text{OF}}^{\text{top}}$ and the standard deviation $\sigma_{\text{OF}}^{\text{top}}$ of OF-link loads in $\hat{\mathcal{L}}_{\text{OF}}^{\text{top}}$ by Eqs. (3) and (4), respectively, where $\hat{\mathcal{L}}_{\text{EC}}^{\text{top}}$ is replaced by $\hat{\mathcal{L}}_{\text{OF}}^{\text{top}}$. After that, we can calculate the z-score of each OF link in $\hat{\mathcal{L}}_{\text{OF}}^{\text{top}}$ by Eq. (2) with parameters $\mu_{\text{OF}}^{\text{top}}$ and $\sigma_{\text{OF}}^{\text{top}}$, and then judge whether that link is busy.

2) Pod-Layer Checking Method: As their names suggest, pod-layer links are used for intra-pod communications. In a fat tree, every pod has a similar structure, as shown in Fig. 1. Thus, we can measure the average load of links in each pod, and check if congestion occurs in some pods. Let $\hat{\mathcal{L}}_n^{\text{pod}}$ be the set of links in the n-th pod. Since $\hat{\mathcal{L}}_n^{\text{pod}}$ has ECs and OFs, we find the *weighted arithmetic average* of all links in $\hat{\mathcal{L}}_n^{\text{pod}}$:

$$\mu_n^{\text{pod}} = \frac{\sum_{\forall l_j \in \hat{\mathcal{L}}_n^{\text{pod}}} w_j \times (b_j^{\text{use}}/b_j^{\text{max}})}{\sum_{\forall l_j \in \hat{\mathcal{L}}_n^{\text{pod}}} w_j}, \tag{5}$$

where w_j is the weight of a link l_j, which is set as follows:

$$w_j = \begin{cases} \frac{b_{\text{EC}}^{\text{max}}}{b_{\text{EC}}^{\text{max}} + b_{\text{OF}}^{\text{max}}} & \text{if } l_j \text{ is an EC} \\ \frac{b_{\text{OF}}^{\text{max}}}{b_{\text{EC}}^{\text{max}} + b_{\text{OF}}^{\text{max}}} & \text{if } l_j \text{ is an OF,} \end{cases} \tag{6}$$

where $b_{\text{EC}}^{\text{max}}$ and $b_{\text{OF}}^{\text{max}}$ denote the maximum bandwidth of ECs and OFs, respectively. Evidently, OF links have larger weights than EC links due to $b_{\text{OF}}^{\text{max}} > b_{\text{EC}}^{\text{max}}$.

Let Ψ be the set of all links in the pod layer. Then, $\hat{\mathcal{L}}_n^{\texttt{pod}}$'s z-score is estimated as follows:

$$z_n^{\texttt{pod}} = (\mu_n^{\texttt{pod}} - \mu_\Psi)/\sigma_\Psi, \quad \text{if } \sigma_\Psi \neq 0 \tag{7}$$

where

$$\mu_\Psi = \sum\nolimits_{\forall \hat{\mathcal{L}}_x^{\texttt{pod}} \subset \Psi} \mu_x^{\texttt{pod}}/|\Psi|. \tag{8}$$

$$\sigma_\Psi = \sqrt{\sum\nolimits_{\forall \hat{\mathcal{L}}_x^{\texttt{pod}} \subset \Psi} (\mu_x^{\texttt{pod}} - \mu_\Psi)^2/|\Psi|}. \tag{9}$$

If $z_n^{\texttt{pod}} > 1$, then congestion occurs in the n-th pod.

D. Congestion Eliminating Module

This module mitigates congestion of a busy link according to its layer and type, which considers three cases below.

1) Top-Layer EC Links: Let l_i be a busy EC link in $\hat{\mathcal{L}}_{\texttt{EC}}^{\texttt{top}}$ and b_i^{res} be l_i's residual bandwidth (i.e., $b_i^{\text{res}} = b_i^{\max} - b_i^{\text{use}}$). As each AGG switch connects with $\frac{1}{2}m$ core switches, there will exist other paths to help route l_i's packets in the top layer. Fig. 1 presents an example, where link (s_5, s_1) is busy, and link (s_5, s_2) has more residual bandwidth than link (s_5, s_1). For path $\langle s_5, s_1, s_7\rangle$ which contains the busy link, there is an alternative path $\langle s_5, s_2, s_7\rangle$ to share its load. Here, we call link (s_5, s_2) a *helper* for busy link (s_5, s_1).

