Cooperative Flow Management in Multi-domain SDN-based Networks with Multiple Controllers

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

En-Jui Chang
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
m073040019@student.nsysu.edu.tw

***Abstract*—The technology of *software-defined networking (SDN)* facilitates network management by adopting a controller to direct the operations of switches. This paper aims to efficiently manage flows in a *multi-domain SDN-based (MDS)* network, where each domain (i.e., subnetwork) is governed by a controller and there exist links between domains. To improve MDS performance, a *cooperative flow management (CFM)* framework is proposed in the paper. Each controller finds paths for the flows in its domain such that the loads of links can be balanced. When some links are still congested but there is no substitute path, the controller seeks help from neighboring domains. In this case, cross-domain paths will be constructed to replace congested paths. Since each controller finds paths based on only the local view of its own domain, CFM builds cross-domain paths in a cooperative, distributed manner. Through simulations, we verify that CFM can efficiently resolve congestion and significantly improve throughput.**

***Keywords*—congestion, flow management, multi-domain, path selection, software-defined networking (SDN).**

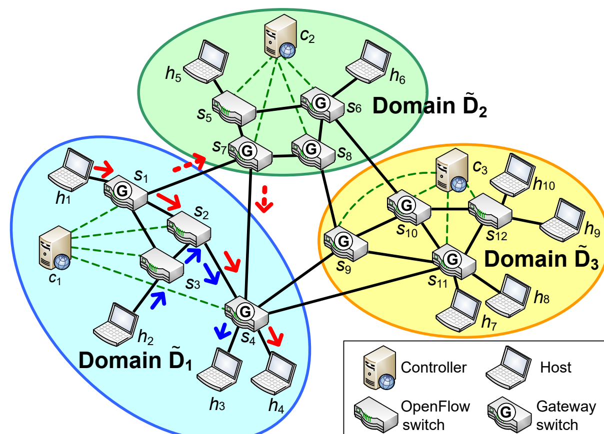


Fig. 1. An MDS network with three domains.

I. Introduction

A network can be divided into two parts. The *control plane* determines how packets are routed and the *data plane* is the actual forwarding process. In traditional networks, control and data planes both reside in switches, which makes it not easy for users to master traffic flows. *Software-defined networking (SDN)* gives facilitation and flexibility to network management by putting the control plane in a stand-alone *controller*. Thus, users can handily apply their policies to the network, such as dynamically changing routes [1] or discarding certain packets [2]. This can be accomplished by implementing programs on the controller, rather than configuring switches one by one.

SDN plays a key role in 5G and its successors, so one can expect that SDN will be widely used to manage large networks like campus and enterprise networks [3]. Such networks are usually divided into multiple subnetworks called *domains*, as a university or company is composed of many departments. Each domain is managed by a controller and operates autonomously. Since these domains belong to the same network, the cooperation among domains (e.g., path borrowing) is allowed and even encouraged. We call this type of network a *multi-domain SDN-based (MDS) network*. Fig. 1 presents an example, where the network has three domains, each ruled by a single controller. As compared with the case of using one controller to govern the whole network, using an MDS network has some benefits. First, each department has full autonomy in its own domain. Second, the load of each controller can reduce, since it controls only a domain instead of the whole network. Third, the failure of a controller merely affects one domain.

This paper proposes a *cooperative flow management (CFM)* framework to schedule paths of flows in an MDS network with the objective of mitigating congestion and raising throughput. Each controller monitors traffic loads in its domain and adjusts weights of links accordingly. Then, it finds a minimum-weight path for each flow in the domain to balance the loads of links. However, if some links are still overloaded but no alternative path can be found in its domain, the controller borrows some links from nearby domains. In this way, a *cross-domain path* can be built to replace the congested one. Since each controller relies on the local view of its domain to find paths (i.e., without the global knowledge of the overall network), the cross-domain paths are constructed in a cooperative and distributed fashion. Thus, our CFM framework is fit for MDS networks (i.e., there is no central control on all domains and each domain is also autonomous). Simulation results show that CFM can efficiently mitigate congestion and increase network throughput.

