MaxPass: Credit-Based Multipath Transmission for Load Balancing in Data Centers

Minkyung Park, Sungmin Sohn, Kwangwook Kwon, and Ted Taekyoung Kwon

Abstract: **Various applications require the data center networks to carry their traffic efficiently. The data center networks usually have a hierarchical topology and exhibit distinct traffic patterns, which is different from the traditional Internet. These features have driven the data center networks to reduce the flow completion time (FCT) and to achieve high throughput. One of the possible solutions is balancing network loads across multiple paths by leveraging transport mechanisms like equal-cost multipath (ECMP) routing. ECMP allows flows to exploit multiple paths by hashing the metadata of the flows. However, due to the random nature of hash functions, ECMP often distributes the traffic unevenly, which makes it hard to utilize the links' full capacity. Thus, we propose an adaptive load balancing mechanism for multiple paths in data centers, dubbed MaxPass, to complement ECMP. A sender adaptively selects and dynamically changes multiple paths depending on the current network status like congestion. To monitor the network status, the corresponding receiver transmits a probe packet periodically to the sender; its loss indicates a traffic congestion. We carry out the quantitative analysis on the ns-2 simulator to show that MaxPass can improve the FCT and the throughput.**

Index Terms: **Data center, equal-cost multipath (ECMP), fat-tree topology, multipath transmission, transport layer protocol.**

I. INTRODUCTION

DATA centers are a key platform to enable enterprises to migrate applications to cloud systems successfully and improve the flexibility of their businesses. Since there are various types of applications running in the data centers such as cache follower [1], data mining, and web search, the data center networks may have diverse traffic patterns. First of all, there exists a various mix of short and long flows. Greenberg *et al.* [2] analyzed traffic patterns of a real-world data center that supports data mining. They reveal that the size of flows in the data center network varies from about 10 KB to about 1 MB and the short flows (under 10 KB) account for 78%, as summarized in Table 1. According to the analysis [2], most of the small flows are control messages which are essential to operating the data center network. Thus, the flow completion time (FCT) of a short flow should be minimized. However, since the traffic patterns of data center networks are unpredictable [3], it is difficult to carry the short flows more quickly than the long flows. As a consequence, the data center network has tried to improve the FCTs of the short flows and the throughput of the long flows [4].

Table 1. Flow size distribution of data mining applications in data center. Its average flow size is 7.41 MB.

Flow size	0–10 KB	10–100 KB	100 KB–1 MB	1 MB $\leq$
Percentage	78%	5%	8%	9%

Many studies [5]–[8] have tried to balance the traffic load among multiple links to satisfy the above requirements of data center applications by taking advantage of their network topology. Data centers usually use a Clos [9]-based topology such as fat-tree [10], which scales well in large data center environments. The Clos-based topology is hierarchical (or multi-layered) and multi-rooted, and hence provides multiple paths between a pair of hosts, the latter of which is used to balance the traffic across multiple links to avoid or mitigate the congestion. The primary technique to distribute flows across available paths is equal-cost multipath (ECMP) [11] routing. When a packet has arrived at a switch, it is forwarded to a next hop that is chosen based on a digest from hashing the five-tuple of its TCP/IP header. As ECMP is widely deployed to commodity switches, other protocols can leverage ECMP to use multiple paths without modifying the switches. For example, Raiciu *et al.* [5] adopt multipath TCP (MPTCP) [12] using ECMP in the data center and show its performance improvement.

Although ECMP helps to exploit the multiple path availability, it has shortcomings due to the nature of hash functions. Since hash collisions can occur even with different flows (i.e., different five tuples), it is hard to utilize full bandwidth of multiple links. Al-Fares *et al.* [13] show that ECMP utilizes 40–80% network bandwidth with substantial fluctuation. In addition, ECMP is not adaptive in the sense that it determines a path without considering the current status of links. That is, although there is a less congested path between two hosts, a flow between the two hosts might still choose the most congested path.

In this paper, we propose MaxPass, which is an adaptive load balancing mechanism, that complements ECMP to mitigate the above problems. MaxPass splits a single flow into multiple subflows; a subflow is defined as a stream of data packets along the same path between the sender and the receiver. MaxPass adaptively selects multiple different[1] paths to avoid congested links. The receiver continuously tries to select (and re-select) multiple paths (toward the sender) that are estimated to be less congested. To probe the congestion status of each path, we adopt the idea

Manuscript received January 17; approved for publication by Yu Hua, Division III, July 20, 2019.

This work was supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2016M3C4A7952587) funded by the Ministry of Science and ICT (MSIT) of Korea and BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF) (21A20151113068).

The authors are with the Department of Computer Engineering, Seoul National University, email: {mkpark, smsohn, kwkwon}@mmlab.snu.ac.kr, tkkwon@snu.ac.kr.

Ted Taekyoung Kwon is the corresponding author.

Digital Object Identifier: 10.1109/JCN.2019.000047

[1]Multiple paths can be partially or wholly overlapping depending on the network topology, number of subflows, and path re-selection, to be detailed later.

of sending a *credit packet* by the receiver in ExpressPass [14], which is one of credit-based congestion control mechanisms.

In ExpressPass, a receiver sends a small probing packet, called a credit packet, periodically to a sender. The receiver adjusts the sending rate of the credit packets by the ExpressPass feedback control algorithm. The feedback control algorithm determines the sending rate of the credit packets based on the number of dropped credit packets. The algorithm decreases the sending rate as the frequency of dropped credit packets increases. Otherwise, it increases the sending rate. Then, the sender sends as many data packets as received credit packets to the receiver. As each switch manages a credit queue and a data queue for each port, the credit queue mirrors the status of the data queue in the opposite direction for the sender. If there are many senders that share a switch port from the senders to the corresponding receivers, the credit queue of the opposite direction would be congested and some credit packets would be dropped. That is, any dropped credit packet indicates that the data queue is likely to be congested. As a result, the sending rate of the corresponding data packets is automatically limited by the rate of incoming credit packets. The details of ExpressPass will be explained in Section II.D.

In MaxPass, as the receiver sends credit packets over N different paths; at each path, credit packets are sent at the predetermined initial rate. As the sender sends a data packet over the corresponding path for each received credit packet, the receiver selects k (out of N) paths over which the data packets arrive earlier than the ones over the other $N-k$ paths. Through this process, we can determine relatively less congested k paths among N paths. The receiver adjusts the sending rate of the credit packets for each subflow to adapt to the network congestion. As the ExpressPass feedback control algorithm is too aggressive to adopt in the environments of multiple paths, we introduce a new feedback control algorithm to adjust the sending rate for each subflow, to be detailed in Section II. If the receiver concludes that credit packets are dropped too frequently, she concludes that the current k paths are too congested, and sends credit packets through N paths to re-select k subflows. In general, credit packets of the long flows may be more dropped than those of the short flows, which incurs the long tail distribution of the FCT. We will show that the adaptive path selection (i.e., path re-selection) also address this issue.