Suppose that link l_h is the helper for l_i (i.e., $b_h^{\text{res}} > b_i^{\text{res}}$). When there are multiple helpers, we choose the one with the largest b_h^{res} value. However, if no helper is found, we employ the method in Section IV-D2 to limit bandwidth usage of flows sent via l_i. Otherwise, we ask l_h to share l_i's load by rerouting some of its packets via l_h, which can be performed through the group table. To do so, we use two thresholds $\delta_{\texttt{low}}$ and $\delta_{\texttt{high}}$ to adjust bucket weights in the group table, which decide the ratio of packets sent via l_i and l_h, where $0 < \delta_{\texttt{low}} \leq 0.25$ and $0.75 \leq \delta_{\texttt{high}} < 1$. There are three rules to set bucket weights:

2-1. $\delta_{\texttt{low}} \leq b_i^{\text{res}}/b_h^{\text{res}} \leq \delta_{\texttt{high}}$. To ensure that the amount of data sent through l_i and l_h is proportional to their residual bandwidth, we set budget weights of l_i and l_h to $\frac{100\times b_i^{\text{res}}}{b_i^{\text{res}}+b_h^{\text{res}}}$ and $(100-\frac{100\times b_i^{\text{res}}}{b_i^{\text{res}}+b_h^{\text{res}}})$, respectively. For instance, if $b_i^{\text{res}} = 3500$ kbps and $b_h^{\text{res}} = 5000$ kbps, l_i's budget weight is $\frac{100\times 3500}{3500+5000} \approx 41$ and l_h's budget weight is 59.

2-2. $b_i^{\text{res}}/b_h^{\text{res}} < \delta_{\texttt{low}}$. In this case, l_i is seriously congested. If we apply rule 2-1, l_h may be flooded with l_i's packets and become busy. Even worse, budget weights of l_i and l_h will be frequently exchanged, causing a ping-pong effect. Thus, we set budget weights of l_i and l_h to $(100 \times \delta_{\texttt{low}})$ and $100(1 - \delta_{\texttt{low}})$, respectively. For example, suppose that $b_i^{\text{res}} = 850$ kbps, $b_h^{\text{res}} = 9950$ kbps, and $\delta_{\texttt{low}} = 0.2$. If rule 2-1 is applied, budget weights of l_i and l_h are $\frac{100\times 850}{850+9950} \approx 8$ and 92, respectively, which have extreme values. Instead, we set budget weights of l_i and l_h to 20 and 80, respectively, so as to solve the problem.

2-3. $b_i^{\text{res}}/b_h^{\text{res}} > \delta_{\texttt{high}}$. The loads of l_i and l_h are similar. If l_h has enough bandwidth (e.g., $b_h^{\text{res}}/b_h^{\max} \geq 0.2$), we set budget weights by rule 2-1. Otherwise, both l_i and l_h are busy, so we adopt the method in Section IV-D2 to limit bandwidth usage of flows sent by these links.

TABLE IV
EXAMPLES OF MITIGATING CONGESTION IN TOP-LAYER OF LINKS.

(a) All flows in $\hat{\mathcal{F}}_i$ are homogeneous:

Flow	*Size*	*Queue*	*Max-rate*	*Allocation*
f_1	900 Mbps	Queue 0	600 Mbps	600 Mbps
f_2	600 Mbps	Queue 1	400 Mbps	400 Mbps

(b) Flows in $\hat{\mathcal{F}}_i$ are heterogeneous:

Flow	*Size*	*Meter-rate*	*Queue*	*Max-rate*	*Allocation*
f_1	900 Mbps	800 Mbps	Queue 0	480 Mbps	480 Mbps
f_2	600 Mbps	800 Mbps	Queue 1	320 Mbps	320 Mbps
f_3	500 Mbps	200 Mbps	-	-	200 Mbps

