Adjusting Switching Granularity of Load Balancing for Heterogeneous Datacenter Traffic

Jinbin Hu, Jiawei Huang, *Member, IEEE*, Wenjun Lyu, Weihe Li, Zhaoyi Li, Wenchao Jiang, Jianxin Wang, *Senior Member, IEEE*, and Tian He, *Fellow, IEEE, ACM*

***Abstract*— The state-of-the-art datacenter load balancing designs commonly optimize bisection bandwidth with homogeneous switching granularity. Their performances surprisingly degrade under mixed traffic containing both short and long flows. Specifically, the short flows suffer from long-tailed delay, while the throughputs of long flows also degrade dramatically due to low link utilization and packet reordering. To solve these problems, we design a traffic-aware load balancing (TLB) scheme to adaptively adjust the switching granularity of long flows according to the load strength of short ones. Under the heavy load of short flows, the long flows use large switching granularity to help short ones obtain more opportunities in choosing short queues to complete quickly. On the contrary, the long flows reroute flexibly with small switching granularity to achieve high throughput. Furthermore, under extremely bursty scenario, we utilize the packet slicing scheme for long flows to release bandwidth for short ones. The experimental results of NS2 simulation and testbed implementation show that TLB significantly reduces the average flow completion time of short flows by 16%-67% over the state-of-the-art load balancers and achieves the high throughput for long flows. Moreover, for extreme bursty case, at the acceptable throughput degradation of long flows, TLB with packet slicing reduces the deadline missing ratio of bursty short flows by up to 80%.**

***Index Terms*— Data center, load balancing, multipath.**

I. Introduction

MODERN data centers deploy the multi-rooted tree networks such as Fat-tree and Clos to provide high bisection bandwidth via multiple paths between any given pair of hosts [1]–[4]. To meet the increasing traffic demands of latency-sensitive and throughput-oriented applications, how to efficiently balance traffic across available multiple paths becomes a crucially important issue in large-scale data center networks (DCNs).

As the $defacto$ load balancing scheme, ECMP (Equal Cost Multipath) [5] is widely deployed in data centers to randomly map each flow to one of the paths by using flow hashing. ECMP is very simple but suffers from well-known performance problems such as hash collisions and the inability to reroute flow adaptively. Recently, a lot of better load balancing designs have emerged in DCNs. Random Packet Spraying (RPS) [6], DRILL [3] and Hermes [2] split flows at the granularity of packet and choose the next hop for each packet to leverage multiple paths. Presto [8] picks paths for fixed-sized chunks of data (i.e., 64KB) to boost throughput and reduce packet reordering. CONGA [1] and LetFlow [9] adopt flowlet switching to provide a finer granularity without causing much packet reordering.

However, prior load balancing designs reroute all flows at a certain granularity, either flow, flowlet, flowcell or packet, regardless of the flow sizes. They are agnostic to the heterogeneous traffic feature that the mixed short and long flows are generated by large-scale applications such as web search, social networking and retailing system in data centers [10], [11]. In general, the datacenter heterogeneous traffic can be characterized by heavy-tailed distribution from a macro perspective [12]–[14]. That is, around 80% of traffic is provided by only about 20% throughput-sensitive long flows, and about 80% of delay-sensitive short flows provide only about 20% of traffic [12]–[14].

As a result, when the short and long flows are rerouted at the same granularity, the short flows easily experience long-tailed queueing delay since they are difficult to seize the less-congested paths under the overwhelming data of long flows, resulting in large flow completion time (FCT). Moreover, under the non-adaptive granularity, the long flows suffer from the throughput loss due to the low link utilization or out-of-order problem when the network traffic changes dynamically [15], [16].

In this paper, we present a new load balancing scheme TLB, which differentiates the switching granularity for different types of flows to achieve low latency for short flows and high throughput for long flows. When the traffic load of short flows is high, the long flows are rerouted at the large granularity to leave more less-congested paths for short flows. In contrast, under the low load of short flows, the long flows are switched at the small granularity to improve the link utilization. The short flows pick paths on packet-level to flexibly seize the fast paths. TLB successfully splits heterogeneous traffic across

Manuscript received January 9, 2020; revised July 16, 2020, November 30, 2020, April 8, 2021, and May 30, 2021; accepted June 7, 2021; approved by IEEE/ACM Transactions on Networking Editor P. Giaccone. This work was supported in part by the National Natural Science Foundation of China under Grant 61872387, in part by the Ministry of Education (MOE) Start-up Research Grant (SRG) under Grant ISTD2020159, and in part by the Project of Foreign Cultural and Educational Expert under Grant G20190018003. This article was presented in part at the ACM ICPP 2019: 48th International Conference on Parallel Processing. *(Corresponding author: Jiawei Huang.)*

Jinbin Hu, Jiawei Huang, Weihe Li, Zhaoyi Li, and Jianxin Wang are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China (e-mail: jiaweihuang@csu.edu.cn).

Wenjun Lyu is with the Department of Computer Science, Rutgers University, Piscataway, NJ 08854 USA.

Wenchao Jiang is with the Pillar of Information Systems Technology and Design, Singapore University of Technology and Design, Singapore 487372.

Tian He is with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Digital Object Identifier 10.1109/TNET.2021.3088276

multiple paths to avoid the long-tailed queueing delay for short flows, improve the link utilization for long flows and reduce packet reordering.

In summary, our major contributions are:

- We conduct an extensive simulation-based study to explore the issues of mixed heterogeneous flows rerouted at the same and non-adaptive granularity in typical load balancing designs: the short ones experience long-tailed queueing delay and out-of-order problem, and the long ones suffer from the low link utilization and packet reordering.
- We propose a load balancing scheme TLB, which flexibly reroutes short and long flows at different granularities by perceiving the traffic load. Specifically, TLB adaptively adjusts the switching granularity for long flows according to the load strength of short flows and picks paths for short flows at packet-level to achieve low queueing delay for short flows and high link utilization for long ones.
- To protect short flows under extremely bursty scenario, we utilize the packet slicing scheme to adjust the packet size of long flows. We theoretically analyze the effectiveness of packet slicing and show that the packet slicing scheme greatly reduces the deadline missing ratio of short flows.
- By using both NS2 simulations and testbed implementation, we demonstrate that TLB performs remarkably better than the state-of-the-art load balancing schemes. Specifically, TLB greatly reduces the AFCT by ∼16%-67% for short flows under heavy workload. Meanwhile, TLB yields up to ∼35% and ∼49% throughput improvement for long flows over Presto and LetFlow, respectively.

The rest of the paper is organized as follows. In Section II and III, we describe our design motivation and overview, respectively. In Section IV, we give the model analysis of TLB. We discuss the implementation in Section V. In Section VI and VII, we show the test results of NS2 simulation and testbed experiment, respectively. We present the related works in Section VIII and conclude the paper in Section IX. The appendixes provide supplementary experiments under realistic workloads and P4 software switch.

II. Motivation

A. Load Balancing With the Same Granularity

The existing load balancing designs typically use one of the three switching granularities, flow-level, flowlet-level, and packet-level. In the flow-level schemes, each flow is transferred on one path without flexible switching. The flowlet-level schemes reroute flows only when the flowlets emerge. The packet-level schemes choose path for each packet to make full use of multiple paths, but easily cause serious packet reordering under the topology asymmetry.

Case Study: We use a simple example to illustrate the issues of typical load balancing schemes. Fig. 1 shows that 3 senders (S_1, S_2, S_3) and 3 receivers (R_1, R_2, R_3) connect to the corresponding leaf switches. There are 3 queues (Q_1, Q_2, Q_3) on the corresponding output ports of the leaf switch L_1. S_1 continuously sends a long flow to R_1 at time T_1, while S_2 and S_3 respectively send a short flow to R_2 and R_3 at time T_2 and T_3. We use four different granularities to reroute flows at the leaf switch L_1.

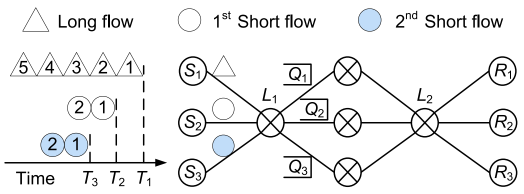


Fig. 1. Leaf-spine topology.

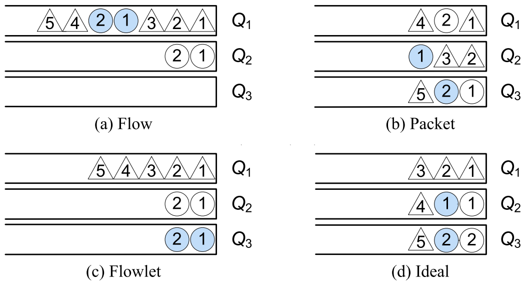


Fig. 2. Queueing under different granularities.