II. Related Work

Some studies consider using one SDN controller to manage flows. In [4], the controller keeps watching utilization of links.

If the utilization tops 80%, new traffic requests will be declined to avoid congestion. Liu et al. [5] find multiple paths to send the packets of elephant flows to prevent them from consuming much link bandwidth. The work [6] assigns a cost to each link. In case of congestion, it replaces the old path by a new path with the minimum link cost and traffic load. Lan et al. [7] detect congestion by estimating the variance of links' loads. Then, the flows on top 10% of busiest links are passed to other links whose utilization is low. The study [8] finds overloaded links by a load-deviation factor, and uses the OpenFlow group table to reroute their packets for load sharing. Breiki et al. [9] adjust the amount of traffics sent on a link through the meter table in OpenFlow. However, the above studies can be applied to only a single domain and may not be fit for MDS networks.

Some issues for MDS networks are also discussed. When a controller fails, [10] asks another controller to fast take over its job, so as to reduce the service interruption time of its domain. Hu et al. [11] divide a network into clusters (corresponding to domains) such that their sizes will be as balanced as possible [12]. In this way, the load of each controller can be similar. The work [13] picks a controller to be master, which takes charge of clustering other (slave) controllers and balancing their loads. If the master is broken, the study [14] picks a controller with the minimum load and high throughput to be the new master. When the controller in a domain is too busy, the work [15] moves some switches to other domains to reduce its load. As can be seen, none of them handles flow management in MDS networks. This motivates us to develop the CFM framework to dynamically adjust routes of flows, so as to mitigate congestion and increase MDS performance.

III. The Proposed CFM Framework

Let us consider an MDS network that consists of disjointed domains, as shown in Fig. 1. Each domain has some switches managed by a controller. The controller can use the *OFPPort-StatsRequest* function in OpenFlow to query each switch inside the domain about its status (e.g., load and topology). However, neither switch status nor link topology *outside* the domain is known by the controller. In other words, each controller has only the local view of its own domain.

There may exist links between two domains. If a switch has links to other domains, it is called a *gateway*. For example, s_1 and s_4 are the gateways of domain $\tilde{D}_1$. Switches periodically exchange Hello messages to update the neighboring relationship, of which their controllers are also notified [16]. In this way, a controller can know which switches serve as gateways and the exterior domains that they connect with.

Our CFM framework has three mechanisms to manage flows in the MDS network, including *information table maintenance*, *local path selection*, and *cross-domain path construction*. We then elaborate on each mechanism.

A. Information Table Maintenance

To keep updated with the status in a domain, the controller maintains four tables. Specifically, the *routing path table* stores the selected path for each pair of source and destination, which will be discussed in Section III-B. When both hosts are located in the domain, it stores a *complete* path. Otherwise, the path will start from the source to the gateway that connects to the exterior domain where the destination resides. Let us consider controller c_1 in Fig. 1. For hosts h_1 and h_4, it stores a complete path $h_1 \to s_1 \to s_2 \to s_4 \to h_4$. For hosts h_2 and h_6, the path will be $h_2 \to s_3 \to s_1$, as h_6 resides in domain $\tilde{D}_2$ which is linked by gateway s_1.

The *host access table* stores the information of hosts that a controller has learned. Each entry has a format of (a_i, s_i, p_i), where a_i is the host's IP address, s_i is the switch to contact the host, and p_i is s_i's port that connects to the host. Take h_2 as an example. Its entry will be $(10.0.1.2, s_3, 4)$, which means that h_2 (whose IP address is 10.0.1.2) is connected by s_3 via its port 4. If the host resides in an exterior domain, say, $\tilde{D}_x$, then s_i is the gateway that links to $\tilde{D}_x$, and p_i is s_i's port that connects to $\tilde{D}_x$. Take h_6 as an example, where its IP address is 10.0.2.6. Since h_6 is not in c_1's domain but in domain $\tilde{D}_2$, its entry is $(10.0.2.6, s_1, 5)$, where gateway s_1 uses its port 5 to link to domain $\tilde{D}_2$. Note that the controller can use the *subnet mask* to ascertain which domain an exterior host resides in.