2) Top-Layer OF Links: Since the capacity of an OF link is much larger than that of an EC link, it is not a good idea to ask EC links to share the loads of OF links (as these EC links would be seriously congested). In view of this, we restrict the amount of bandwidth used by each flow on busy OF links. Let l_i be a busy OF link in $\hat{\mathcal{L}}_{\texttt{OF}}^{\texttt{top}}$ and $\hat{\mathcal{F}}_i$ be the set of flows sent via l_i. When all flows in $\hat{\mathcal{F}}_i$ are homogeneous (i.e., they have the same rank), we use the Set_Queue operation to allot bandwidth to them. More concretely, each port of the switch is associated with eight queues in OpenFlow. We assign each flow $f_j \in \hat{\mathcal{F}}_i$ to one queue, and find its *max-rate* (which gives the upper bound on bandwidth usage for the flow) as follows:

$$\frac{\tilde{S}(f_j) \times b_i^{\max}}{\sum_{\forall f_x \in \hat{\mathcal{F}}_i} \tilde{S}(f_x)}, \tag{10}$$

where $\tilde{S}(f_j)$ denotes f_j's size (i.e., traffic amount). Thus, l_i's bandwidth is allocated to flows in proportion to their sizes. Table IV(a) gives an example, where two $R_{\mathbf{A}}$ flows f_1 and f_2 are sent via the same OF link whose capacity is 1 Gbps. Then, the amount of bandwidth allocated to f_1 and f_2 will be $\frac{900}{900+600} \times 1$ Gbps $= 600$ Mbps and 400 Mbps, respectively.

When $\hat{\mathcal{F}}_i$ contains heterogeneous flows, we provide different QoS supports based on their ranks. Recall that $Q_{\mathbf{A}}$ and $Q_{\mathbf{B}}$ percents of l_i's bandwidth are reserved for $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows, respectively. Thus, we divide $\hat{\mathcal{F}}_i$ into three subsets $\hat{\mathcal{F}}_i^{\mathbf{A}}$, $\hat{\mathcal{F}}_i^{\mathbf{B}}$, and $\hat{\mathcal{F}}_i^{\mathbf{C}}$ that include all $R_{\mathbf{A}}$, $R_{\mathbf{B}}$, and $R_{\mathbf{C}}$ flows, respectively. Then, we use a meter table to adjust bandwidth usage of flows [22]. For $\hat{\mathcal{F}}_i^{\mathbf{A}}$, the meter-rate is set to

$$\beta_{\mathbf{A}} = \min\left\{\sum\nolimits_{\forall f_j \in \hat{\mathcal{F}}_i^{\mathbf{A}}} \tilde{S}(f_j), b_i^{\max} \times Q_{\mathbf{A}}\%\right\}. \tag{11}$$

Here, once the total size of $R_{\mathbf{A}}$ flows in $\hat{\mathcal{F}}_i^{\mathbf{A}}$ is above the $Q_{\mathbf{A}}$ quota, we give them $(b_i^{\max} \times Q_{\mathbf{A}}\%)$ bandwidth. For $\hat{\mathcal{F}}_i^{\mathbf{B}}$ the meter-rate is set to

$$\beta_{\mathbf{B}} = \min\left\{\sum\nolimits_{\forall f_j \in \hat{\mathcal{F}}_i^{\mathbf{B}}} \tilde{S}(f_j), b_i^{\max} \times Q_{\mathbf{B}}\%, b_i^{\max} - \beta_{\mathbf{A}}\right\}. \tag{12}$$

If $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows coexist, the $Q_{\mathbf{A}}$ demand should be met first. That is why we include the term $(b_i^{\max} - \beta_{\mathbf{A}})$ in Eq. (12). Then, $R_{\mathbf{C}}$ flows can use only the residual bandwidth (as the best-effort policy is taken), so the meter-rate for $\hat{\mathcal{F}}_i^{\mathbf{C}}$ is set to

$$\beta_{\mathbf{C}} = b_i^{\max} - \beta_{\mathbf{A}} - \beta_{\mathbf{B}}. \tag{13}$$