Fig. 2 shows that the packets of short and long flows are queued based on the flow, packet, flowlet and ideal switching granularity, respectively. Fig. 2 (a) shows that, under the flow-level switching, one short flow is queued behind the long flow in Q_1 though Q_3 is empty, resulting in the large queueing delay and low link utilization. The packet-based switching is shown in Fig. 2 (b). Although all packets of short and long flows are evenly spread among multiple paths, there exist reordering packets and large queueing delay experienced by short flows. Fig. 2 (c) shows the case of flowlet-based switching. Three flows are always transmitted in their respective queues due to insufficient inactivity gap. The inflexible switching unavoidably leads to the low link utilization. The ideal case is shown in Fig. 2 (d). At the beginning, the switching granularity of the long flow and the short ones are 3 and 1, respectively. The long flow is rerouted at its 4^{th} packet to leave the less-congested paths for the short flows, which flexibly pick the short queues of Q_2 and Q_3 per packet to avoid the large queueing delay. When the short flows are finished, the long flow decreases the switching granularity to packet to improve its throughput.

B. Impact of Switching Granularity

We conduct NS2 simulation test to analyze the impact of switching granularity on the short and long flows. We use a leaf-spine topology with 15 equal-cost paths between host pairs. The bottleneck bandwidth is 1Gbps and the round-trip propagation delay is 100μs. The switch buffer size is 256 packets. Each sender sends a DCTCP [12] flow to a receiver via the leaf and spine switches. We set the ECN marking threshold to 128 packets in our simulations. In our test, the mixture of 100 short flows with random size of less than 100KB and 5 long flows larger than 10MB are generated in heavy-tailed distribution. The flowlet timeout is 150μs [2].

We test the impact of switching granularity on short flows. We compare the cumulative density function (CDF) of queueing length experienced by each packet of short flows under three granularities. As shown in Fig. 3 (a), with the increasing of switching granularity, the queue length becomes larger. Moreover, due to the growing queue length caused by

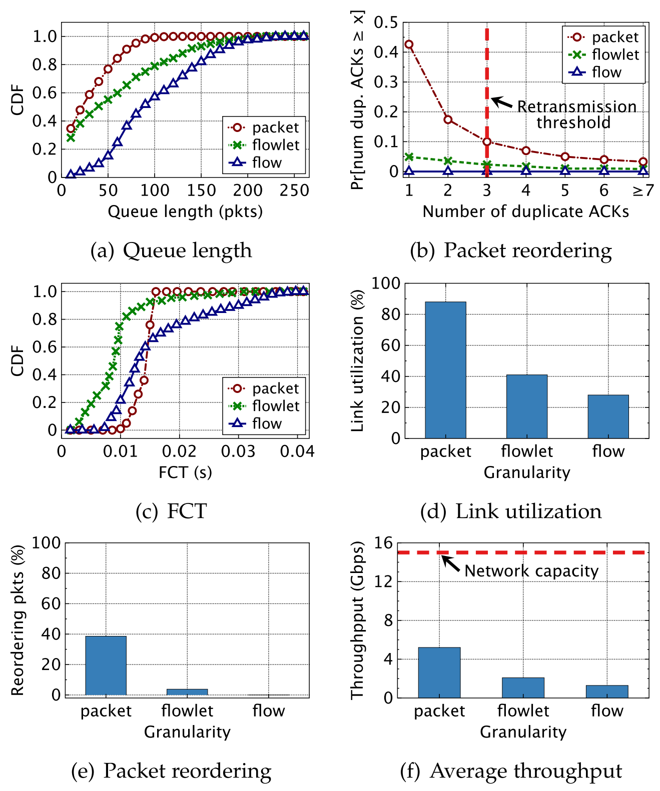


Fig. 3. The impact on short and long flows.

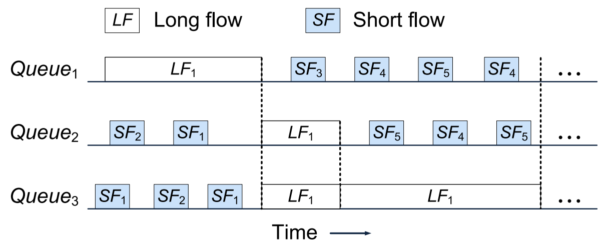


Fig. 4. Queueing process.

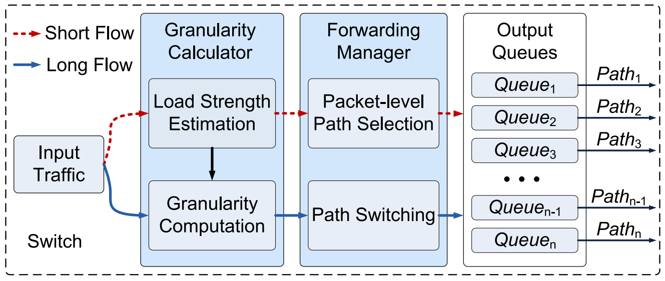


Fig. 5. TLB architecture.

long flows, the difference of queue lengths under flow-level switching is large. For flow-level granularity, all short and long flows are hashed to 15 paths, meaning that the long flows only occupy up to 30% of the total paths (i.e., 5 paths). Therefore, the majority of short flows do not collide with the long ones. The bursty short flows that share paths with long ones are likely to experience the queue length of ECN marking threshold, while the queue lengths of the other short flows are less than the ECN marking threshold.

Fig. 3 (b) shows the ratio of TCP duplicate ACKs to demonstrate the packet reordering. Compared with the flow-level and flowlet-level switching, the number of duplicate ACKs is quite large under the packet-level switching, causing the spurious packet loss. When the number of duplicate ACKs reaches the retransmission threshold, the TCP sender cuts the congestion window. Fig. 3 (c) shows the CDF of flow completion time. We observe that the tailed delay increases with the increasing of switching granularity due to the larger difference of queue lengths. Moreover, though obtaining the smallest queue length, the packet-level switching scheme still does not achieve the best performance of FCT because of its packet reordering problem.

Next, we examine the impact on long flows. Fig. 3 (d) shows that, since the long flows provide the overwhelming amount of data in data centers, the load balancing with large granularity (i.e., flow-level) causes low link utilization. As shown in Fig. 3 (e), though splitting the traffic of long flows in a more balanced way, the flow switching with smaller granularity introduces more out-of-order packets. Fig. 3 (f) shows that, due to the dilemma between the link utilization and packet reordering, long flows only obtain the average throughput of less than 35% of the network capacity.

C. Summary

Our analysis of the impact of load balancing with same switching granularity on the network performance leads us to conclude that (i) the short flows experience the large queueing delay as the granularity increases and have more reordering packets as the granularity decreases, (ii) the long flows suffer from the throughput degradation due to the non-adaptive granularity in rerouting flows. These conclusions motivate us to design and implement a traffic-aware load balancing scheme with adaptive granularity to achieve good performance for both short and long flows.

III. Design Overview

In this section, we present an overview of TLB. The key point of TLB is to adaptively adjust the switching granularity of long flows and flexibly pick path for each packet of short flows to alleviate the large queueing delay and packet reordering. Specifically, on the one hand, the long flows are switched dynamically with adaptive granularity based on the load strength of short flows, leaving more less-congested paths for short flows. On the other hand, each packet of the short flows is routed on the shortest queue to avoid being blocked by long flows.

Fig. 4 shows the queueing process with adaptive switching granularity for short and long flows. The long flow is dynamically rerouted across multiple paths to achieve high throughput and simultaneously reduce the queueing delay for short flows. At the beginning, the long flow occupies the $Queue_1$ with large switching granularity, leaving non-congested $Queue_2$ and $Queue_3$ for the short flows with the packet-level switching granularity. As soon as the arrival rate of short flows decreases, the long flow is rerouted to $Queue_2$ and $Queue_3$ with smaller granularity to flexibly make use of multiple paths. In contrast, under the high load of short flows, the switching granularity of the long flow adaptively increases after being rerouted to $Queue_3$. Therefore, the short flows are able to efficiently and quickly transmit each packet on the paths unused by the long flow. Fig. 5 shows the architecture of TLB.

1) **Granularity Calculator:** At the switch, the granularity calculator consists of two parts: load strength estimation and granularity computation. Firstly, the load strength of short flows is estimated according to their arrival rates, and then the switching granularity of long flows is calculated to guarantee the low queueing delay for short flows.

Specifically, when the traffic load of short flows is heavy, the switching granularity of long flows should be larger to ensure that the short flows have more opportunities to use enough non-congested queues to avoid large queueing delay. Conversely, with the decrease of the load strength of short flows, the switching granularity of long flows should be smaller to switch flexibly among the multiple paths to improve link utilization. At each time interval t (500μs by default [1]), the granularity computation module updates the switching granularity periodically. The most important work is adjusting the switching granularity of long flows to achieve the good tradeoff between small flow completion time of short flows and high throughput of long flows.

2) **Forwarding Manager:** The forwarding manager is responsible for switching the short and long flows with different granularities and selecting the forwarding path according to the real-time queue lengths of the output ports.

For a long flow, when a new packet arrives at the switch, TLB reroutes the packet to the shortest queue only if the current queue length of the long flow reaches the switching threshold calculated by the granularity computation module. Otherwise, TLB forwards the new arriving packet to the same queue as the last arrival packet in the same flow. For short flows, TLB reroutes each arriving packet to the output port with shortest queue length to reduce queueing delay caused by the long flows. In high-speed data center networks, to reduce the packet processing overhead, TLB chooses the shortest queue as the forwarding port for short-flow packets in a time-driven manner. That is, TLB periodically updates the minimum queue length every base RTT and forwards the arrival packets of short flows to the port with the shortest queue length.