Our ns-2 simulator-based evaluation shows that MaxPass balances traffic across different links better than ExpressPass. Also, our evaluation with the realistic workload shows that MaxPass significantly reduces the FCT and improves throughput compared to MPTCP and ExpressPass.

The rest of the paper is organized as follows. Section II presents the background. In Section III, we detail the design of MaxPass. Section IV studies the performance of the MaxPass and discusses the simulation results. Section V illustrates the related works. We conclude the paper in Section VI.

II. BACKGROUND

A. Data Center Network Topology

Some data centers have up to hundreds of thousands of machines. To provide high bandwidth in a data center, its network topology typically has a hierarchical structure. Besides, since the traffic inside the data center may go back and forth between distant racks, the switches of upper layers can be more overloaded (and thus become bottlenecks) than those of lower layers. A most popular way to mitigate this issue is to have a topology of multiple roots like Clos topology [9].

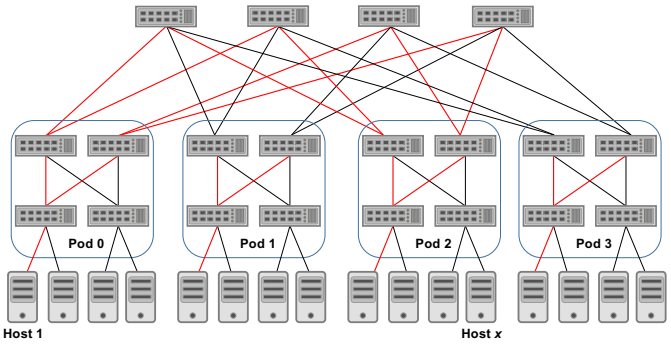


Fig. 1. An architecture of a 4-ary fat-tree topology is shown.

A fat-tree topology [10] is a representative example of a Clos topology. Fig. 1 shows an example of a 4-ary fat-tree. A fat-tree topology typically consists of three switch layers called core, aggregation, and edge layers from top to bottom. For a k-ary fat-tree, there are k pods, each containing two layers of $k/2$ switches each. Each switch in the lower layer (i.e., edge switch) is directly connected to hosts and $k/2$ switches in the upper layer (i.e., aggregation switch). Besides, each switch in the upper layer is connected with the core switches. There are $(k/2)^2$ core switches. Each core switch has k ports, each of which is connected to one of k pods. That is, the ith port of any core switch is connected to pod i. In general, a pair of hosts belonging to different pods has $(k/2)^2$ different paths, which is the number of the core switches. On the other hand, each pair of hosts in the same pod but not in the same edge switch has $(k/2)$ different paths. As shown in Fig. 1, the fat-tree with 4-ary has 4 core switches, 8 aggregation switches and 8 edge switches. In Fig. 1, there exists only one path from Host 1 to each core switch, and one path from each core switch to Host x. Therefore, there are at most 4 different paths between Host 1 and Host x, each of which passes through a different core switch.

B. Multipath Routing

In a dense interconnection topology adopted by a data center, multiple parallel paths exist between each pair of hosts for link availability and bandwidth efficiency. To spread traffic evenly over these paths, a multipath routing technique such as ECMP [11] is required. ECMP chooses a path for a flow from the hash result of the five-tuple of its packet header, which consists of source IP, destination IP, source port, destination port, and protocol fields. As a result, ECMP splits the traffic among multiple paths and potentially increases their bandwidth efficiency. Also, ECMP is easily adoptable since most of commodity switches support such functionality. However, since ECMP relies on the deterministic hash results, a path is statically determined regardless of whether or not the path is congested. Consequently, congestions occur frequently even in network under-

utilization if hash collisions occur between different flows. Furthermore, current ECMP implementations limit the multiplicity of paths to 8–16, which is sometimes less diverse than required to deliver high bisection bandwidth for a larger data center [10].

C. Multipath Transport Protocol

MPTCP [12] is one of the multipath transport protocols that allows a single TCP connection to use multiple paths when sending or receiving data. MPTCP establishes a connection with a three-way handshake like TCP except that its initiation packet enables the MP_CAPABLE option informing a recipient that its sender is capable of MPTCP and carries the keys used to authenticate the endpoints. Additional subflows can be opened after the three-way handshake. Thus a client can advertise its additional IP addresses by sending TCP packets to the server, and then its subflows can be opened. A newly created subflow will be linked to the first connection after its authentication. These subflows are linked together inside a single MPTCP connection, and send or receive data. By repeating the process, subflows can be created additionally using additional IP addresses or different ports with the same IP address pair. In the former case, the path is determined by the combination of source IP and destination IP. In the latter case, ECMP routing should be able to route subflows over different paths.

D. Credit-Based Congestion Control

Hop-by-hop credit-based congestion control is introduced for ATM networks [15]. Before sending a data cell (i.e., a short fixed-size data packet) to the receiver, the sender needs to receive a credit cell from the receiver. Receiving a credit cell means that the sender sends a data cell to the receiver without loss. Since the sender and the receiver are adjacent switches, there is only one link between the sender and the receiver. Thus, the credit-based congestion control algorithm works well for the hop-by-hop congestion control. To extend the hop-by-hop control to end-to-end settings, the data packet should be delivered over the same (but in the opposite direction) path of the credit packet. That is, the path should be symmetric.

Cho *et al.* [14] recently proposed ExpressPass to leverage credit-based congestion control in data center networks. Like [15], a sender can send a data packet only after receiving a credit packet from its receiver. One of the differences from [15] is that the sender and the receiver are not switches but hosts. Since the data center networks are under control of its administrator, ECMP in ExpressPass makes a data packet to be delivered over the same path (but in the reverse direction) for the corresponding credit packet. Moreover, they assign each switch port two queues: One is for data packets, and the other is for credit packets. Note that the credit queue size is much smaller than the data queue size since the size of a credit packet is much smaller than that of a data packet.

In ExpressPass, since a data packet follows the same (but in the reverse direction) path of the received credit packet, the credit queue (i.e., the direction from the receiver and the sender) reflects the status of the corresponding data queue (i.e., the direction from the sender and the receiver). In this way, the sending rate of data packets (by the sender) is limited by the sending rate of credit packets (by the receiver). If a congestion occurs in the credit queue, credit packets will be dropped due to the credit queue overflow, which decreases the number of arrived credit packets at the sender. Consequently, the sending rate of the data packets will be decreased.

Algorithm 1 ExpressPass feedback control algorithm.