The *switch load table* keeps track of the amount of traffics passing through a switch in the domain. The format of each entry is $(s_i, p_j, d_j^{\mathbf{T}}, d_j^{\mathbf{R}}, c_j)$, which means that for port p_j of switch s_i, $d_j^{\mathbf{T}}$ and $d_j^{\mathbf{R}}$ bytes of data have been sent and received in the last period, respectively, and p_j's capacity is c_j (in kbps). Here, the length of a period is set to 10 seconds. For example, an entry $(s_1, 1, 4871, 3043, 10000)$ indicates that there were 4871 and 3043 bytes sent and received by s_1's port 1 in the last 10 seconds, whose capacity is 10Mbps.

The *flow load table* helps the controller grasp the state of each flow in its domain. For a flow f_j, there is a corresponding entry $(a_j^{\mathbf{S}}, a_j^{\mathbf{D}}, \zeta_j^{\mathbf{T}}, \zeta_j^{\mathbf{R}}, \xi_j)$, where $a_j^{\mathbf{S}}$ is f_j's source, $a_j^{\mathbf{D}}$ is f_j's destination, $\zeta_j^{\mathbf{T}}$ is the number of packets sent by $a_j^{\mathbf{S}}$, $\zeta_j^{\mathbf{R}}$ is the number of packets received by $a_j^{\mathbf{D}}$, and ξ_j is the packet loss rate. Both $\zeta_j^{\mathbf{T}}$ and $\zeta_j^{\mathbf{R}}$ are measured during a period. Moreover, ξ_j is calculated as follows:

$$\xi_j = (1 - \zeta_j^{\mathbf{R}}/\zeta_j^{\mathbf{T}}) \times 100\%. \tag{1}$$

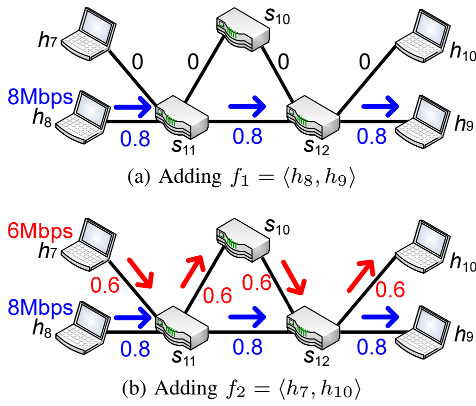
Suppose that host h_1 (whose IP address is 10.0.1.1) generates a flow to host h_3 (whose IP address is 10.0.1.3). An entry $(10.0.1.1, 10.0.1.3, 4822, 2259, 53\%)$ indicates that in the last 10 seconds, there were 4822 packets generated by h_1, but only 2259 packets were gotten by h_3. Thus, the packet loss rate is $(1 - 2259/4822) \times 100\% \approx 53\%$.

B. Local Path Selection

To reduce latency and alleviate congestion, we should select a path for each flow such that it has fewer links and these links are not burdened with heavy loads. To do so, each link l_k is assigned with a weight w_k, which is defined as follows:

$$w_k = \frac{1}{c_k} \sum\nolimits_{\forall f_j \in \hat{F}_k} (\zeta_j^{\mathbf{T}} \times \varepsilon)/t, \tag{2}$$

where $\hat{F}_k$ is the set of flows sent on l_k, ε is the packet size, and t is the period length. In Eq. (2), $\sum\nolimits_{\forall f_j \in \hat{F}_k} (\zeta_j^{\mathbf{T}} \times \varepsilon)/t$ is



(a) Adding $f_1 = \langle h_8, h_9 \rangle$

(b) Adding $f_2 = \langle h_7, h_{10} \rangle$

Fig. 2. Example of local path selection (a part of domain $\tilde{D}_3$).