IV. Adaptive Switching Granularity

A. Model Analysis

The goal of this paper is to develop a simple load balancing scheme that meets the requirements of both short and long flows. The primary goal of short flows is reducing their average flow completion time to meet their deadlines [17], [18], while the long flows require large throughputs. In our design, the switching granularity is elaborately adjusted to firstly meet the delay requirements (i.e., deadline) of short flows and then leave as much resources as possible for long flows to improve their throughputs. To achieve the design goal, we use the queueing model to calculate the optimal switching granularity as follows.

However, the datacenter traffic is very bursty and unpredictable at short timescales (e.g., 10∼100s of microseconds) [1], leading to the varying distribution of flow size from a micro viewpoint. For example, during the distributed training process of machine learning, a large number of model parameters are updated synchronously between thousands of servers at the end of each iteration [19]. In web search application, the massive responses are generated concurrently among servers to aggregators, incurring bursty transient traffic [12]. Specifically, the bursty traffic of short flows in these applications shows ON/OFF mode [14]. The bursty short flows start and finish their transmissions during the ON periods, which are much shorter than the OFF periods. The number of bursty short flows varies in each ON period. During the ON periods of short flows, the long flows are always existing since they have much larger flow size.

In general, the number of short and long flows varies with traffic pattern. TLB periodically updates the number of short and long flows at switches to capture traffic patterns that chang over time. If none packet is received during the sampling interval, which is set to 500μs as same as the updated interval of switching granularity of long flows [1], the corresponding flow record is removed from the flow table. Based on the number of short flows under different traffic patterns, TLB calculates the arrival rate and load strength of short flows. Then TLB dynamically adjusts the switching granularity of long flows according to the load strength of short flows. The theoretical relation between switching granularity and traffic pattern is as follows.

Let n denote the number of all equal-cost paths, which includes n_S and n_L paths allocated by TLB for m_S short flows and m_L long flows, respectively. We use C and RTT to denote the bottleneck link capacity and the round-trip propagation delay, respectively. The long flows continuously send packets with the maximum window size W_L limited by the receiver buffer (64KB by default in Linux) [20] after quickly entering the congestion avoidance phase.

We assume that m_S short flows with average size of X bytes are transmitted over n_S paths with an average completion time of FCT_S. The arrival rate λ of the short flows following a Poisson distribution is calculated as

$$\lambda = \frac{m_S \cdot X}{FCT_S \cdot n_S}. \tag{1}$$

TLB estimates the load strength of bursty short flows according to their arrival rates. The load strength ρ of bursty short flows is calculated as

$$\rho = \frac{\lambda}{C} = \frac{m_S \cdot X}{FCT_S \cdot n_S \cdot C}. \tag{2}$$

where C is the link capacity in packets per second.

We use q_{th} to denote the switching granularity of long flows. When the queue length of long flow is larger than q_{th}, the long flow is rerouted to another egress port. The value of q_{th} is updated periodically for each time interval t. Then, since the sum of the number of queued and in-flight packets on the n_L paths occupied by long flows is equal to the amount of data sent by long flows within the time interval t, we have

$$q_{th} \cdot n_L + t \cdot C \cdot n_L = m_L \cdot W_L \cdot \frac{t}{RTT}. \tag{3}$$

Given the total number of paths $n = n_L + n_S$, from Equation (3), we obtain the number of paths n_S allocated for short flows as

$$n_S = n - \frac{m_L \cdot W_L \cdot \frac{t}{RTT}}{q_{th} + t \cdot C}. \tag{4}$$

For short flows, the mean flow completion time FCT_S includes the queueing and transmission delay, that is

$$FCT_S = E[W] \cdot r + \frac{X}{C}, \tag{5}$$

where X is the average flow size of short flows, $\frac{X}{C}$ is the transmission delay, r is the number of RTT rounds to finish transfer of a short flow and $E[W]\cdot r$ is the expected queueing delay for r rounds of short flows.

In data centers, about 80% of TCP flows are less than 100KB [12]–[14], [20]–[22]. These short flows mostly finish in the slow-start phase [20], [23], in which each flow firstly sends out 2 packets, then 4, 8, 16, etc. Thus, the number of RTT rounds r to finish transfer of a short flow with size x bytes is calculated as

$$r = \lfloor \log_2 \frac{x}{MSS} \rfloor + 1, \tag{6}$$

where MSS is the size of a TCP segment.

We use the M/G/1-FCFS queueing model [20], [22], [24], to analyze the average queueing delay $E[W]$ of each round. Since the short flows are transferred per packet across the multiple paths and each packet of the short flows chooses the shortest path, the expected queueing delay $E[W]$ is equal to the average waiting time in each queue for a single packet, which is obtained from the Pollaczek-Khintchine formula [25] as

$$E[W] = \frac{1 + C_v^2}{2} \cdot \frac{\rho}{1-\rho} \cdot E[S], \tag{7}$$

where $E[S]$ is the service time of a single packet with its value as $\frac{1}{C}$. C_v^2 is the squared coefficient of variation of the service time distribution as

$$C_v^2 = \frac{Var[S]}{E^2[S]} = \frac{E[S^2] - E^2[S]}{E^2[S]}. \tag{8}$$

Then, the expected queueing delay $E[W]$ is

$$E[W] = \frac{\rho}{2(1-\rho)} \cdot \frac{E[S^2]}{E[S]} = \frac{\rho}{2(1-\rho)} \cdot \frac{1}{C}. \tag{9}$$

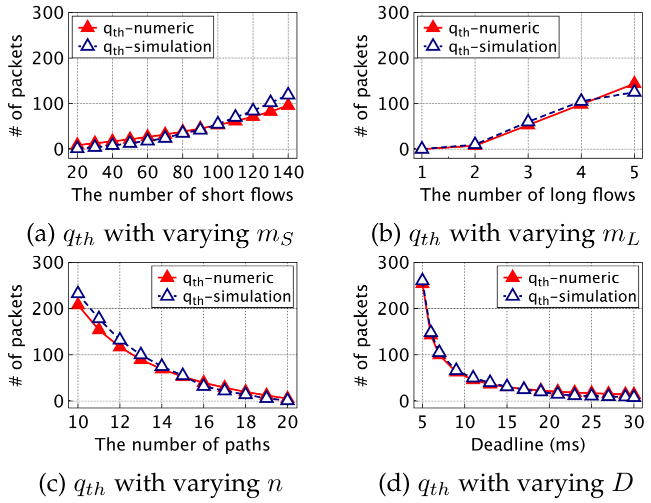
Thus, according to the above Equation (2), (4), (5) and (9), we have the equation about the mean FCT of short flows FCT_S as

$$FCT_S = \frac{m_S \cdot X \cdot \frac{r}{C}}{2[FCT_S \cdot (n - \frac{m_L \cdot W_L \cdot \frac{t}{RTT}}{q_{th} + t \cdot C}) \cdot C - m_S \cdot X]} + \frac{X}{C}. \tag{10}$$

Given that the deadline of each short flow is randomly distributed between $[D_{min},\ D_{max}]$, we set the associated default deadline D to the value of 25th percentile in CDF of the statistical flow deadlines to calculate the switching granularity for long flows. The reason why the deadline is set to the 25th percentile and the corresponding experiments are introduced in Section 6.2. To guarantee that the short flows complete within their associated deadline D, the mean flow completion time for short flows FCT_S should satisfy $FCT_S \leq D$. Finally, the switching threshold q_{th} of queue length for rerouting the long flows is adjusted according to the load strength of short flows to meet their deadlines. Specifically, q_{th} should satisfy the following expression

$$q_{th} \geq \frac{m_L \cdot W_L \cdot \frac{t}{RTT}}{n - \frac{r \cdot \frac{X}{C} + 2(D - \frac{X}{C}) \cdot X}{2(D - \frac{X}{C}) \cdot D \cdot C} \cdot m_S} - t \cdot C. \tag{11}$$

To leave as much resources as possible for long flows to improve their throughputs, the long flows are rerouted at the minimum value of q_{th} satisfying Equation (11) in our design. Specifically, when a new packet of a long flow arrives at the switch, TLB makes forwarding decision based on the real-time queue length of the current output port of the flow. Once the queue length reaches the minimum value of q_{th} (i.e., the optimal switching granularity), TLB selects the shortest queue to reroute the arriving packet. Otherwise, TLB forwards the arriving packet to the same path as the lastly arrival packet in the same flow without switching. Since TLB is able to dynamically adjust the switching granularity of long flows according to the load strength of short flows, the load balancing mechanism of TLB is resilient to the traffic patterns with temporal variations.



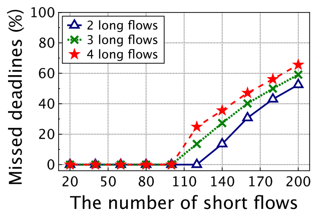
(a) q_{th} with varying m_S (b) q_{th} with varying m_L (c) q_{th} with varying n (d) q_{th} with varying D

Fig. 6. Numeric and simulation comparison.