1: $w = w_{init}$
2: $cur\_rate = initial\_rate$
3: **repeat** per update period (RTT by default)
4: $\quad credit\_loss = \#\_credit\_dropped/\#\_credit\_sent$
5: $\quad$ **if** $credit\_loss \leq target\_loss$ **then**
6: $\quad\quad$ (increasing phase)
7: $\quad\quad$ **if** previous phase was increasing phase **then**
8: $\quad\quad\quad w = (w + w_{max})/2 \quad (w_{max} = 0.5)$
9: $\quad\quad$ **end if**
10: $\quad\quad cur\_rate = (1 - w) \times cur\_rate$
11: $\quad\quad\quad + w \times max\_rate \times (1 + target\_loss)$
12: $\quad$ **else**
13: $\quad\quad$ (decreasing phase)
14: $\quad\quad cur\_rate = cur\_rate \times (1 - credit\_loss)$
15: $\quad\quad\quad \times (1 + target\_loss)$
16: $\quad\quad w = max(w/2, w_{min})$
17: $\quad$ **end if**
18: **until** End of flow

The receiver adjusts the credit sending rate based on a credit feedback algorithm. As each credit packet has a unique sequence number, the sender copies the corresponding number to the data packet. Thus, if a credit packet is dropped, the receiver can detect its loss from the sequence number in the received data packet. If credit packets are lost continuously, the receiver will decrease the credit sending rate based on the algorithm. Otherwise, it maintains or increases the rate. The ExpressPass feedback control algorithm is described in Algorithm 1. It uses an aggressiveness factor w, which adjusts the credit sending rate (cur_rate) between the current rate and the maximum rate. That is, w in the feedback control algorithm strikes a balance between stability and fast convergence. If the w value is large, it ramps the current rate up more aggressively. If w is small, it conservatively adjusts the rate. For each update period (line 3), it computes the credit loss $credit\_loss$ (line 4). The credit-based congestion controls accept some credit losses because the credit loss does not affect the data loss. Therefore, if $credit\_loss$ is lower than $target\_loss$, it goes to the increasing phase (lines 5–11). Otherwise, the decreasing phase is entered (lines 12–18). When the increasing phase happens twice in a row (line 7), it concludes that the network is severely underutilized. Therefore, it increases the aggressiveness factor (line 8). The increasing phase increases the credit sending rate as a result (lines 10 and 11). If $credit\_loss$ is larger than $target\_loss$, ExpressPass concludes congestion and goes to the decreasing phase. The algorithm decreases the credit sending rate using the credit loss and target loss values (lines 14–15). Also, the aggressiveness factor w becomes the half of the current value (line 16). The feedback control algorithm of ExpressPass aggressively increases the credit sending rate since it does not result in data losses.

ExpressPass is designed to use credit packets to control the

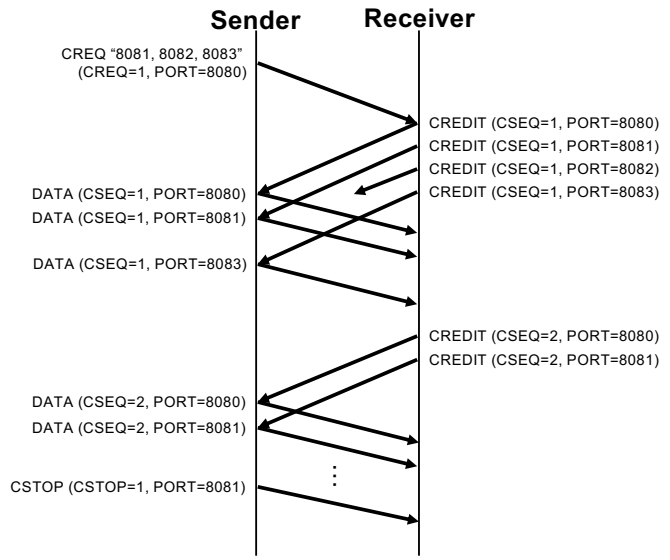


Fig. 2. An example of the flow of MaxPass is illustrated. CSEQ, CSTOP, and PORT in parenthesis are the fields of packet headers in MaxPass. Especially, PORT refers the port number of the sender-side.

sending rate of the data packets only for a single path, which cannot utilize the multipath diversity of the data center topology. Suppose that the ExpressPass feedback control algorithm is used for each subflow. Since credit packets are sent until the sender sends a control packet (`CREDIT_STOP`) to stop sending credit packets, the credit packets will be wasted if there are no pending data packets. The number of wasted credit packets will increase in proportion to the number of the subflows. The waste of credit packets eventually degrades the network-wide throughput, especially the performance of long flows. Moreover, ExpressPass does not consider how to select less congested paths. In the next section, we explain how MaxPass extends ExpressPass in such a way that multiple subflows will be delivered over less congested paths.

III. MAXPASS

A. Design Overview

MaxPass is a multi-path credit-based transport mechanism for load balancing. It initially sets up k paths between a sender and a receiver by probing all the paths since the knowledge of the global link state is unavailable (Section III.C). MaxPass leverages credit packets to select k relatively less congested paths. Moreover, it adaptively changes k paths as the link status varies over time (Section III.D). Finally, MaxPass employs a feedback control algorithm to adjust the credit sending rate, which in turn adjusts the data sending rate (Section III.E). We assume that MaxPass is used for all hosts in the data center network.

In the initialization phase, the sender sends a credit-request packet $CREQ$ to the receiver to inform that the sender has data to send. The sender selects N available ports that are embedded into $CREQ$, where N is the number of core switches (to be detailed later). After receiving the credit-request packet, the receiver can figure out N ports; one is from the source port field in the TCP header, and the others are embedded in $CREQ$. As the receiver and the sender assign a port to each of N flows, the receiver can build N subflows using N different ports. Note that the receiver uses a single port number. The receiver sends a credit packet ($CREDIT$) for each N subflow by the feedback control algorithm in response. A value in the credit sequence field $CSEQ$ increases by one per subflow. The sender sends one data packet for one received credit packet. The data packet should contain the same $CSEQ$ value as the one in the received credit packet.

As the initialization phase is over, the receiver measures which paths have shorter delays than the other paths. That is, he chooses k best paths out of N ones, where k is a system parameter. Thus, he stops sending credit packets through the other $N-k$ paths. This way, the receiver uses only k paths by continuously sends credit packets over the k paths. Finally, when the sender has no more data to send, she sends a credit-stop packet $CSTOP$ to the receiver. When the receiver receives the credit-stop packet from the sender, it stops sending the credit packets for all the subflows. Fig. 2 illustrates the initialization phase of MaxPass, where N is four and k is two. The $CREDIT$ packet for $CSEQ = 1$ and $PORT = 8082$ is dropped.

We assume that the switch design is same to that of ExpressPass. Each switch port manages a credit queue and a data queue. For instance, the size of a credit packet is set to the minimum Ethernet frame size (say, 84 bytes). Each credit packet can trigger the sender to send a data packet with a maximum Ethernet frame size (say, 1538 bytes). Thus, each credit queue and data queue need to accommodate $84/(84+1538) \approx 5\%$ and for 95% of traffic, respectively. The data packet should follow the same (but in the reverse direction) path of the corresponding credit packet so that the credit queue mirrors the data queue in the opposite direction. Since the data packet and the corresponding credit packet have the same five-tuple, which results in the same hash value, ECMP allows the data packet and the credit packet to pass through the same switches.