Algorithm 1: Cross-domain path construction (case 2)

Data: $\tilde{D}_x$: local domain, $\tilde{D}_y$: neighboring domain

1. Let $\hat{F}_k$ be a set of flows on a congested link in $\tilde{D}_x$ and $\hat{F}'_k \subseteq \hat{F}_k$ contain all flows whose bandwidth consumption $\geq \delta_{\mathbf{B}}$.
2. Pick a flow $f_j = \langle h_u, h_v \rangle$ from $\hat{F}'_k$ with the least bandwidth consumption. If $\hat{F}'_k = \emptyset$, we pick a flow from $\hat{F}_k$ that consumes the most bandwidth.
3. Find the gateway $s_{\alpha 1}$ that is closest to h_u and build a path $\mathcal{P}_1$ from h_u to $s_{\alpha 1}$. Here, $s_{\alpha 1}$ is an outgoing gateway to $\tilde{D}_y$ whose incoming gateway is $s_{\beta 1}$.
4. In $\tilde{D}_y$, we find a path $\mathcal{P}_2$ from $s_{\beta 1}$ to an outgoing gateway $s_{\beta 2}$ back to $\tilde{D}_x$.
5. Let $s_{\alpha 2}$ be the incoming gateway in $\tilde{D}_x$ that links to $s_{\beta 2}$. Then, we find a path $\mathcal{P}_3$ from $s_{\alpha 2}$ to h_v.
6. Build a new path for f_j by concatenating $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$.

the total bandwidth consumption of the flows in $\hat{F}_k$, and c_k is l_k's capacity. The controller can refer to the routing path, switch load, and flow load tables to obtain the values of $\hat{F}_k$, c_k, and $\zeta_j^{\mathbf{T}}$, respectively.

When a new flow is generated (whose source and destination are h_u and h_v, respectively), the controller finds a path from h_u to h_v such that the sum of weights of its constituent links is minimized. In Section III-C, we will further discuss how to deal with the case when h_u and h_v reside in different domains. There have been several solutions to find a minimum-weight path. In particular, we adopt the popular Dijkstra's algorithm [17]. After finding the path, the routing path table is updated and the controller will notify all switches on that path.

Fig. 2 gives an example, where each link has a capacity of 10Mbps and its initial weight is 0. Suppose that an 8Mbps flow f_1 is sent from h_8 to h_9 (let us denote it by $f_1 = \langle h_8, h_9 \rangle$). The controller finds a path $h_8 \rightarrow s_{11} \rightarrow s_{12} \rightarrow h_9$ for f_1 and updates weights of links (h_8, s_{11}), (s_{11}, s_{12}), and (s_{12}, h_9) to $8/10 = 0.8$, as shown in Fig. 2(a). Then, a 6Mbps flow $f_2 = \langle h_7, h_{10} \rangle$ is added. Although the path $h_7 \rightarrow s_{11} \rightarrow s_{12} \rightarrow h_{10}$ has the minimum hop count, the controller picks an alternative path $h_7 \rightarrow s_{11} \rightarrow s_{10} \rightarrow s_{12} \rightarrow h_{10}$ for f_2, which bypasses a heavily loaded link (s_{11}, s_{12}), as shown in Fig. 2(b). In this way, we can balance the loads of links in a domain.

Once a flow transmits fewer packets and reduces bandwidth consumption, its path need not change but the weight of each link on that path will be recalculated. On the contrary, if the flow raises bandwidth consumption by producing more packets and makes some links overloaded (i.e., their weights $w_k \geq 1$), the controller uses the above mechanism to select another path in its domain to reroute the flow, so as to mitigate congestion.