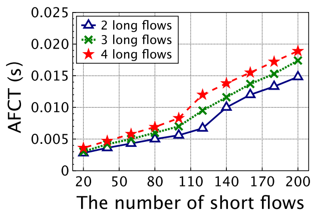
B. Model Verification

We evaluate the correctness of the theoretical analysis by NS2 simulations. We use the leaf-spine topology with 15 equal-cost paths between any pair of end-hosts. The bottleneck link bandwidth and switch buffer size are 1Gbps and 512 packets, respectively. We use DCTCP as the underlying transport protocol. By default, 3 long flows (>10MB) and 100 short flows (<100KB) are generated with heavy-tailed distribution. The average size of short flows is 70KB in this simulation. The deadline of each short flow is randomly distributed between [5ms, 25ms] as illustrated in [17]. We set the default deadline D as the value of 25th percentile deadline (i.e., 10ms), which is also used in the following simulations. In this test, we measure the minimum value of switching threshold q_{th} for rerouting the long flows under the condition that no short flows miss their deadlines in the time interval t of 500μs, which is the general inactivity gap between two bursts of packets in short flows [1].

Fig. 6 shows the comparison of numerical and simulation results. The switching threshold q_{th} of queue length for rerouting the long flows increases as the number of short flows increases as shown in Fig. 6 (a). With the increasing number of long flows, q_{th} also increases as shown in Fig. 6 (b), because the long flows need larger switching granularity under the increasing workload to meet the delay requirements of short flows. As shown in Fig. 6 (c) and (d), q_{th} decreases with the increasing of total number of path and deadline, respectively.

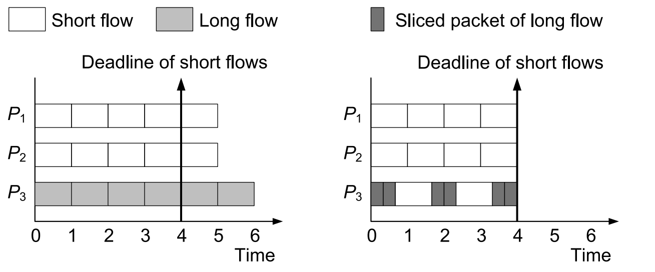


(a) Missed deadline of short flows



(b) AFCT of short flows

Fig. 7. Impact of extreme burstiness of short flows on TLB.



(a) Short flows miss deadline without packet slicing (b) None of short flows miss deadline with packet slicing

Fig. 8. TLB with and without packet slicing scheme for long flows under the bursty scenario.

In brief, the switching granularities of long flows in numerical analysis closely follow the corresponding simulation results.

C. Protect Short Flows Under Extreme Burstiness

In Section IV-A, we analyze how to adjust the switching granularity of long flows to ensure that the short flows complete within their deadlines. This scheme shows effectiveness in handling bursty traffic caused by a moderate number of short flows. However, this scheme still cannot cope with the situation when the number of concurrent short flows is extremely large. In this subsection, we analyze the impact of extreme burstiness of short flows and then give the corresponding solution to protect short flows.

1) Problem Analysis: Traffic bursts are very common in modern data centers. For example, in Online Data Intensive (OLDI) applications, thousands of servers are likely to respond nearly at the same time to queries, causing multiple fan-in bursts at the aggregated switches [17], [26]. In distributed machine learning jobs, large number of workers simultaneously fetch the latest parameters to update a global model at each synchronized training iteration, generating extremely bursty traffic [27]. In such extremely bursty scenarios, even the long flows are transmitted with the largest switching granularity, there is still not enough available bandwidth to ensure that all short flows meet their deadlines.

We conduct NS2 simulations to illustrate the impact of increasing number of short flows on TLB. We use the leaf-spine topology with 15 parallel paths. The other simulation settings are same as that in Section IV-B. The short flows start between random pairs of end-hosts following a Poisson process. We gradually increase the number of short and long flows from 20 to 200 and 2 to 4, respectively.

Firstly, we measure the deadline missing ratio of short flows. Fig. 7 (a) shows that the deadline missing ratio keeps close to 0 until the number of short flows increases to a large value. For example, when the number of short flows is less than 100, the deadline missing ratio is nearly 0. However, when the number of short flows increases to 140, corresponding to 2, 3 and 4 long flows, 13.6%, 27.3% and 35.6% of short flows miss their deadlines, respectively. Then, we test the corresponding AFCT of short flows. Fig. 7 (b) shows that the AFCT increases with larger number of short flows, resulting in performance degradation.

2) Packet Slicing for Long Flows: Broadly speaking, the short flows have much smaller the lifetime than the long ones in data center traffic, exhibiting the bursty pattern. Since the general inactivity gap between two bursts of short flows is 500μs in data centers [1], [9], we adopt 500μs as the timescale to update the bursty threshold m'_S for distinguishing the extremely bursty scenarios. In addition to timescale, m'_S is also related to traffic pattern and traffic load. Based on the number of short flows and the number of paths occupied by short flows, TLB obtains the arrival rate and load strength of short flows. Then TLB calculates the expected queueing delay and flow completion time of short flows according to the load strength. Finally, the bursty threshold m'_S for distinguishing the extremely bursty scenarios is calculated to ensure that short flows meet their deadlines.

Specifically, TLB periodically measures the number of short flows m_S every 500μs at the switch. If the number of short flows m_S increases to the bursty threshold m'_S, TLB deems currently arrival traffic extremely bursty and the switch triggers the packet slicing scheme [28] to reduce the packet size of long flows to release bandwidth. Then more short flows are able to make use of the spare bandwidth to meet deadline requirements. Therefore, since adjusting the bursty threshold m'_S to trigger the packet slicing according to the load strength ρ, TLB works well under the dynamic traffic load.

We use an example to show the effectiveness of packet slicing. As shown in Fig. 8 (a), there are 3 parallel paths, two of which are occupied by bursty short flows with deadline as 4. One long flow without deadline requirement uses the 3rd path. Although the long flow do not switch paths, the remaining two paths are not enough for the short flows to finish before deadline. In Fig. 8 (b), the packet size of long flow is reduced to release more bandwidth for short flows. Specifically, each packet of the long flow is cut to 1/3 of the original packet, then 2 packets of short flows have opportunity to seize the spare bandwidth. Finally, none of short flows miss deadline.

We explain the theoretical relation between bursty threshold, timescale, traffic pattern and load as follows.

(1) **Bursty threshold and timescale:** TLB periodically updates the bursty threshold m'_S every 500μs to distinguish the extremely bursty scenarios.

(2) **Bursty threshold and traffic pattern:** Under the dynamic traffic pattern, the numbers of long and short flows change frequently. Thus, to obtain a proper bursty threshold, TLB updates the number of paths used by different flows. Under the extremely bursty scenario, each long flow only takes a single path without switching in its whole transmission, the maximum number of paths occupied by the long flows n_{L_max} is equal to the number of long flows m_L. Therefore, the number of remaining paths for short flows n'_S is obtained

as $n'_S = n - n_{L\_max} = n - m_L$, where n is the total number of parallel paths.

(3) **Bursty threshold and traffic load:** TLB estimates the load strength ρ of bursty short flows according to their arrival rates λ. We assume that m'_S short flows with average size of X bytes are transmitted over n'_S paths with an average completion time of FCT'_S. The load strength ρ bursty short flows is obtained as

$$\rho = \frac{\lambda}{C} = \frac{m'_S \cdot X}{FCT'_S \cdot n'_S \cdot C} = \frac{m'_S \cdot X}{FCT'_S \cdot (n - m_L) \cdot C}, \quad (12)$$

where C is the service rate in packets per second.

With Equation (5), (9) and Equation (12), we calculate the mean FCT of short flows FCT'_S as

$$FCT'_S = \frac{m'_S \cdot X \cdot \frac{r}{C}}{2[FCT'_S \cdot (n - m_L) \cdot C - m'_S \cdot X]} + \frac{X}{C}. \quad (13)$$

Thus, we obtain the value of m_S when FCT'_S equals to the flow deadline D as

$$m'_S = \frac{2DC \cdot (n - m_L) \cdot (X - DC)}{X \cdot (r - 2X + 2DC)}. \quad (14)$$

When the measured number of concurrent short flows m_S exceeds the threshold m'_S, the network scenario is deemed extremely bursty. TLB triggers the packet slicing scheme [28] to reduce the packet size of long flows to release bandwidth for short flows to meet deadlines.