B. Credit Request

During the credit request, the sender selects N port numbers (as described in the above). Note that since N is the number of core switches, N is the maximum different paths between two hosts. To utilize all the available paths by ECMP, the sender is required to find N different port numbers whose hash values are mapped to N different paths. The data center network is private and controllable. The assumption that the entire nodes including hosts and switches can share a single hash function for ECMP is feasible. Every switch needs to share the hash function and parameters so that the receiver receives the packets over the different but consistent paths. For example, in the 4-ary fat-tree topology, every edge or aggregate switch can have four hash buckets. Each switch should arrange the ith bucket to hold packets that are forwarded to the ith core switch.

C. Path Probing

ECMP enables the receiver and the sender to select paths by choosing port numbers. Although it is important to select fast and less-congested paths, it is hard to keep track of which path

is not congested. The data center network accommodates high-speed links and high bandwidths, and its traffic is bursty and unpredictable. Therefore, the links and the switch queues are changing too dynamically for a centralized entity to monitor and reflect their states to (re-)routing and load balancing.

MaxPass employs a decentralized approach when to probe the congestion status of individual paths by exploiting the credit packets. The credit packets traverse links and switches. Some of them may be dropped in the switches or experience queueing delays in some switches. The loss or delay of a credit packet implies that the corresponding data packet (in the reverse direction) may experience congestion. If these "congested" paths are selected, it is likely for the data packets to suffer from congestion. Therefore, MaxPass selects paths over which the credit packets arrive earlier.

After receiving the credit request $CREQ$, the receiver sends credit packets and receives data packets corresponding to the credit packets over its N ports. As the receiver can measure the round trip time (RTT) for each path using the data packet and the corresponding credit packet with the same credit sequence number, she prefers the path with the smaller RTT. Finally, the receiver selects k paths with the smaller RTTs.

There may exist less than N different paths between the sender and the receiver. For example, when the sender and the receiver are in the same pod of the 4-ary fat tree topology, there are two available paths. In this case, two different subflows will go through the same path. Note that MaxPass still utilizes all the possible paths.

Algorithm 2 MaxPass feedback control algorithm.

1: $w = w_{init}$
2: $cur\_rate = initial\_rate$
3: $(w_{max} = 0.5,\ n = \#\_sub\_flows)$
4: **repeat** per update period (RTT by default)
5: $\quad credit\_loss = \#\_credit\_dropped/\#\_credit\_sent$
6: $\quad$ **if** $credit\_loss \leq target\_credit\_loss$ **then**
7: $\quad\quad$ (increasing phase)
8: $\quad\quad$ **if** previous phase was increasing phase **then**
9: $\quad\quad\quad w = (w + w_{max})/(2 \times n^2)$
10: $\quad\quad$ **end if**
11: $\quad\quad cur\_rate = (1 - w) \times cur\_rate$
12: $\quad\quad\quad\quad + w \times max\_rate \times (1 + target\_loss)$
13: $\quad$ **else**
14: $\quad\quad$ (decreasing phase)
15: $\quad\quad$ **if** $\#\_decrease > t$ **then**
16: $\quad\quad\quad \#\_decrease = 0$
17: $\quad\quad\quad$ Path Re-selection
18: $\quad\quad$ **else**
19: $\quad\quad\quad cur\_rate = cur\_rate$
20: $\quad\quad\quad\quad \times(1-credit\_loss)\times(1+target\_loss)$
21: $\quad\quad\quad cur\_rate = min(cur\_rate/2, min\_rate)$
22: $\quad\quad\quad w = max(w/2, w_{min})$
23: $\quad\quad$ **end if**
24: $\quad$ **end if**
25: **until** End of flow

D. Adaptive Path Selection

After the initialization, the sender and the receiver use k subflows using path probing. Suppose that a new flow is initiated by another pair of a sender and a receiver. If N is not substantially greater than k, some of subflows of the new flow may share their paths with those of the existing flow(s). Such dynamics of flows may degrade the throughput of the flows since they will be interfering with each other. Therefore, MaxPass introduces path re-selection to cope with congestion due to competing flows. The degree of congestion can be estimated using the feedback control algorithm. The feedback control algorithm increases the credit sending rate or decreases the rate opportunistically. If the rate decrease happens t times consecutively at the same subflow, the receiver starts the path re-selection. It is similar to the initialization phase since the receiver behaves as if it receives $CREQ$. It sends credit packets over N different ports.

E. Feedback Control Algorithm

We design a new feedback control algorithm that controls the credit sending rate for each subflow. The ExpressPass feedback control algorithm might be used for each subflow of MaxPass. However, we should consider the case in which a link is shared by different number of subflows. For instance, a link can be shared by two subflows of flow 1 and one subflow of flow 2. The feedback control algorithm of MaxPass is inspired by EWTCP [16] in the sense that MaxPass seeks to give an equal share of a link capacity to flows (not subflows). Moreover, the ExpressPass feedback control algorithm aggressively increases and conservatively decreases the credit rate (and hence data rate). Although credit losses do not result in data losses directly, the waste of credit packets may eventually degrade the entire throughput. In contrast, our feedback control algorithm tries to reduce congestions due to credit packets compared to ExpressPass. As the sending rate of credit packets increases and approaches the pre-determined maximum rate (*max_rate* in Algorithm 2), the MaxPass feedback control algorithm reduces the amount of increased sending rate gradually.

Algorithm 2 details the feedback control algorithm of MaxPass. Except for the re-selection parameter t, other parameters are the same as those of ExpressPass. In the increasing phase, the aggressiveness factor w is moderately increased (lines 8–9). Also, the decreasing phase halves the credit sending rate (line 21) to mitigate the congestion by reducing the credit packets.

F. Credit Stop

The credit stop message $CSTOP$ is used to terminate all the subflows. However, the sender can terminate a single subflow by sending a credit stop message $SUBCSTOP$ as MPTCP. The subflow termination can be used for the sender to change the set of paths for the subflows. To differentiate the termination of a single subflow and the termination of all the subflows, different fields should be used.

After the sender sends the credit request packet, the sender can set $SUBCSTOP$ in the data packet for a specific subflow. Then the receiver excludes the subflow from the k subflows.

Table 2. Flow size distribution of the two real-world workloads are classified.

Workload	Web Search	Cache Follower
0–10 KB (S)	15%	50%
10 KB–100 KB (M)	38%	3%
100 KB–1 MB (L)	17%	18%
Over 1 MB (XL)	30%	29%
Average flow size	1.6 MB	701 KB

IV. EVALUATION

In this chapter, we analyzed the performance of MaxPass with a focus on (1) throughput, (2) flow completion time (FCT), and (3) load balancing. We conducted a quantitative comparison with MPTCP [12], which is the most representative multipath transport scheme, and ExpressPass [14], which is the single path credit-based scheme that inspires MaxPass.