C. Cross-domain Path Construction

Given a flow $f_j = \langle h_u, h_v \rangle$, there are two cases to trigger this mechanism: 1) h_u and h_v belong to different domains; 2) h_u and h_v reside in the same domain but its controller cannot find any internal path to replace congested links caused by f_j.

Case 1: Suppose that hosts h_u and h_v belong to domains $\tilde{D}_x$ and $\tilde{D}_y$, whose controllers are c_x and c_y, respectively. By the host access table, c_x realizes that h_v is an exterior host, so it chooses a gateway to $\tilde{D}_y$, say, s_α to be its representative. Afterward, c_x finds a path from h_u to s_α in domain $\tilde{D}_x$ based on the local path selection mechanism. Here, s_α is called an *outgoing gateway*. On the other hand, let s_β be the gateway in $\tilde{D}_y$ that is a neighbor of s_α, and we call it an *incoming gateway*. Since the source h_u is an exterior host, c_y picks s_β to stand for h_u and finds a path from s_β to the destination h_v. Therefore, a cross-domain path $h_u \rightarrow \cdots \rightarrow s_\alpha \rightarrow s_\beta \rightarrow \cdots \rightarrow h_v$ is built in a cooperative, distributed manner, as c_x and c_y do not know link topology outside their domains. Note that the controller that directs the outgoing gateway (i.e., c_x) takes charges of adding the link (s_α, s_β) to the path. Take Fig. 1 as an example. For flow $\langle h_2, h_5 \rangle$, controller c_1 picks a path $h_2 \rightarrow s_3 \rightarrow s_1 \rightarrow s_7$ for it. After that, controller c_2 builds the residual path $s_7 \rightarrow s_5 \rightarrow h_5$.

Case 2: When there are many flows or parts of flows have long paths in a domain, some links may become *bottlenecks*. Fig. 1 presents an example, where there are two flows $f_3 = \langle h_1, h_4 \rangle$ and $f_4 = \langle h_2, h_3 \rangle$ in domain $\tilde{D}_1$. As can be seen, link (s_2, s_4) is the bottleneck, as both flows must go through this link. Suppose that f_3 and f_4 consume 6Mbps and 8Mbps bandwidth, respectively, and each link has capacity of 10Mbps. Thus, the weight of link (s_2, s_4) will be (6+8)/10 = 1.4, which means that it is congested. However, we cannot find any other path inside domain $\tilde{D}_1$ to replace link (s_2, s_4) for f_3 or f_4. In this case, controller c_1 should seek help from nearby domains (e.g., $\tilde{D}_2$). More concretely, Algorithm 1 details our method for a controller to borrow paths from an exterior domain and build a cross-domain path to replace the congested one. For example, suppose that $\delta_{\mathbf{B}}$ = 1Mbps. Thus, $\hat{F}'_k = \{f_3, f_4\}$. As f_3 consumes less bandwidth than f_4, controller c_1 chooses to reroute f_3. Based on Algorithm 1, we can have $\mathcal{P}_1 : h_1 \rightarrow s_1$, $\mathcal{P}_2 : s_7$ (via different ports), and $\mathcal{P}_3 : s_4 \rightarrow h_4$. Thus, the new path for f_3 will be $h_1 \rightarrow s_1 \rightarrow s_7 \rightarrow s_4 \rightarrow h_4$.