Next, we analyze the slicing ratio k of long flows. For example, if packet size is halved, the corresponding slicing ratio is 2. After the packet slicing is triggered, the corresponding number of available paths for short flows n''_S can be presented as

$$n''_S = n - \frac{m_L}{k}. \quad (15)$$

According to Equation (2), (5), (9) and (15), we have the equation about the mean FCT of short flows FCT''_S as

$$FCT''_S = \frac{m_S \cdot X \cdot \frac{r}{C}}{2[FCT''_S \cdot (n - \frac{m_L}{k}) \cdot C - m_S \cdot X]} + \frac{X}{C}. \quad (16)$$

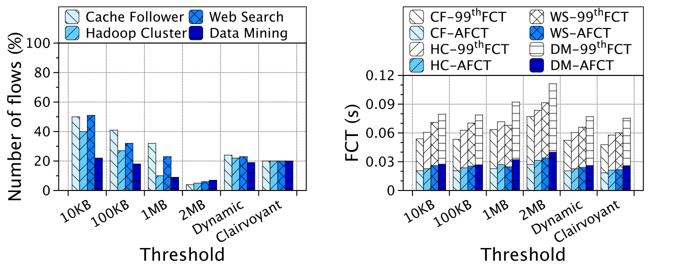
To protect short flows from missing deadline D, FCT''_s should satisfy $FCT''_S \leq D$. Therefore, the slicing ratio k of long flows should satisfy the following expression

$$k \geq \frac{m_L}{n - \frac{m_S \cdot X \cdot \frac{r}{C}}{2(D - \frac{X}{C}) \cdot D \cdot C} - \frac{m_S \cdot X}{D \cdot C}}. \quad (17)$$

In order to ensure the throughput of long flows as high as possible, we use the minimum value of k satisfying Equation (17) in our design.

D. Threshold Analysis of Distinguishing Flow Type

As the traffic workload pattern changes over time, an improper threshold for distinguishing short and long flows degrades the performance of TLB. On the one hand, if the threshold is too large, some long flows are mistakenly distinguished as short ones, resulting in increasing of queueing delay of short flows. On the other hand, under a too small threshold, some short flows are mistakenly distinguished as long ones. The short flows handled as long ones are not able to flexibly utilize multiple paths to finish quickly.



(a) Long flows identified by TLB (b) AFCT and 99^{th}-ile FCT of TLB

Fig. 9. The impact of thresholds for realistic workloads. Specifically, CF, HC, WS and DM stand for Cache Follower, Hadoop Cluster, Web Search and Data Mining, respectively.

According to the flow size distribution in data centers, that is, most flows are less than 100KB [13], [14], [20]–[23], [29], TLB sets the flow size threshold as 100KB to distinguish short and long flows. At the beginning, all flows are treated as short flows. Once the number of sent bytes in one flow exceeds the threshold (i.e., 100KB), the flow is considered as a long one.

We conduct testbed experiments to analyze the impact of thresholds under four different realistic workloads. The experimental settings are as same as that in Section A. We test four different static thresholds (i.e., 10KB, 100KB, 1MB, 2MB) without any prior knowledge of flow size distribution. With prior knowledge, the clairvoyant threshold is set to the 80^{th} percentile size of all flows. Fig.9 (a) shows the number of flows distinguished as long ones by TLB. When the threshold is 100KB, the numbers of long flows distinguished by TLB under Hadoop Cluster and Data Mining are close to that of clairvoyant threshold, because around 20% flows are larger than 100KB in these workloads. Some short flows are mistakenly classified as long ones under Cache Follower and Web Search with the 100KB threshold, resulting in larger number of long flows than the clairvoyant threshold. Fig.9 (b) shows the average and 99^{th}-ile FCTs of all flows. When the threshold becomes larger, more long flows are distinguished as short ones, resulting in larger queueing delay of short flows and longer FCTs of all flows.

To keep up with dynamic traffic patterns more accurately, we propose a simple yet effective scheme called TLB^{dth} to dynamically adjust the threshold for distinguishing short and long flows at the receivers. Specifically, since multiple senders send flows to the receiver in typical many-to-one transmission scenarios, TLB^{dth} maintains active and finished flow tables at each receiver to keep track of traffic variations. When a flow is established, a new entry is inserted into the active flow table and the received bytes are updated. When a flow is finished, the corresponding entry is removed from the active flows table and a new entry is added in the finished flows table to record its flow size. The flow size threshold is updated periodically (e.g., 500μs) to the 80^{th} percentile size of finished flows. Since historical traffic cannot reflect the future traffic perfectly, mismatches between threshold setting and underlying traffic are inevitable.

However, before the long flows are identified, the negative impact on the short flows is very small. Firstly, the threshold for detecting large flows is relatively small, which is close to the size of the largest small flows. Since short flows randomly start transmission during the whole live time of long flows, the new short flows will not be affected by the long flows after the long flows are identified. Secondly, the number of large flows is relatively small, accounting for about 20% of the total number of flows. Thus, the aggregated negative impact of unidentified long flows on short ones is still bounded.

Once the received bytes of a flow exceed the threshold, this flow is distinguished as a long one. To notify the sender, the receiver sets the highest bit of Type of Service (ToS) field in ACK header to 1. After receiving this notification from receiver, the sender marks the highest ToS bits of all the remaining data packets of this flow to 1, to guarantee the classification consistency. Finally, for each arrival packet, the switch distinguishes its flow type according to the highest ToS bit in packet header. Therefore, the notification delay is less than one RTT. Note that, if none packet is received during the threshold update period, the corresponding flow record is removed from the active flow table. Compared with the fixed threshold, Fig.9 (a) shows the number of long flows distinguished by TLB$^{\mathrm{dth}}$ under dynamic threshold are closer to the clairvoyant value. Fig.9 (b) shows TLB$^{\mathrm{dth}}$ achieves the lower average and 99^{th}-ile FCTs of all flows than the other static thresholds.

TLB$^{\mathrm{dth}}$ is proposed as an alternative version of TLB to dynamically adjust the threshold for distinguishing short and long flows at the receivers. TLB$^{\mathrm{dth}}$ can be easily deployed on all servers in a controllable environment. However, TLB$^{\mathrm{dth}}$ becomes infeasible if the administrator cannot control protocol stack of all servers in some complex cases such as multi-tenant data centers. In our future work, we will investigate sketch methods [30]–[33] of distinguishing long and short flows at switches to improve the scalability and efficiency of TLB.

V. Implementation

We implement TLB on a commodity programmable switch and use P4 programming language to specify how the switch processes packets. To store the variable value, the P4-programmed switch provides a limited amount of registers, which are implemented in on-chip static random access memory (SRAM) [34]–[36]. However, unlike the extremely small number of registers in CPU architecture, P4 switch can extend the number of registers by using more SRAM. In terms of access manner, register in P4 switch is stateful memory whose value can be directly read and written at line rate according to the address index [35], [37], [38]. The challenge of implementing TLB in Application Specific Integrated Circuit (ASIC) is to ensure accessing resources exclusively in the pipelined packet processing. The ingress pipeline for data packets of TLB implementation is shown in Fig.10, which contains multiple primary match-action tables.

Specifically, in the $Flowtype$ table, if the ingress metadata $flow.sent.size$ representing the number of bytes sent by a flow is less than 100KB, the flag of long flow lf is set to 0, otherwise lf is set to 1. TLB reroutes the long flows according to the number of short flows, which is recorded in the ingress metadata $sf.num$. In the Clonepkt table, the packet slicing scheme is triggered if $sf.num$ is larger than m'_S, which is calculated according to Equation (12). Meanwhile, the arrival packet of long flow is cloned from ingress to egress by using the primitive action clone. The intrinsic metadata of the cloned packet is modified as the same as Internet Control

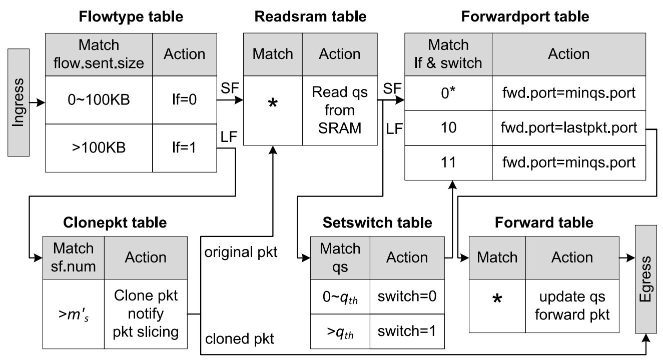


Fig. 10. Ingress pipeline for data packets in TLB's P4 implementation.

TABLE I

Resource Consumption of Different Schemes

Resource	ECMP (flow)	RPS (packet)	LetFlow (flowlet)	TLB (packet)
Match Crossbar	2.41%	1.24%	4.82%	6.18%
Hash Bits	3.08%	2.44%	5.67%	6.68%
Gateway	1.39%	1.04%	1.56%	10.04%
SRAM	1.56%	1.25%	3.33%	4.79%
VLIW Actions	1.56%	1.3%	2.34%	4.69%
ALU Instruction	2.6%	2.08%	5.2%	7.81%

Message Protocol (ICMP) packet, which includes 20 bytes of IP header, 8 bytes ICMP header and 8 bytes of ICMP data. The type and code fields in the ICMP header are set to 3 and 4, respectively. The Next-Hop MTU field in ICMP header is updated as the original packet size divided by k obtained from the Equation (15). In the IP header, the ICMP message's destination address is changed to the source address of the original packet. The source and destination ports of the long flow are filled in the ICMP data field. Then the cloned ICMP packets carrying the packet slicing information are directly forwarded to the senders. The original packet selects destination port in the next stage.