We simulated all the three schemes under ns-2 simulator [17]. For a medium-scale data center topology, all the three metrics are measured under a 4-ary fat-tree topology as illustrated in Fig. 1. The 4-ary fat-tree topology consists of 4 core switches, 8 aggregation switches, 8 edge switches, and 48 hosts (i.e., 6 hosts for each edge switch). For a large-scale data center topology, the throughput and FCT are measured in a 8-ary Clos-based topology. The large-scale topology consists of 8 core switches, 32 aggregation switches, 32 edge switches, and 256 hosts (i.e., 6 hosts for each edge switch). Every link in the topology has a speed of 10 Gbps. Link propagation delays and host delays are set to 4 μs and 1 μs, respectively.

The size of a credit buffer has a significant impact on the performance. A large credit queue can cause the queuing delay, which lags the feedback of congestion. On the other hand, a small credit queue drops credit packets unnecessarily, which may hinder the delivery of data packets, and consequently under-utilizes the network. For our evaluation, the credit buffer accommodates up to eight credit packets, which is proved to be sufficient in ExpressPass. The size may have to be coordinated for other environments.

To make realistic environments for data center networking, we use two real-world workloads. One is the web search workload [18], where a query is sent to many aggregators and workers, and merged later. The other is the cache follower workload [19], where cache followers forward and write messages to a single leader. Table 2 shows the flow size distribution and the average flow size for the web search and cache follower workloads. Each workload covers a wide range of average flow sizes ranging from less than 10 KB to more than 1 MB. While the web search workload has a similar fraction of 1 MB or higher size flows compared to the cache follower one, the size of large flows is substantially high, resulting in the higher average flow size. Each run generates 10,000 flows based on the distribution of each workload. The interval between two consecutive arrivals follows the Poisson distribution so that the average link utilization is 60%.

A. Throughput

We measure the throughput for each flow. As ExpressPass and MaxPass are the credit-based congestion control algorithms, the waste of credit packets may degrade the throughput of the network. Besides, as the feedback control algorithm of MaxPass is less aggressive than ExpressPass, the experiments are carried out to show that the feedback control algorithm of MaxPass does not harm the throughput. The comparison with MPTCP will show that the merits of the credit-based congestion control are still kept.

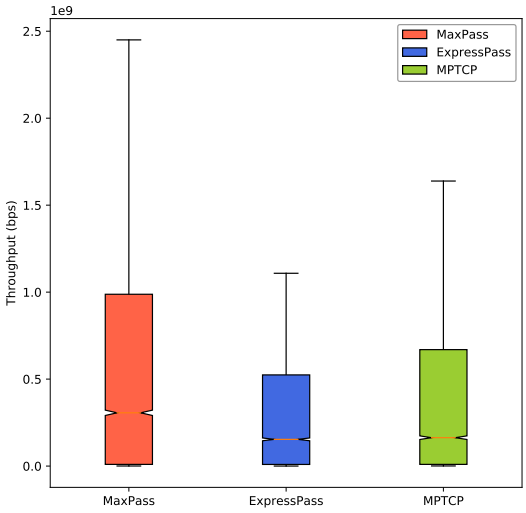


Fig. 3. Throughput of each flow in the medium-scale data center topology is plotted using box plot for the Cache Follower workload.

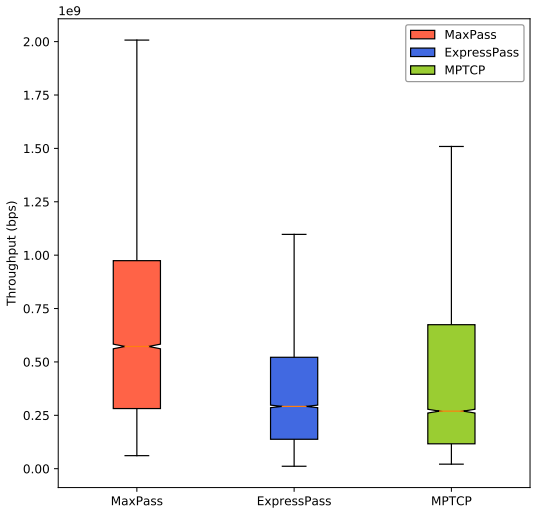


Fig. 4. Throughput of each flow in the medium-scale data center topology is plotted using box plot for the Web Search workload.

Figs. 3 and 4 show the throughput of each flow in the medium-scale data center network topology as box plot for the Cache Follower and Web Search workloads, respectively. They show that the overall metrics for the box of MaxPass are higher than others. Figs. 5 and 6 show the same tendency in the large-scale data center network. Therefore, MaxPass achieves the higher throughput than ExpressPass and MPTCP.

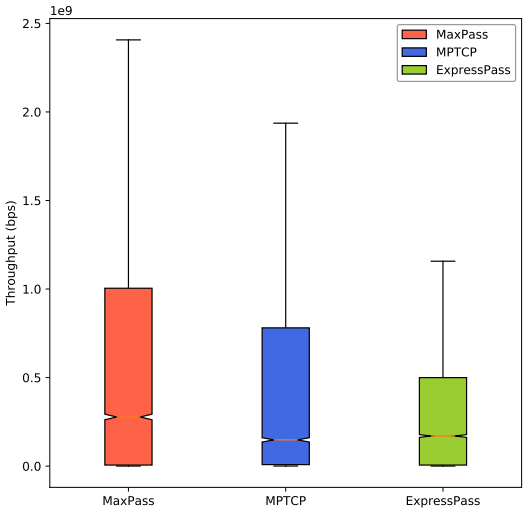


Fig. 5. Throughput of each flow in the large-scale data center topology is plotted using box plot for the Cache Follower workload.

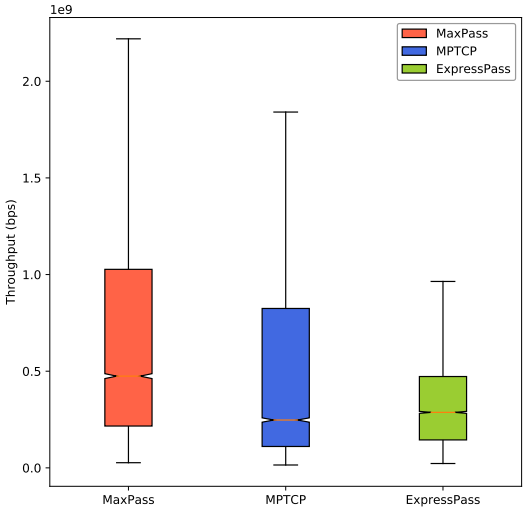


Fig. 6. Throughput of each flow in the large-scale data center topology is plotted using box plot for the Web Search workload.