Afterward, the Set_Queue operation is used to allot bandwidth to each flow in a subset. Take an example in Table IV(b), where $\hat{\mathcal{F}}_i^{\mathbf{A}} = \{f_1, f_2\}$, $\hat{\mathcal{F}}_i^{\mathbf{B}} = \{f_3\}$, $b_i^{\max} = 1\,\text{Gbps}$, $Q_{\mathbf{A}} = 80$, and $Q_{\mathbf{B}} = 60$. The meter-rate for $\hat{\mathcal{F}}_i^{\mathbf{A}}$ is $\min\{900\,\text{Mbps} + 600\,\text{Mbps}, 1\,\text{Gbps} \times 80\%\} = 800\,\text{Mbps}$. By Eq. (10), f_1 and f_2 can obtain bandwidth of $\frac{900}{900+600} \times 800\,\text{Mbps} = 480\,\text{Mbps}$ and $\frac{600}{900+600} \times 800\,\text{Mbps} = 320\,\text{Mbps}$, respectively. Then, the meter-rate for $\hat{\mathcal{F}}_i^{\mathbf{B}}$ is $\min\{500\,\text{Mbps}, 1\,\text{Gbps} \times 60\%, 1\,\text{Gbps} - 800\,\text{Mbps}\} = 200\,\text{Mbps}$. As $\hat{\mathcal{F}}_i^{\mathbf{B}}$ contains only f_3, it can get 200 Mbps bandwidth.

3) Pod-Layer Links: When a pod is congested, we select the busiest link l_i in the pod and adjust its flows. If the link carries only intra-pod flows (i.e., rule 1-2), we employ the methods in Sections IV-D1 and IV-D2 to eliminate congestion. However, if there exist inter-pod flows (i.e., rule 1-3), changing the subpath in the pod may also alter the subpath in the top layer. Fig. 1 shows as an example, where servers h_1 and h_9 are the source and destination, respectively. Suppose that the initial path is $\langle s_{13}, s_5, s_1, s_9, s_{17}\rangle$, and link (s_{13}, s_5) is busy. If we replace link (s_{13}, s_5) by link (s_{13}, s_6), the new path will be $\langle s_{13}, s_6, s_3$ (or s_4), $s_{10}, s_{17}\rangle$. Apparently, the change in the path is drastic, and some links may be burdened with heavy loads. In this case, we may have to also mitigate congestion in these links, which increases the controller's workload.

Instead, we choose one *substitute subpath* Υ_i in the pod (to replace l_i) without changing the residual subpath. For example, the substitute subpath for link (s_{13}, s_5) is $\langle s_{13}, s_6, s_{14}, s_5\rangle$. If Eq. (14) holds, we reroute l_i's packets via subpath Υ_i:

$$\sum\nolimits_{\forall l_j \in \Upsilon_i} \tau_j \leq \alpha \tau_i, \tag{14}$$

where τ_j is the average packet delay of link l_j (referring to the delay table $\Gamma_{\mathbf{D}}$) and $\alpha \in (0, 1]$ is a coefficient. From Eq. (14), even if the substitute subpath is longer, we can still reduce the average packet delay of flows, as l_i's packets can be sent by links of Υ_i in a shorter time due to their smaller loads.

E. Price Evaluating Module

Each switch counts the amount of bandwidth usage of flows via its ports. Then, the controller periodically gathers statistics from switches and calculates user fees by using this module, which takes account of both *basic fees* and *extra fees*.

For basic fees, we adopt the *pay-as-you-go (PAYG)* method [23]. Specifically, the basic fee of a flow f_j is estimated by

$$\varepsilon_j^{\text{BF}} = P_j \times \zeta_j, \tag{15}$$

where P_j is the unit price, which is set to $P_{\mathbf{A}}$, $P_{\mathbf{B}}$, and $P_{\mathbf{C}}$ if f_j's rank is $R_{\mathbf{A}}$, $R_{\mathbf{B}}$, and $R_{\mathbf{C}}$, respectively. Besides, ζ_j is the amount of f_j's bandwidth usage that the basic fee can cover, which depends on its rank. For the case of $R_{\mathbf{A}}$ (i.e., f_j is a $R_{\mathbf{A}}$ flow), ζ_j is the total bandwidth spent by f_j. For the case of $R_{\mathbf{B}}$, each $R_{\mathbf{B}}$ flow is given a quota $\Lambda_{\mathbf{B}}$ on using OF links at the basic fee. Let ζ_j^{EC} and ζ_j^{OF} be the amount of bandwidth that f_j uses on EC and OF links, respectively. Then, we set ζ_j to $\zeta_j^{\text{EC}} + \zeta_j^{\text{OF}}$ if $\zeta_j^{\text{OF}} < \Lambda_{\mathbf{B}}$ and $\zeta_j^{\text{EC}} + \Lambda_{\mathbf{B}}$ otherwise. For the case of $R_{\mathbf{C}}$, the basic fee covers merely the bandwidth usage on EC links, so we have $\zeta_j = \zeta_j^{\text{EC}}$.