The implementation leverages the current queue size of egress ports to choose output port. However, there is no such information in the ingress intrinsic metadata on hardware P4 platform. Thus, we implement the queue size SRAM qs with a similar functionality by counting the packets that enqueue and dequeue the egress port. In the $Readsram$ table, the queue length is read from the queue size SRAM qs by using stateful ALUs. Then, in short flows, the packet chooses the destination port $fwd.port$ with the minimum queue length in the $Forwardport$ table. In long flows, the packet enters into $Setswitch$ table. Once the queue size of output port of the previous packet in the same flow exceeds the threshold q_{th}, the switching path flag $switch$ is set to 1 and then $fwd.port$ is set to the port with the minimum queue size $minqs.port$ in the $Forwardport$ table. Otherwise, $switch$ is set to 0, and $fwd.port$ is set to the same output port of the previous packet in the same flow. Finally, the queue size of selected port is updated by using $resubmit$ primitive. The packet is forwarded according to the $Forward$ table in the ingress pipeline.

Table I shows the hardware usage of various resources of four schemes in our testbed test. RPS has the lowest overhead for all resources due to its random forwarding. Compared to the other schemes, TLB uses a relatively larger proportion of resources, because TLB needs more pipeline stages to identity short and long flows, calculate the switching granularity of long flows, and compare the queue length of all ports. Meanwhile, TLB needs more SRAM and stateful ALUs to store and update the flow size, the number of short flows and the queue size of the egress ports. Nonetheless, the overall resources consumption for TLB remains very low, meaning that TLB can be deployed in high-speed switch with reasonable resources consumption.

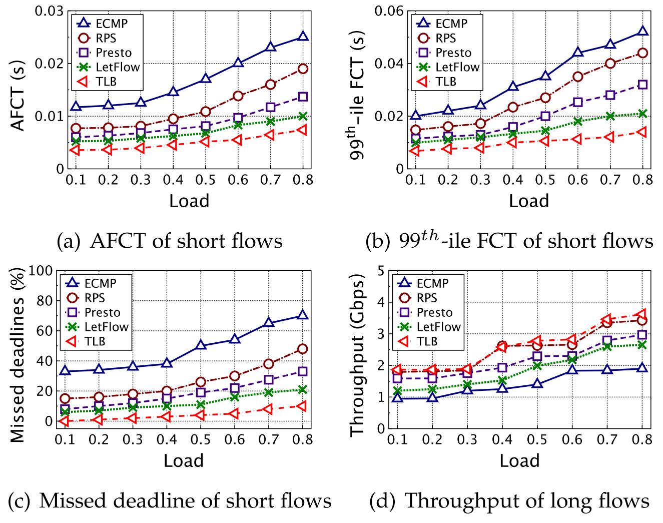
There are some considerations in TLB implementation. Although most OLDI datacenter applications such as web search, social networking and retail have strict deadline constraints (i.e., 300ms latency) [17], [22], [39], some datacenter applications such as HTTP chunked-transfer and database access generate short flows without deadlines [22]. Moreover, various applications may have different deadline requirements of short flows. In the case of lacking deadline information of short flows, we use an alternative method which deduces the specified flow deadline from the statistics of network traffic. Specifically, we set the deadline to the value of 25^{th} percentile in CDF of the statistical flow deadlines. The corresponding experimental results in Section VI-B show that TLB works well in dark.

Since the switches have no prior knowledge of the flow size, TLB distinguishes short and long flows according to the number of bytes a flow has already sent. Specifically, TLB records a flow as a key-value pair (key, count), where key is the flow ID, and count is the number of sent packets belonging to this flow. A flow is identified by the unique 5 tuples, i.e., source/destination IP addresses, protocol, and source/destination ports. When a packet enters the ingress pipeline, TLB uses CRC16 algorithm as the hash function that converts 5 tuples in packet header into flow ID. For each flow record, TLB uses a flow ID of 16bits, a counter of 7bits, a flow type flag of 1bit, a switching path flag of 1bit and the output port of the previous packet of 7bits, with a total of 32 bits. Since the long flows are identified at the receivers, TLB^{dth} distinguishes the flow type at all switches according to the highest ToS bit in packet header marked by the senders, which does not cause additional lookup delay. Therefore, TLB^{dth} only adds 1bit of flow type in each flow entry at all switches. Compared with TLB, TLB^{dth} does not need to record the number of sent packets for each flow at switch, reducing storage and computing overhead at switches. For TLB^{dth}, there is no 7bits counter for each flow, thus saving 21.8% SRAM space. According to the measurement results of datacenter traffic in [12] and [14], there are generally hundreds of concurrent flows on a server and up to 10,000 concurrent flows through a switch. For 10,000 flows, TLB^{dth} needs 31.25KB SRAM to record the flow states. Today's P4 switches support up to 50∼100 MB SRAM [40], which is sufficient for TLB^{dth} to handle a reasonably large number of concurrent flows.

VI. Simulation Evaluation

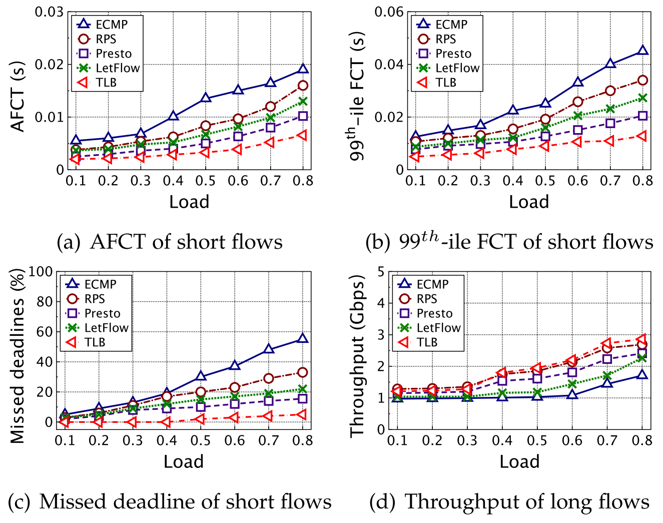
A. Large-Scale Test

We conduct the large-scale simulation to evaluate TLB's performance in the typical applications in data centers. We choose the web search and data mining applications [20], [22], [23]. In these workloads, the distributions of flow size are heavy-tailed. For the web search workload, about 30% flows are larger than 1MB, while in data mining scenario there are around less than 5% flows larger than 35MB.



(a) AFCT of short flows
(b) 99^{th}-ile FCT of short flows
(c) Missed deadline of short flows
(d) Throughput of long flows

Fig. 11. Web search application.



(a) AFCT of short flows
(b) 99^{th}-ile FCT of short flows
(c) Missed deadline of short flows
(d) Throughput of long flows

Fig. 12. Data mining application.

We compare TLB with the following four state-of-the-art load balancing schemes. ECMP hashes each flow to one of the available paths based on the 5 tuples in the packet header. RPS randomly sprays each packet to all available paths to make traffic evenly distribute among parallel paths. Presto performs rerouting at the same granularity with uniform unit of flowcell (i.e., 64KB), which are assigned over multiple paths evenly in a round-robin fashion. LetFlow randomly switches flowlets among equal-cost paths.

We use the common leaf-spine topology with 8 ToR and 8 core switches [2]. The whole network has 256 hosts connected by 1Gbps links. The round-trip propagation delay is $100\mu s$ and the switch buffer size is set to 256 packets. The traffic is generated by randomly starting flows via a Poisson process between random pair of hosts. The deadlines of short flows are randomly distributed between [5ms, 25ms]. The switching granularity of long flows is updated every $500\mu s$. We vary the overall workload from 0.1 to 0.8 to thoroughly evaluate TLB's performance.

Fig. 11 (a), (b) and Fig. 12 (a), (b) show the average and 99^{th} percentile FCT of short flows under the web search and data mining workloads, respectively. TLB improves AFCT and tail FCT significantly compared with the other four schemes, especially in high workloads. Specifically, in the web search scenario, TLB reduces AFCT by ∼68%, ∼55%, ∼45% and ∼25% at 0.8 workload over ECMP, RPS, Presto and LetFlow, respectively. Fig. 11 (c) and Fig. 12 (c) show the deadline missing ratio of short flows. TLB ensures that more than 90% short flows meet their deadlines.

These test results demonstrate the advantage of TLB with adaptive switching granularity for different kinds of flows. Under the same switching granularity, when the workload becomes high, more mixed flows are queued in the same output ports at the switches, resulting that more short flows hit the long tail and experience packet reordering. Especially, since most short flows experience large queueing delay under large rerouting granularity (e.g., ECMP) or have lots of reordering packets under small switching granularity (e.g., RPS), the delay performance is degraded. For LetFlow, the performance is better in the high load because more flowlet gaps occur for switching under bursty and congestion scenarios.