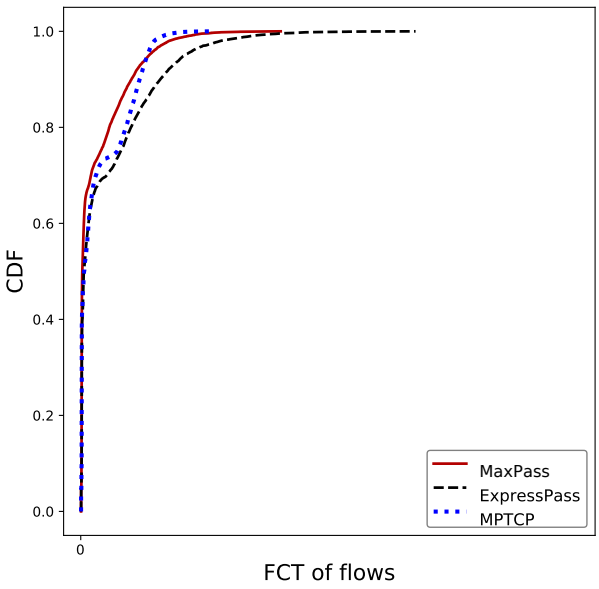


Fig. 7. Flow completion time of each flow in the medium-scale data center network topology is shown for the Cache Follower workload using CDF. Notice that the plots of MaxPass are located left compared to those of MPTCP and ExpressPass, which shows the shorter flow completion times.

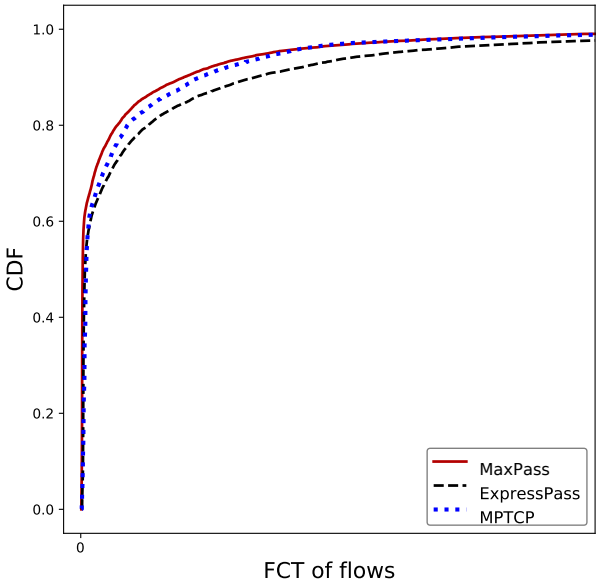


Fig. 8. Flow completion time of each flow in the medium-scale data center network topology is shown for the Web Search workload using CDF. The plots of MaxPass are located left compared to those of ExpressPass and similar to those of MPTCP.

B. Flow Completion Time (FCT)

FCT is an important metric in data center networks because most of its traffic is composed of short flows. As explained before, the short flows are usually control messages, which should be delivered promptly. Most of the workload is attributed to short flows. Therefore, FCT will show whether MaxPass can accommodate the data center traffic well.

Figs. 7 and 8 show the FCT of each flow in the medium-size data center network topology as the CDF for the Cache Follower and Web Search workloads, respectively. The plots of MaxPass are located left compared to the others, which means that FCTs are shorter than the others. Figs. 9 and 10 reach the same conclusion in the large-scale data center network for the Cache Follower and Web Search workloads, respectively. As the credit-based congestion control inherently avoids queuing in the switch, the flows of MaxPass and ExpressPass suffer from little queuing delay. Note that MaxPass and MPTCP use underutilized paths by distributing traffic loads.

C. Load Balancing

To check whether the load is evenly balanced, we measure the link load as bytes for each link per unit time in the medium-

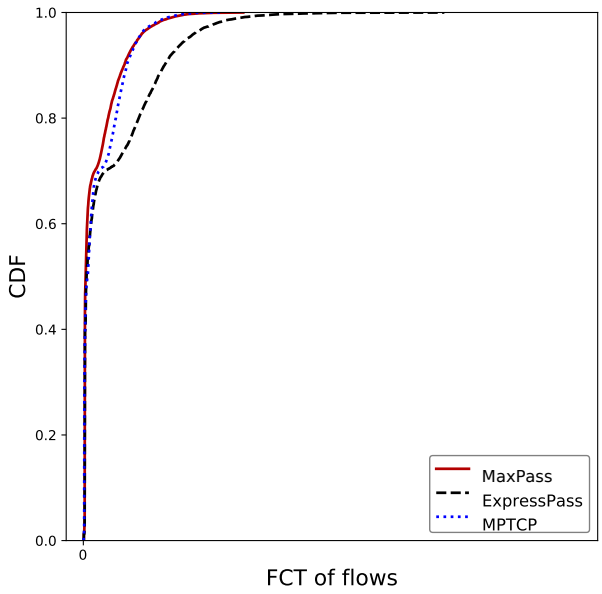


Fig. 9. Flow completion time of each flow in the large-scale data center network topology is shown for the Cache Follower workload using CDF. Notice that the plots of MaxPass are located left compared to those of MPTCP and ExpressPass, which shows the shorter flow completion times.

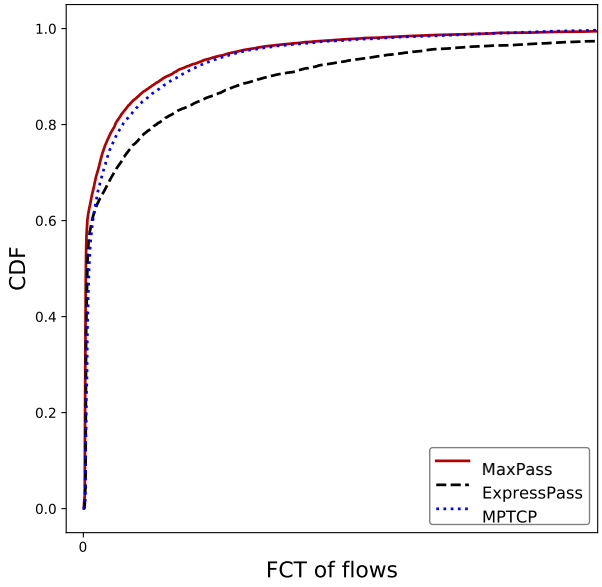


Fig. 10. Flow completion time of each flow in the large-scale data center network topology is shown for the Web Search workload using CDF. The plots of MaxPass are located left compared to those of ExpressPass and similar to those of MPTCP.

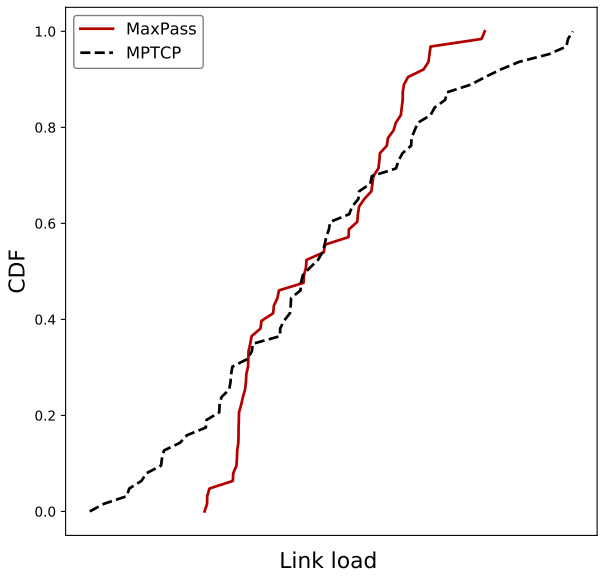


Fig. 11. Load of each link for the Cache Follower workload is plotted using CDF to see how well MaxPass achieves load balancing. The higher slope indicates the more balanced load.