In Algorithm 1, the reason why we select a flow in $\hat{F}'_k$ that has the least amount of bandwidth consumption for rerouting

TABLE I
GENERATION OF FLOWS IN THE SIMULATION.

flow	bandwidth	start time	end time	duration
$\langle h_3, h_2\rangle$	9Mbps	0	100	100
$\langle h_4, h_1\rangle$	6Mbps	40	130	90
$\langle h_5, h_6\rangle$	6Mbps	110	180	70
$\langle h_1, h_3\rangle$	9Mbps	135	300	165
$\langle h_2, h_4\rangle$	9Mbps	180	330	150
$\langle h_6, h_3\rangle$	6Mbps	260	400	140
$\langle h_3, h_1\rangle$	6Mbps	360	480	120
$\langle h_4, h_2\rangle$	9Mbps	420	580	160
$\langle h_9, h_5\rangle$	9Mbps	500	600	100

is that the controller has no idea about the status in the exterior domain $\tilde{D}_y$. If we simply reroute the largest flow (which may send volumes of data), its packets may cause congestion in $\tilde{D}_y$. However, if the congested link in the local domain $\tilde{D}_x$ carries many small flows, each using no more than δ_{B} bandwidth, we can reroute the one with the maximum bandwidth consumption to mitigate congestion in a more efficient manner.

After rerouting flow f_j by Algorithm 1, the controller will check if the packet loss rate of f_j (by referring to the flow load table discussed in Section III-A) can decrease. If not, it means that the borrowing path (from the exterior domain) would be also overloaded and thus cannot help reroute f_j. In this case, the controller will revert to the original path (i.e., the internal path in the local domain) for f_j.

IV. PERFORMANCE EVALUATION

We use the Mininet simulator [18] for performance evaluation. To enable OpenFlow in Mininet, switches and controllers are implemented by the OVS module [19] and the Ryu SDN framework [20], respectively. Fig. 1 shows the topology of the MDS network considered in our simulations, where each link has a capacity of 10Mbps. During the 600-second simulation time, there are nine flows generated, as listed in Table I.

We compare our CFM framework with two methods. One is the *local path selection (LPS)* method. It uses the mechanism in Section III-B to initialize and adjust paths of flows in each domain. The other method is the *traffic restrictions on heavily loaded links (TRHL)*. If a link is congested, the switch allocates bandwidth to each flow passing through that link based on its size. This technique is widely used in many studies to mitigate congestion or support QoS [6]–[9]. Since both LPS and TRHL are designed for the one-controller environment, we apply case 1 discussed in Section III-C to them to deal with the situation where the source and the destination of a flow are located in different domains (e.g., $\langle h_6, h_3\rangle$ and $\langle h_9, h_5\rangle$ in Table I).

Fig. 3 compares network throughput by the three methods. According to the generation of flows in Table I, there are three time periods that some links will encounter serious congestion:

- 70th–100th seconds: congestion is caused by two flows $\langle h_3, h_2\rangle$ and $\langle h_4, h_1\rangle$.
- 210th–300th seconds: congestion is caused by three flows $\langle h_1, h_3\rangle$, $\langle h_2, h_4\rangle$, and $\langle h_6, h_3\rangle$.
- 440th–480th seconds: congestion is caused by two flows $\langle h_3, h_1\rangle$ and $\langle h_4, h_2\rangle$.