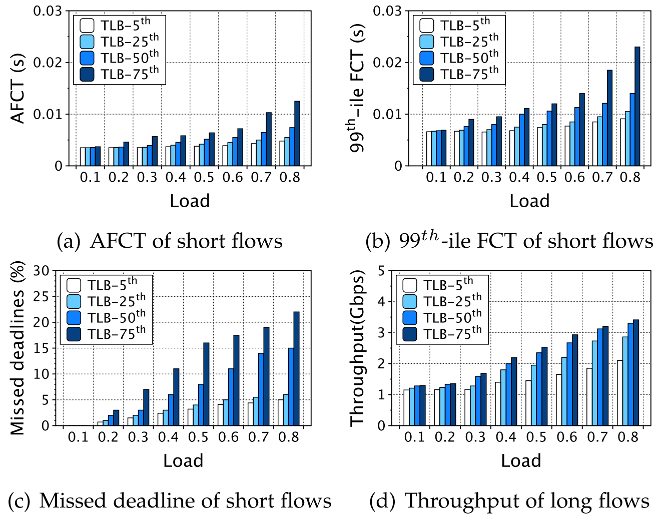
Moreover, we find that short flows in the web search workload have larger FCT than those in the data mining. The reason is that there are more medium flows with the size from 100KB to 1MB and more long flows in the web search, leading to larger queue length and more packet reordering. While in the data mining, the size between large number of short flows and a few long flows has obvious boundary, thus resulting in less out-of-order packets. For LetFlow scheme, however, the performance in the data mining scenario is worse than that of in the web search because there are fewer flowlet gaps to change path.

We also test the throughputs of long flows. As shown in Fig. 11 (d) and Fig. 12 (d), the long flows in the schemes with large rerouting granularity suffer the greater throughput degradation. Since TLB flexibly adjusts the size of switching granularity for long flows based on the traffic strength of short flows, it makes good use of multiple paths and thus achieving the high throughputs of long flows.

B. Deadline-Agnostic Case

In some cases, it is hard to get the deadline information of short flows in advance. Moreover, since some latency-sensitive applications have various delay requirements, the deadlines of short flows may be different. For these scenarios without prior deadline knowledge or with different deadlines, we use an alternative method to protect short flows. Specifically, we set a fixed deadline for all short flows according to the statistics of deadline distribution. In this test, the accurate deadline of each short flow is randomly distributed between [5ms, 25ms] as illustrated in [17]. We specify the 5^{th}, 25^{th}, 50^{th} and 75^{th} percentile deadlines with the specified values of 5ms, 10ms, 15ms and 20ms, respectively [17]. The simulation topology and settings are as same as that in the large-scale simulations in Section VI-A. We measure the AFCT, 99^{th}-ile FCT and deadline missing ratio of short flows, and the throughputs of long flows under the web search application scenario.

Fig. 13 shows the performances of deadline-agnostic TLB. As shown in Fig. 13 (a) and (b), the AFCT and 99^{th}-ile FCT become larger with the increasing workload. Moreover, the cases of TLB-5^{th} and TLB-25^{th} obtain the small FCT compared with the other cases. Fig. 13 (c) shows the deadline missing ratios of TLB-5^{th} and TLB-25^{th} are much lower than the laxer deadline settings. In Fig. 13 (d), the cases of TLB-25^{th}, TLB-50^{th} and TLB-75^{th} obtain the almost same throughputs of long flows, which are much higher than that of TLB-5^{th} case. Therefore, we set the deadline of all short flows as the 25^{th} percentile of statistical deadlines, which achieves the best performance in the deadline-agnostic TLB.



(a) AFCT of short flows (b) 99^{th}-ile FCT of short flows

(c) Missed deadline of short flows (d) Throughput of long flows

Fig. 13. Performances of deadline-agnostic TLB.

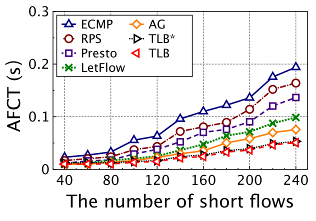
VII. Testbed Evaluation

In this section, we conduct experiments on a physical testbed shown in Fig.1. The default topology consists of 3 senders and 3 receivers with 10 cores Intel Xeon W-2255 CPU, 64GB memory, Mellanox ConnectX-5 EN 100GbE dual-port QSFP28 NICs and Ubuntu 18.04 (Linux version 4.15.0-1090). These servers connect to the corresponding leaf switch with 100Gbps links. Between two leaf switches, there are 6 parallel paths, each of which has 40Gbps bandwidth. We implement 6 different schemes including ECMP, RPS, Presto, LetFlow, AG [41] and TLB at the hardware programmable switch by using P4 programming language. AG adaptively adjusts switching granularity according to the asymmetric degree of all paths and reroutes randomly.

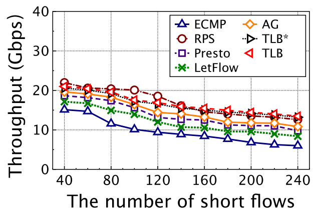
A. Performances Under Symmetric Scenario

In this test, we evaluate TLB performance with and without prior knowledge of flow size (i.e., TLB* and TLB) under varying number of short and long flows. The default numbers of short flows (<100KB) and long flows (>5MB) are 200 and 3, respectively. The overall traffic obeys the heavy-tailed distribution in web search workload [14]. The deadlines of short flows are randomly distributed between [0.1s, 0.3s]. We use the 25^{th} percentile deadline (i.e., 0.15s) to calculate the switching threshold of queue length for rerouting long flows. The flowlet timeout and updating interval of switching granularity are 500μs [9].

At first, we gradually increase the number of short flows from 40 to 240 with fixed 3 long flows. Then we increase the

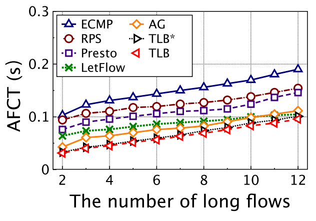


(a) AFCT of short flows

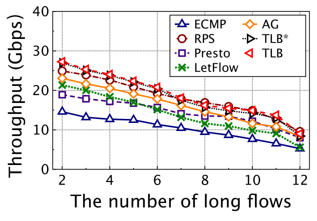


(b) Throughput of long flows

Fig. 14. Varying the number of short flows.

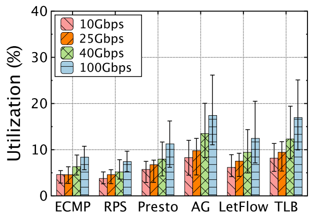


(a) AFCT of short flows

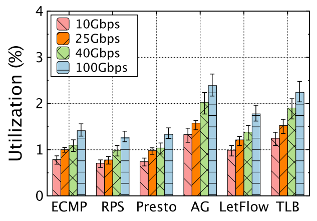


(b) Throughput of long flows

Fig. 15. Varying the number of long flows.



(a) CPU utilization



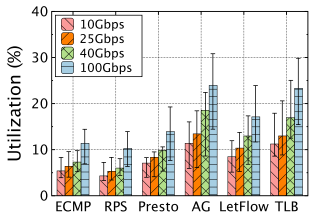
(b) Memory utilization

Fig. 16. Overhead of the leaf switch with varying short flows.

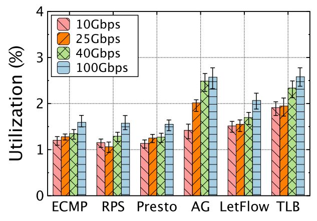
number of long flows from 2 to 12 with fixed 200 short flows. Fig.14 (a) shows that, with more short flows, TLB reduces the AFCT of short flows by ∼41-73%, ∼36-68%, ∼22-59%, ∼16-50% and ∼7-28% over ECMP, RPS, Presto, LetFlow and AG, respectively. As shown in Fig.14 (b), TLB improves the average throughput of long flows by ∼40-93%, ∼16-35%, ∼23-49% and ∼9-23% over ECMP, Presto, LetFlow and AG, respectively. With more long flows, Fig.15 (a) and (b) show that TLB achieves better performance compared with the other schemes by adaptively adjusting the switching granularity of long flows. Due to the inflexibility in rerouting flows, ECMP and LetFlow suffer from the long-tailed delay and low link utilization. For RPS and Presto, the packet reordering degrades their performances especially under the heavy workload.

Moreover, though TLB has not any prior knowledge of the flow size, the threshold for long flows is only 100KB, which is much smaller than the long flow size, and the number of long flows is very small, accounting for less than 10%, the impact of long flows on load balancing is negligible when the long flows are mistaken as short flows in the beginning. Therefore, TLB achieves almost the same performance as TLB*.

Fig.16 and Fig.17 show the maximum, minimum and average values of CPU and memory utilization ratios of switch with varying short and long flows, respectively. For ECMP and RPS, due to their simple flow-level hash and packet-level random routing, the CPU utilization is very low as shown in Fig.16 (a) and Fig.17 (a). The other schemes incur a little



(a) CPU utilization



(b) Memory utilization

Fig. 17. Overhead of the leaf switch with varying long flows.