If $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ users want to use OF links, they may have to pay extra fees. To avoid overcharging them, we take the PED model. Specifically, the relationship between user demand $\tilde{d}_i$ and price $\tilde{p}_i$ is expressed by $\tilde{d}_i = \lambda \tilde{p}_i^{-\nu}$, where λ is a scaling factor and ν is the price elasticity. Here, we derive that

$$\tilde{d}_2/\tilde{d}_1 = (\tilde{p}_1/\tilde{p}_2)^{\nu} \;\Rightarrow\; \nu = \frac{\ln(\tilde{d}_2/\tilde{d}_1)}{\ln(\tilde{p}_1/\tilde{p}_2)}. \tag{16}$$

From Eq. (16), a larger ν value implies that lowering price can raise user demand, so the service is more elastic. Afterward, we modify the pricing equations in [24], which takes the PED model into account, to calculate the extra fee:

$$R_{\mathbf{B}}\colon \varepsilon_j^{\text{EF}} = (P_{\mathbf{A}} + \nu_{\mathbf{B}}) \times (\tilde{e} - \tilde{e}^{-1}) \times (\zeta_j^{\text{OF}} - \Lambda_{\mathbf{B}}), \tag{17}$$

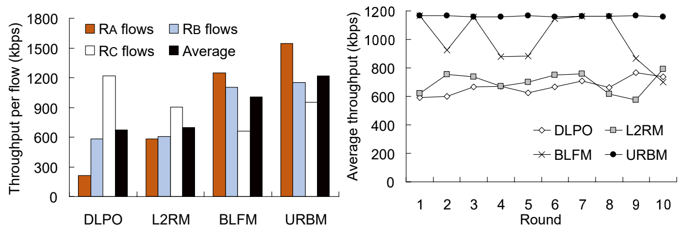
$$R_{\mathbf{C}}\colon \varepsilon_j^{\text{EF}} = (P_{\mathbf{A}} + \nu_{\mathbf{C}}) \times (\tilde{e} - \tilde{e}^{-2}) \times \zeta_j^{\text{OF}}, \tag{18}$$

where $\nu_{\mathbf{B}}$ and $\nu_{\mathbf{C}}$ denote the price elasticity of $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ users, respectively, and $\tilde{e}$ is the Euler's number. Note that the condition $\zeta_j^{\text{OF}} > \Lambda_{\mathbf{B}}$ holds in Eq. (17), or f_j's user need not pay the extra fee. Specifically, we take the unit price of $R_{\mathbf{A}}$ flows (i.e., $P_{\mathbf{A}}$) as the basis, and add additional prices $\nu_{\mathbf{B}}$ and $\nu_{\mathbf{C}}$ for $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ flows in Eqs. (17) and (18), respectively. Because $\nu_{\mathbf{B}} < \nu_{\mathbf{C}}$ and $\tilde{e}^{-1} > \tilde{e}^{-2}$, $R_{\mathbf{B}}$ users can pay less money than $R_{\mathbf{C}}$ users for using OF links to route their flows.

V. Performance Evaluation

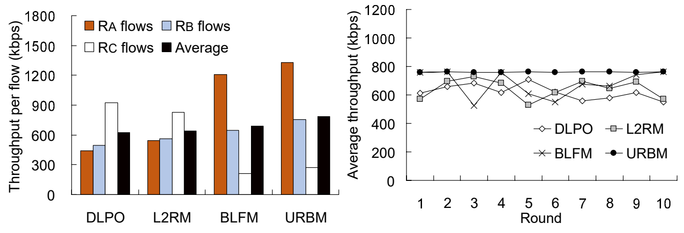
We adopt the Mininet simulator [25] to evaluate the system performance, where the controller and all switches are implemented by the Ryu framework [26] and Open vSwitch module [27], respectively. Fig. 1 gives the network topology, where the capacity of EC and OF links is set to 10 Mbps and 1 Gbps, respectively. We use the iPerf tool [28] to simulate the generation of flows. Each flow is a TCP connection that produces packets in a three-stage manner: (1) keep transmitting packets, (2) send packets every 10 seconds, and (3) send packets per 50 seconds. Flows may have different starting stages. Moreover, two scenarios are considered. In scenario I, we randomly pick 12 servers as sources. They will send packets to 4 servers in the same pod. Scenario I is used to make congestion in a pod. In scenario II, 2 servers are selected as sources, which send packets to all other servers. In this case, congestion will occur at the edge switch that connects with a source.