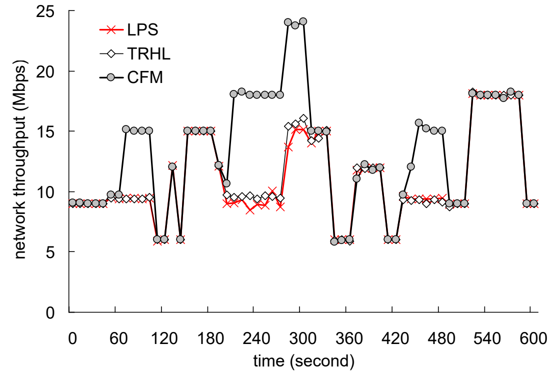
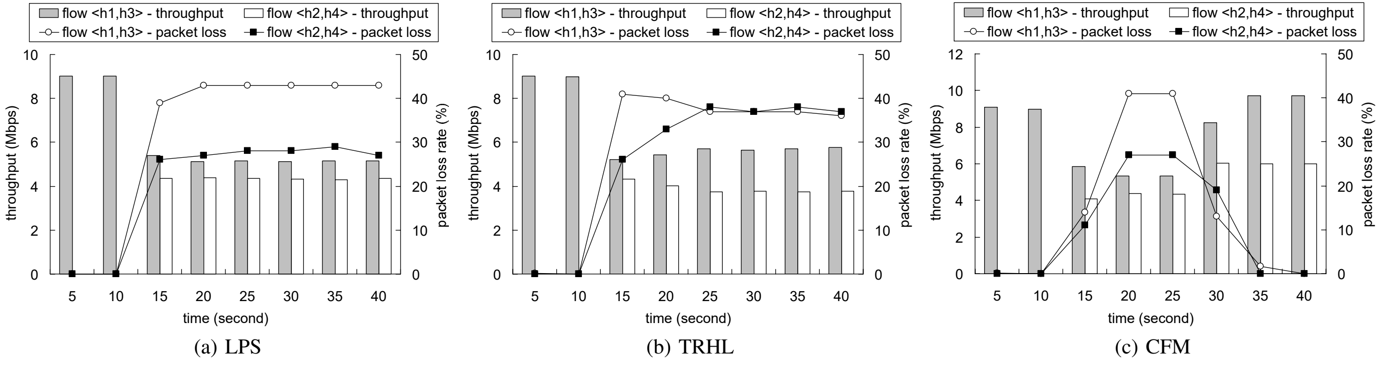


Fig. 3. Comparison on network throughput by the three methods.

When congestion occurs, LPS attempts to find other low-load links in the local domain to replace the congested links, while TRHL limits the amount of bandwidth that each flow can use on a congested link. However, since some bottleneck links in a domain may be congested by multiple flows, which makes LPS hard to find replaceable links, it has slightly lower throughput than TRHL (especially during 210th–300th seconds). On the contrary, CFM allows a controller to borrow some links from neighboring domains once it cannot find substitute links in its domain. In this way, the loads of congested links can be shared out, thereby significantly raising throughput. On the average, our CFM framework improves 21.45% and 20.46% of network throughput, as compared with LPS and TRHL, respectively.

To further investigate how each method deals with congestion on a link, we generate two flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ in domain $\tilde{D}_1$ and observe their throughput and packet loss. Flow $\langle h_1, h_3\rangle$ is produced in the beginning, while flow $\langle h_2, h_4\rangle$ is added on the 10th second. Based on the topology in Fig. 1, the initial paths for both flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ will be $h_1 \to s_1 \to s_2 \to s_4 \to h_3$ and $h_2 \to s_3 \to s_2 \to s_4 \to h_4$, respectively. Since flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ consume 9Mbps and 6Mbps, respectively, they will cause congestion on link (s_2, s_4). Fig. 4 shows the amount of throughput and the packet loss rate of each flow by the three methods. For LPS, flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ compete for the bandwidth of the congested link (s_2, s_4), so their throughput would be similar (in particular, 5.1Mbps and 4.3Mbps, respectively). In this case, flow $\langle h_1, h_3\rangle$ incurs a higher packet loss rate than flow $\langle h_2, h_4\rangle$ (i.e., around 43% and 28% after the 15th second, respectively). On the other hand, TRHL allocates bandwidth to both flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ proportionally to their sizes. Thus, the ratio of the throughput of flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ will be close to that of their sizes (i.e., 9:6). In this case, both flows also have similar packet loss rates (in particular, around 37% after the 25th second). In our CFM framework, when a link becomes congested but there is no substitute path in the domain, the controller will borrow links from a neighboring domain, as discussed in Section III-C. Thus, the throughput of each flow increases and its packet loss rate decreases after the 25th second. Both flows $\langle h_1, h_3\rangle$ and $\langle h_2, h_4\rangle$ will have no packet loss after the 35th second, which means that the



(a) LPS (b) TRHL (c) CFM

Fig. 4. Comparison on throughput and packet loss of two flows $\langle h_1, h_3 \rangle$ and $\langle h_2, h_4 \rangle$ in domain $\tilde{D}_1$ by the three methods.

congestion on link (s_2, s_4) has been completely resolved. This experiment shows the superiority of using cross-domain path construction to mitigate congestion in our CFM framework.