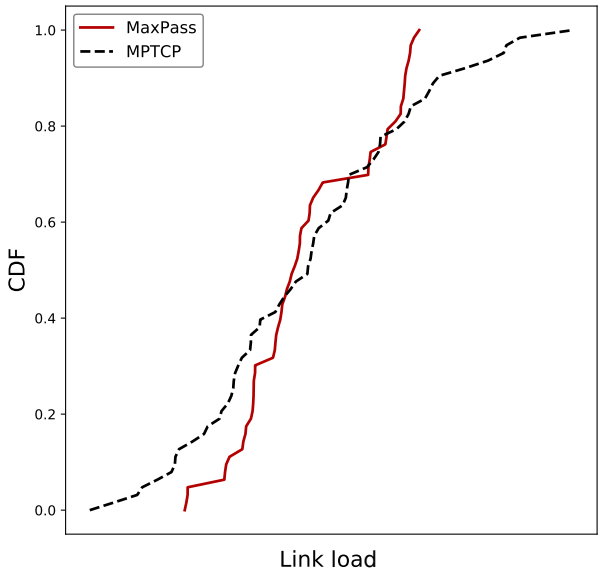


Fig. 12. Load of each link for the Web Search workload is also plotted using CDF to see how well MaxPass distributes the entire load.

scale data center network topology. The first goal of MaxPass is balancing loads using multiple paths for a single connection. Therefore, the load balancing of MaxPass is compared with that of ExpressPass that is the single path scheme for comparison purposes. Figs. 11 and 12 represent the CDFs of the link loads for the two workloads. If the traffic is evenly distributed to all links, the loads of all the links would be almost the same. As the same seed is used for the two experiments of ExpressPass and MaxPass, their generated flow sizes and arrival times are the same. Therefore, the higher slope means that the load is more evenly distributed. Figs. 11 and 12 show that MaxPass achieves the higher slope than ExpressPass. As MaxPass adaptively selects less congested paths through path probing, the underutilized paths will be selected and the over-utilized paths will be avoided.

V. RELATED WORK

As mentioned before, data centers have special network topology, e.g., fat-tree. Also, they have distinct traffic characteristics such as many-to-one communication patterns, ON-OFF patterns, or skewed distributions of flow sizes. Due to such particular topology and traffic characteristics, traditional load balancing algorithms are not well suited for the data center networking en-

vironments. Hence, many studies on load balancing techniques for data centers have been carried out. Load balancing schemes for the data centers could be divided into centralized techniques and decentralized techniques with respect to flow assignment mechanisms.

A. Centralized

Centralized load balancing mechanisms typically run on a central controller, collect network congestion information, and assign flows to under-utilized paths. The central controller usually holds a global view of data center networks, which helps to select the least congested path. However, centralized load balancing mechanisms require the communication overhead of network switches and the controller. Besides, there are time constraints for the centralized entity to take effective actions on the network switches since short flows sends traffic only for a short interval (say, on the order of microseconds). In Mahout *et al.* [20], a centralized balancing mechanism is deployed in software-defined networks. Mahout *et al.* suggest that the elephant flow detection process should be performed by end-hosts to reduce high overhead caused by centralized detection of elephant flows. However, although Mahout reduces the overhead of detecting large flows, it still cannot work at the level of milliseconds. In FDALB [21], it classifies flows to short flows and long flows and schedules them differently. Long flows are marked with tags at end-hosts. Short flows are transmitted by switches using static schemes like ECMP, while a central controller schedules long flow depending on the global network status (e.g., congestion). In Freeway [22], it considers different requirements depending on flow types, i.e., low latency for short flows and high throughput for long flows, and proposes a centralized algorithm to meet those requirements. Freeway dynamically divides paths into low latency paths for short flows and high throughput paths for long flows. The size of flows is estimated at end-hosts. Then, long flows exchange their traffic information with the controller before being sent, and the central controller schedules long flows. On the other hand, short flows are transmitted directly using ECMP without scheduling of the central controller. Freeway achieves the higher throughput and lower latency than ECMP. However, it has a limitation when all low latency paths have link utilization over 50%. Even though there are residual link capacities to transfer long flows, Freeway leaves these capacities unused. In Fastpass [23], it controls the sending time and path of each packet using a central controller. It adopts a hybrid mechanism that combines transport control and traffic scheduling in order to achieve zero-queuing in data center networks. Fastpass can reduce queue lengths and flow completion times, and achieve high throughput compared with TCP. However, it is difficult to be deployed and scaled up since a single controller of Fastpass has the scalability issue of handling hundreds to thousands of endpoints.

B. Decentralized/Distributed

Distributed load balancing schemes without any centralized entity select paths for flows locally at hosts or switches. Therefore, there is no or little scalability issue and hence less constrained in extending the topology compared to centralized schemes. Distributed schemes can relatively quickly respond to traffic dynamics. However, it is quite hard to collect the accurate network information.

CLOVE [24] is implemented in soft edge switches to avoid hardware and end-host stack modifications while quickly reacting to latency-sensitive short flows. It uses standard ECMP in the physical network and changes the header of packets at software switches to directly guide how switches transfer packets. CLOVE reduces average flow completion time. However, since CLOVE uses overlay and virtualization technologies, it has an issue when deploying in non-virtualized data centers.

In CONGA [25], it is optimally designed for 2-tier Leaf-Spine topology. It aims to quickly react to congestion while obtaining the global congestion information from leaf switches without adding an entity. By using four additional fields in VxLAN, CONGA carries the congestion information through data packets from destination leaf switches to source leaf switches. To record the congestion information of paths, each leaf switch maintains a per-destination congestion table. CONGA can achieve lower flow completion times than ECMP and MPTCP. Besides, CONGA achieves from 2 to 8 times better throughput than MPTCP in incast scenarios.

Random packet spraying (RPS) [26] is an intuitive and simple multipath scheme, in which packets of every flow are randomly assigned to one of the available shortest paths to the destination. RPS outperforms ECMP and achieves similar performance to MPTCP in symmetric scenarios, while requiring no modifications to hardware and protocols. The disadvantages of RPS lie in packet reordering and the influence of asymmetry (e.g., due to link failures) in the topology.

MP-ECN [27] aims to enable explicit congestion notification (ECN) for multi-service multi-queue data center network. Each queue adjusts ECN marking threshold to preserve weighted fair sharing. However, this scheme is not implementable in current switch design.