We compare URBM with two SDN-based methods mentioned in Section II. DLPO [19] transfers some flows of the top 10% busiest links to other links. L2RM [21] sends the packets of a flow via multiple paths by using the group table. Since DLPO and L2RM do not consider user ranks, we also propose a *bandwidth-limiting flow management (BLFM)* method for comparison. BLFM is similar to URBM. However, when links are busy, BLFM only limits bandwidth usage of flows (through the meter table and Set_Queue operation), without rerouting their packets to other paths. By comparing BLFM, we can evaluate the performance of changing routes in our URBM scheme. Each experiment is repeated 10 rounds, and we take their average. The simulation time is 1000 seconds.



(a) Flow throughput (b) Stability

Fig. 4. Comparison on throughput in scenario I.



(a) Flow throughput (b) Stability

Fig. 5. Comparison on throughput in scenario II.

A. Throughput in Scenario I

Fig. 4(a) gives flow throughput in scenario I, where flows will converge on the pod to which destinations belong (below, we call it the *dest-pod*). In this case, most links in the dest-pod are congested. DLPO and L2RM find substitute paths in the dest-pod to share the loads of congested links. However, since the substitute paths may be also busy, they cannot effectively help share loads. Thus, DLPO and L2RM have lower average throughput. BLFM uses the meter table and Set_Queue operation to restrict bandwidth usage of flows on the congested links, so it has higher average throughput than both DLPO and L2RM. Our URBM scheme not only finds substitute paths but also limits bandwidth usage of flows. Moreover, URBM adopts the method discussed in Section IV-D3 to deal with the case of a congested pod. In this way, URBM can have the highest average throughput among all methods. In particular, URBM improves 81.7%, 74.6%, and 21.2% of average throughput, as compared with DLPO, L2RM, and BLFM, respectively.

Let us observe the throughput of different ranks of flows in Fig. 4(a). DLPO and L2RM do not differentiate between high-rank flows and low-rank flows. Thus, $R_{\mathbf{C}}$ flows could consume more bandwidth when they choose to keep sending packets at the beginning. This leads to a strange problem, where low-rank users pay less but get more services. Specifically, $R_{\mathbf{C}}$ flows' throughput occupies 60.6% and 43.1% of total throughput in DLPO and L2RM, respectively. Both BLFM and URBM can conquer this problem by reserving $Q_{\mathbf{A}}$ and $Q_{\mathbf{B}}$ percents of link bandwidth for $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows, which provide better QoS supports for high-rank users. As compared with BLFM, URBM can further improve throughput for each rank of flows.

We evaluate the stability of each method. Fig. 4(b) compares the average throughput of different methods in each round. In BLFM, the same rank of flows may be transmitted via the same busy links, but their packets cannot be rerouted to other paths. Thus, BLFM's throughput is more choppy than others. DLPO and L2RM find substitute links in the dest-pod. Since these substitute links could be also busy, their average throughput will fluctuate. Thanks to the congestion eliminating module in Section IV-D, the average throughput in URBM can almost keep constant. In particular, the standard deviation of average throughput (measured in these 10 rounds) is 55.8, 73.1, 172.4, and 4.4 in DLPO, L2RM, BLFM, and URBM, respectively. This result verifies that our URBM scheme can stably improve flow throughput, as compared with others.

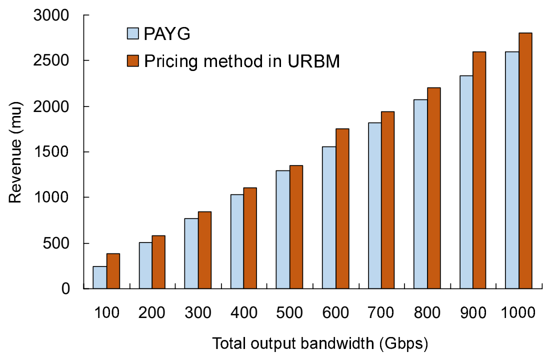


Fig. 6. Comparison on the operator's revenue.