V. Conclusion

In this paper, we propose the CFM framework to efficiently manage flows in an MDS network which comprises connected but autonomous domains. Each controller maintains four information tables to keep track of the status of its domain and then finds a minimum-weight path for each flow to balance the loads of links. If some links are still congested but there is no substitute path in the domain, the controller borrows links from a nearby domain to reroute the flows on the congested links. Such cross-domain paths are constructed in a cooperative and distributed way, which assures the autonomy of each domain. Simulation results verify that our CFM framework can significantly increase network throughput and reduce packet loss, as compared with both LPS and TRHL. For the future work, we will consider managing flows with different priorities (e.g., based on their user levels [21]) in an MDS network. Moreover, how to make multiple domains cooperate to jointly detect and stop DDoS attacks [22] also deserves further investigation.

Acknowledgment

You-Chiun Wang's research is co-sponsored by the Ministry of Science and Technology under Grant No. MOST 108-2221-E-110-016-MY3, Taiwan.

References

[1] H. Xu, Z. Yu, X. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in SDNs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3073–3087, 2017.

[2] 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.

[3] Z. Zaidi, V. Friderikos, Z. Yousaf, S. Fletcher, M. Dohler, and H. Aghvami, "Will SDN be part of 5G?" *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3220–3258, 2018.

[4] M. F. Ramdhani, S. N. Hertiana, and B. Dirgantara, "Multipath routing with load balancing and admission control in software-defined networking (SDN)," in *International Conference on Information and Communication Technology*, 2016, pp. 1–6.

[5] J. Liu, J. Li, G. Shou, Y. Hu, Z. Guo, and W. Dai, "SDN based load balancing mechanism for elephant flow in data center networks," in *International Symposium on Wireless Personal Multimedia Communications*, 2014, pp. 486–490.

[6] 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.

[7] 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.

[8] 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.

[9] M. S. A. Breiki, S. Zhou, and Y. R. Luo, "Development of OpenFlow native capabilities to optimize QoS," in *International Conference on Software Defined Systems*, 2020, pp. 67–74.

[10] Y. C. Chan, K. Wang, and Y. H. Hsu, "Fast controller failover for multi-domain software-defined networks," in *European Conference on Networks and Communications*, 2015, pp. 370–374.

[11] T. Hu, P. Yi, J. Zhang, and J. Lan, "Reliable and load balance-aware multi-controller deployment in SDN," *China Communications*, vol. 15, no. 11, pp. 184–198, 2018.

[12] Y. C. Wang, W. C. Peng, and Y. C. Tseng, "Energy-balanced dispatch of mobile sensors in a hybrid wireless sensor network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1836–1850, 2010.

[13] A. Muthanna, A. Ateya, M. Makolkina, A. Vybornova, E. Markova, A. Gogol, and A. Koucheryavy, "SDN multi-controller networks with load balanced," in *International Conference on Future Networks and Distributed Systems*, 2018, pp. 1–6.

[14] W. H. F. Aly, "Controller adaptive load balancing for SDN networks," in *International Conference on Ubiquitous and Future Networks*, 2019, pp. 514–519.

[15] A. Filali, S. Cherkaoui, and A. Kobbane, "Prediction-based switch migration scheduling for SDN load balancing," in *IEEE International Conference on Communications*, 2019, pp. 1–6.

[16] 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.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.

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

[19] Open vSwitch (OVS). [Online]. Available: https://www.openvswitch.org

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

[21] 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.

[22] 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.