Auto [28] leverages machine learning (ML). It automatically adjusts the network parameters, especially used in PIAS [29]. It collects flow information from end hosts, and take actions to achieve operator-defined goals. Due to the decision overhead, it divides system by peripheral and central nervous systems. While peripheral systems (PS) located in end hosts collects network information and makes a decision for short flows, central system (CS) performs actions for long flows.

Hermes [30] is designed for load balancing resilient to uncertainties such as traffic dynamics, topology asymmetry, and failures. When the uncertainty is detected using RTT, ECN and packet drops, a packet is re-routed.

DCQCN [31] introduces ECN in RDMA over converged ethernet (RoCE) network which uses priority-based flow control (PFC). PFC transmit a pause packet to connected switches when their queues are occupied over the threshold. The switches stop sending packets when they receive the pause packet. Frequent pause packets can degrade overall network performance or induce a deadlock. DCQCN prevents frequent pause packets by enabling ECN.

TIMELY [32] uses RTT to detect a bottleneck switch. RDMA NIC facilitates to measure the RTT precisely with microsecond accuracy. When RTT increases (i.e., RTT gradient is positive), a sender reduces a sending rate as the positive RTT gradient is

regarded as congestion.

A multipath transport for RDMA is presented in MP-RDMA [8]. It proposes three mechanisms 1) to aware congestion without managing per-path state, 2) to mitigate out-of-order packets, and 3) to ensure in-order operations.

VI. CONCLUSION

In this paper we present MaxPass, a new adaptive load balancing mechanism for data center networks. It leverages a credit packet to probe each path to mitigate network congestion. The sender can choose best paths when a flow starts. Also, the sender can exploit other paths when network congestion happens, which is indicated by the number of dropped credit packets exceeds a certain threshold. MaxPass employs a new credit feedback control algorithm to mitigate credit wastes by increasing the credit sending rate in inverse proportion to the number of subflows and decreasing the rate dramatically compared to ExpressPass. Our evaluation on the ns-2 simulation shows the performance improvement in terms of flow completion time and link utilization.

REFERENCES

[1] N. Bronson *et al.*, "TAO: Facebook's distributed data store for the social graph," in *Proc. USENIX ATC*, June 2013, pp. 49–60.

[2] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 51–62.

[3] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, Third quarter 2018.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proc. ACM WREN*, 2009, pp. 65–72.

[5] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Comput. Commun. Review*, vol. 41, pp. 266–277, 2011.

[6] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 465–478, 2015.

[7] M. Kheirkhah and M. Lee, "AMP: A better multipath tcp for data center networks," *arXiv preprint arXiv:1707.00322*, 2017.

[8] Y. Lu *et al.*, "Multi-path transport for rdma in datacenters," in *Proc. USENIX NSDI*, Mar. 2018.

[9] C. Clos, "A study of non-blocking switching networks," *Bell Syst. Technical J.*, vol. 32, no. 2, pp. 406–424, 1953.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Review*, vol. 38, pp. 63–74, 2008.

[11] C. Hopps, "Analysis of an equal-cost multi-path algorithm," tech. rep., 2000.

[12] A. Ford *et al.*, "TCP extensions for multipath operation with multiple addresses, draft-ietf-mptcp-multiaddressed-09," *Internetdraft, IETF*, Mar. 2012.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, Mar. 2010.

[14] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*, 2017, pp. 239–252.

[15] H. Kung, T. Blackwell, and A. Chapman, "Credit-based flow control for atm networks: Credit update protocol, adaptive credit allocation and statistical multiplexing," *ACM SIGCOMM Comput. Commun. Review*, vol. 24, pp. 101–114, 1994.

[16] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multipath congestion control for shared bottleneck," in *Proc. PFLDNeT workshop*, vol. 357, p. 378, 2009.

[17] S. McCanne, "Network simulator ns-2," [Online] Available: *http://www.isi. edu/nsnam/ns/*, 1997.

[18] M. Alizadeh *et al.*, "Data center tcp (dctcp)," *ACM SIGCOMM Comput. Commun. Review*, vol. 41, no. 4, pp. 63–74, 2011.

[19] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *ACM SIGCOMM Comput. Commun. Review*, vol. 45, pp. 123–137, 2015.

[20] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, 2011, pp. 1629–1637.

[21] S. Wang *et al.*, "Fdalb: Flow distribution aware load balancing for datacenter networks," in *Proc. IEEE/ACM IWQoS*, 2016, pp. 1–2.

[22] W. Wang *et al.*, "Freeway: Adaptively isolating the elephant and mice flows on different transmission paths," in *Proc. IEEE ICNP*, 2014, pp. 362–367.

[23] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Comput. Commun. Review*, vol. 44, no. 4, pp. 307–318, 2015.

[24] N. Katta *et al.*, "Clove: How i learned to stop worrying about the core and love the edge," in *Proc. ACM HotNet*, 2016, pp. 155–161.

[25] M. Alizadeh *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Review*, vol. 44, pp. 503–514, 2014.

[26] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, 2013, pp. 2130–2138.

[27] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ecn in multi-service multi-queue data centers," in *Proc. USENIX NSDI*, 2016, pp. 537–549.

[28] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM SIGCOM*, 2018, pp. 191–205.

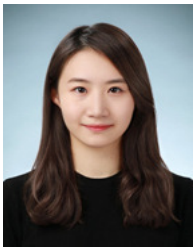
[29] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proc. USENIX NSDI*, 2015, pp. 455–468.

[30] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. ACM SIGCOM*, 2017, pp. 253–266.

[31] Y. Zhu *et al.*, "Congestion control for large-scale rdma deployments," in *Proc. ACM SIGCOM*, 2015, pp. 523–536.

[32] R. Mittal *et al.*, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Review*, vol. 45, pp. 537–550, 2015.

Minkyung Park received her B.S. degree in Computer Science from Korea Aerospace University, Korea, in 2014. She is currently working toward her Ph.D. degree at the School of Computer Science and Engineering, Seoul National University. Her research interests include network security, privacy, anonymity, and wireless network.



Sungmin Sohn received her B.S. degree in Computer Science and engineering from Ewha Womans University. She is attending a master's course in School of Computer Science and Engineering at Seoul National University since 2017. Her research interests include network security and web security.

Kwangwook Kwon received B.S degree on Electronic and Electrical Engineering from Sungkyunkwan University, Korea in 2011. At present, he is working in Samsung Electronics in Korea from 2011. He is also master course student in Seoul National University from 2018. His research interests include software defined network, datacenter network, 5G core network.

Ted Taekyoung Kwon received the BS, MS, and Ph.D. degrees from Seoul National University (SNU) in 1993, 1995, and 2000, respectively. He is a professor with the Department of Computer Science and Engineering, Seoul National University. Before joining SNU, he was a Postdoctoral Research Associate at the University of California Los Angeles and City University New York. During his graduate program, he was a visiting student at the IBM T.J. Watson Research Center and at the University of North Texas. He was a Visiting Professor at Rutgers University in 2010. His research interest lies in future Internet, network security, and wireless networks.