B. Throughput in Scenario II

Fig. 5(a) shows flow throughput in scenario II, where two servers take charge of flow generation. Thus, congestion occurs at each edge switch attached by a source, and it is difficult to find substitute paths to replace congested links of the switch. Even worse, each congested link carries many flows, as each source selects all other servers to be its destinations. Thus, the effect of limiting bandwidth usage of flows by the meter table and Set_Queue operation may reduce. That explains why the throughput of each method reduces as compared with the result in Fig. 4(a). As compared with DLPO, L2RM, and URBM, our URBM scheme can improve 26.4%, 21.9%, and 13.6% of average throughput, respectively. Moreover, $R_{\mathbf{A}}$ flows have the highest throughput, followed by $R_{\mathbf{B}}$ and $R_{\mathbf{C}}$ flows, which shows that URBM can provide different QoS supports to users based on their ranks.

Fig. 5(b) then compares the average throughput of different methods in each round. As most flows concentrate in an edge switch that links to each source, the magnitude of fluctuation in average throughput of DLPO, L2RM, and BLFM will be smaller than that in Fig. 4(b). URBM's average throughput measured in 10 rounds is similar, which means that it can stably improve throughput. In particular, the standard deviation of average throughput in scenario II is 51.6, 67.5, 92.1, and 2.9 by DLPO, L2RM, BLFM, and URBM, respectively.

C. Revenue

Fig. 6 shows the operator's revenue in relation to the total output bandwidth, where the price unit per Gbps is denoted by the *monetary unit (mu)*. According to [23] and [24], we set

$P_{\mathbf{A}} = 2.56\,\mathrm{mu}$, $P_{\mathbf{B}} = 2.29\,\mathrm{mu}$, $P_{\mathbf{C}} = 1.80\,\mathrm{mu}$, $\nu_{\mathbf{B}} = 0.4\,\mathrm{mu}$, and $\nu_{\mathbf{C}} = 0.8\,\mathrm{mu}$. In the PAYG method, the revenue is almost proportional to the output bandwidth (i.e., linear growth). Our pricing method in Section IV-E calculates extra fees for low-rank users to employ OF links by Eqs. (17) and (18), thereby raising the revenue. Specifically, it can improve 9.43% of the revenue on average, as compared with PAYG.

VI. Conclusion and Future Work

In this paper, we propose the URBM scheme by using SDN to efficiently manage flows in a fat-tree DCN containing hybrid switches, where electrical switches form the network backbone and optical switches provide high-speed links. URBM divides users into three ranks and offers them different QoS supports. A portion of link bandwidth (i.e., $Q_{\mathbf{A}}$ and $Q_{\mathbf{B}}$) is reserved for $R_{\mathbf{A}}$ and $R_{\mathbf{B}}$ flows, and they can employ OF links to improve throughput. For $R_{\mathbf{C}}$ flows, we take the best-effort policy. Besides, their users have to pay extra fees for using OF links. To mitigate congestion, URBM reroutes packets to different paths or limits bandwidth usage of flows by adopting the tools in OpenFlow, including the group table, meter table, and Set_Queue operation. Based on user ranks and bandwidth consumption, our pricing method takes the PAYG method and the PED model to assess both basic and extra fees for users, respectively. Simulation results show that URBM can stably improve flow throughput, provide better QoS supports to high-rank flows, and increase the operator's revenue.

For future work, we will consider how to deal with impulse and DDoS flows, which generate many packets but last for a short while [29]. How to fast mitigate congestion caused by such transient flows is a challenge. Besides, it merits further investigation into flow management in a DCN with distributed SDN control [30]. In this case, multiple controllers divide the management work and how to balance their loads is critical.

Acknowledgment

This work was supported by the Ministry of Science and Technology, Taiwan under Grant 108-2221-E-110-016-MY3.

References

[1] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 640–656, 2017.

[2] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2324–2352, 2018.

[3] Y. C. Wang and H. Hu, "A Low-cost, high-efficiency SDN framework to diminish redundant ARP and IGMP traffics in large-scale LANs," in *IEEE Computer Software and Applications Conference*, 2018, pp. 894–903.

[4] Y. C. Wang and H. Hu, "An adaptive broadcast and multicast traffic cutting framework to improve Ethernet efficiency by SDN," *Journal of Information Science and Engineering*, vol. 35, no. 2, pp. 375–392, 2019.

[5] A. Minakhmetov, C. Ware, and L. Iannone, "TCP congestion control in datacenter optical packet networks on hybrid switches," *Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. 71–81, 2018.

[6] F. Yan, X. Xue, and N. Calabretta, "HiFOST: A scalable and low-latency hybrid data center network architecture based on flow-controlled fast optical switches," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. 1–14, 2018.

[7] Y. C. Wang and K. C. Chien, "EPS: Energy-efficient pricing and resource scheduling in LTE-A heterogeneous networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8832–8845, 2018.

[8] N. G. Mankiw, *Principles of Economics*. Cengage Learning, Inc., 2021.

[9] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with Karuna," in *ACM SIGCOMM Conference*, 2016, pp. 174–187.

[10] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *USENIX Symposium on Networked Systems Design and Implementation*, 2016, pp. 537–550.

[11] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *ACM SIGCOMM Conference*, 2017, pp. 225–238.

[12] A. Chatzieleftheriou, S. Legtchenko, H. Williams, and A. Rowstron, "Larry: Practical network reconfigurability in the data center," in *USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 141–156.

[13] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, "Improving flow scheduling scheme with mix-traffic in multi-tenant data centers," *IEEE Access*, vol. 8, pp. 64 666–64 677, 2020.

[14] D. Adami, S. Giordano, M. Pagano, and G. Portaluri, "A novel SDN controller for traffic recovery and load balancing in data centers," in *IEEE International Workshop on Computer Aided Modelling and Design of Communication Links and Networks*, 2016, pp. 77–82.

[15] J. Pang, G. Xu, and X. Fu, "SDN-based data center networking with collaboration of multipath TCP and segment routing," *IEEE Access*, vol. 5, pp. 9764–9773, 2017.

[16] Q. Tang, H. Zhang, J. Dong, and L. Zhang, "Elephant flow detection mechanism in SDN-based data center networks," *Scientific Programming*, vol. 2020, pp. 1–8, 2020.

[17] M. Zaher, A. H. Alawadi, and S. Molnar, "Sieve: A flow scheduling framework in SDN based data center networks," *Computer Communications*, vol. 171, pp. 99–111, 2021.

[18] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in *IEEE International Conference on Advanced Information Networking and Applications*, 2013, pp. 290–297.

[19] Y. L. Lan, K. Wang, and Y. H. Hsu, "Dynamic load-balanced path optimization in SDN-based data center networks," in *International Symposium on Communication Systems, Networks and Digital Signal Processing*, 2016, pp. 1–6.

[20] U. Zakia and H. B. Yedder, "Dynamic load balancing in SDN-based data center networks," in *IEEE Annual Information Technology, Electronics and Mobile Communication Conference*, 2017, pp. 242–247.

[21] Y. C. Wang and S. Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.

[22] Y. C. Wang and L. C. Yen, "Collaborative route management to mitigate congestion in multi-domain networks using SDN," in *IEEE Annual Computing and Communication Workshop and Conference*, 2022, pp. 988–994.

[23] Azure. [Online]. Available: https://azure.microsoft.com/en-us/pricing/

[24] Y. C. Wang and T. Y. Tsai, "A pricing-aware resource scheduling framework for LTE networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1445–1458, 2017.

[25] Mininet. [Online]. Available: http://mininet.org

[26] Ryu. [Online]. Available: https://ryu-sdn.org

[27] Open vSwitch. [Online]. Available: https://www.openvswitch.org

[28] iPerf. [Online]. Available: https://iperf.fr

[29] Y. C. Wang and Y. C. Wang, "Efficient and low-cost defense against distributed denial-of-service attacks in SDN-based networks," *International Journal of Communication Systems*, vol. 33, no. 14, pp. 1–24, 2020.

[30] W. K. Lai, Y. C. Wang, Y. C. Chen, and Z. T. Tsai, "TSSM: Time-sharing switch migration to balance loads of distributed SDN controllers," *IEEE Transactions on Network and Service Management*, pp. 1–13, 2022.