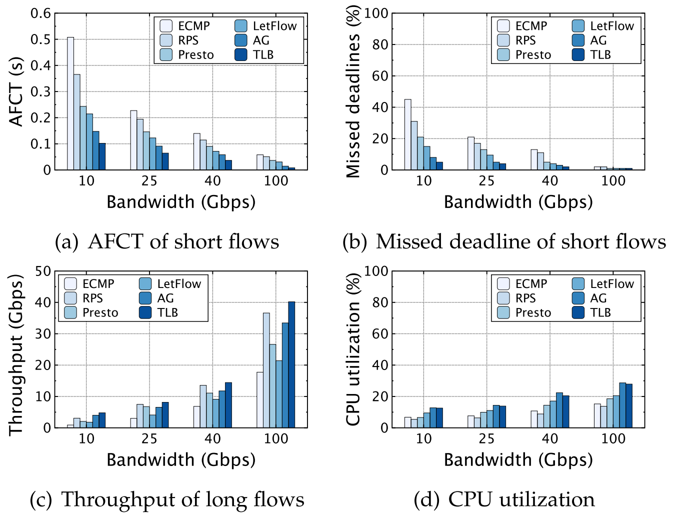
more CPU overhead and memory utilization due to their more sophisticated operations in rerouting packets. Compared with the other schemes, TLB brings about reasonable memory utilization and does not incur excessive CPU overhead. For example, in the case of 100Gbps bandwidth, Fig.16 shows that the average utilizations of CPU and memory under TLB are less than 17% and 2.24%, respectively. In addition, as the link rate increases from 10Gbps to 100Gbps, the utilizations of CPU and memory increase in all schemes because the switch needs to process packets more quickly. Overall, the test results show that TLB is easily deployed in the actual reconfigurable switch with acceptable CPU and memory overheads.

B. Performances Under Varying Rates and Paths

We implement TLB in P4 programming language and compile it to a hardware programmable switch target. As shown in Fig.10, the P4 program of TLB is compiled into a pipeline, which consists of multiple stages that can only be executed serially. In current ultra-high-speed switches, the delay overhead of reading and updating queue lengths of all egress ports potentially results in inaccurate globe information. Moreover, in TLB' P4 implementation, the queue length in the egress intrinsic metadata cannot be directly used in the ingress pipeline control. Therefore, in order to reduce the overhead of querying all egress ports information on per-packet granularity, we estimate the queue length of egress ports at the ingress control logic.

Specifically, when a packet arrives at the switch and enters the ingress pipeline, it reads the queue length of each egress port from the corresponding *Readsram* table by using stateful ALUs. After the packet selects the destination port, the queue length of destination port is updated. The queue size of each egress port is calculated according to the number of packets that enqueue and dequeue the egress port within the time interval between two consecutive packets arriving at the egress port. Here, we conduct the testbed experiments to test the estimation overhead and accuracy with varying link rate and number of output ports.

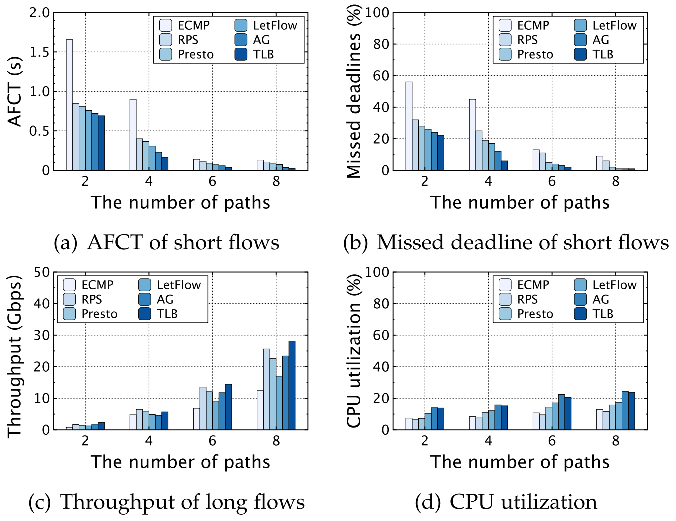
Firstly, we test TLB performance with bandwidth ranging from 10 Gbps to 100Gbps. As shown in Fig.18 (a) and (b), under different bandwidths, TLB outperforms the other schemes in average FCT and deadline missing ratio since less short flows are blocked by the long ones. Meanwhile, Fig.18 (c) shows that TLB achieves high throughput of long flows by adjusting the switching granularity. Fig.18 (d) shows the average CPU utilization of switch. Considering the performance improvements of short and long flows, the CPU utilization overhead of TLB is still acceptable.



(a) AFCT of short flows (b) Missed deadline of short flows

(c) Throughput of long flows (d) CPU utilization

Fig. 18. Increasing link bandwidth.



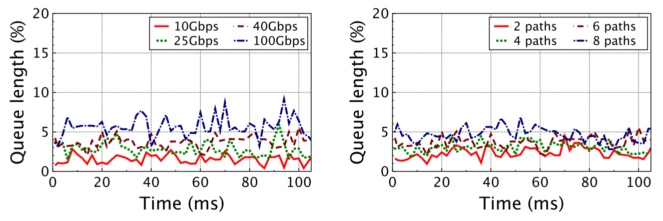
(a) AFCT of short flows (b) Missed deadline of short flows

(c) Throughput of long flows (d) CPU utilization

Fig. 19. Increasing number of parallel paths.

Secondly, we increase the number of parallel paths from 2 to 8. The link bandwidth is 40Gbps. Fig.19 shows that, with more parallel paths, TLB needs to read the queue lengths from more SRAM and update length information for more queues. However, compared with the benefits of TLB's flexible load balancing mechanism, the increased CPU overhead due to reading and writing SRAM is negligible. For example, under 6 parallel paths, TLB reduces AFCT of short flows by 51% and improves the throughput of long flows by 44% with about 20% CPU utilization.

We record the estimated queue length q_e of destination port in the ingress pipelined control and the real queue length q_r in the egress intrinsic metadata. We use the queue length deviation $\frac{|q_e - q_r|}{q_r} \times 100\%$ to evaluate the estimation accuracy of queue length. As shown in Fig.20 (a) and (b), under different bandwidths and number of parallel paths, the estimated queue length is close to the real one. For example, the queue length deviation is less than 5% under 40Gbps bandwidth and 6 parallel paths scenario. The proportions of positive and negative deviation are almost equal. The overestimation and underestimation of queue length may



(a) Varying link bandwidth (b) Varying parallel paths

Fig. 20. Queue length deviation.

TABLE II
VARYING TRAFFIC PATTERNS AND LOAD

Time (ms)	20	40	60	80	100
Traffic ratio (SF:LF)	1:9	2:8	3:7	4:6	5:5
Case1 (short flow size (KB))	11	25	42	66	100
Case2 (number of short flows)	22	50	84	132	200
Case3 (Traffic load)	0.2	0.3	0.4	0.5	0.6

lead to inaccurate switching granularity of long flows. The underestimation of queue length leads to larger switching granularity of long flows. Consequently, the short flows obtain more bandwidth and complete faster within their deadlines. On the contrary, the missing deadline ratio of short flows will increase due to insufficient available bandwidth. However, due to the small deviation between the estimated and real values of queue length, as well as the randomness of favorable and unfavorable deviation, the impact on load balancing is very small. As shown the experimental results, the short flows achieve stably better performance in TLB compared to the other schemes.

C. Performances Under Varying Traffic Patterns

From a macro perspective, the heterogeneous traffic exhibits the stable heavy-tailed distribution in data centers [12]–[14]. From a micro viewpoint, however, datacenter traffic is very bursty and unpredictable at short timescales (e.g., 10∼100s of microseconds) [1]. TLB detects traffic changes by periodically updating the number of flows. Since TLB dynamically adjusts the switching granularity of long flows according to the number of short flows, TLB is resilient to the time-varying traffic patterns. Specifically, each packet of short flows selects the path with the shortest queue length under any traffic pattern. The long flows also choose the shortest queue when rerouting.

To test TLB performance in varying traffic patterns and load, we conduct testbed experiments by changing the ratio of short flows to long ones and traffic load over time. There is one long flow with 10MB. As shown in Table II, in the first case of varying traffic pattern, we increase the average size of 100 short flows per 20ms. In the second case, we increase the number of short flows with a fixed size of 50KB per 20ms. In the third case, both the traffic load and traffic ratio change over time. We vary the traffic load from 0.2 to 0.6. Meanwhile, the traffic ratio of short flows to long ones changes from 1:9 to 5:5. The test topology is shown in Fig.1. We vary the bandwidth of parallel paths from 10Gbps to 100Gbps.

Fig.21 shows that TLB outperforms other schemes in terms of the average FCT and 99th-ile FCT of short flows, because TLB flexibly adjusts the switching granularity of long flows

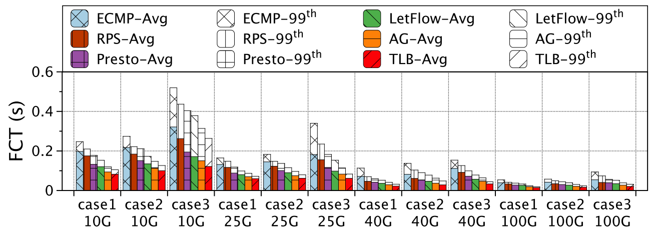


Fig. 21. Varying traffic patterns.

under varying traffic load and adjusts the bursty threshold timely to trigger the packet slicing according to the load strength in bursty scenario. Under presto, many short flows do not switch path due to their sizes less than 64KB. Under LetFlow, the short flows also rarely reroute since the packet interval is not large enough.

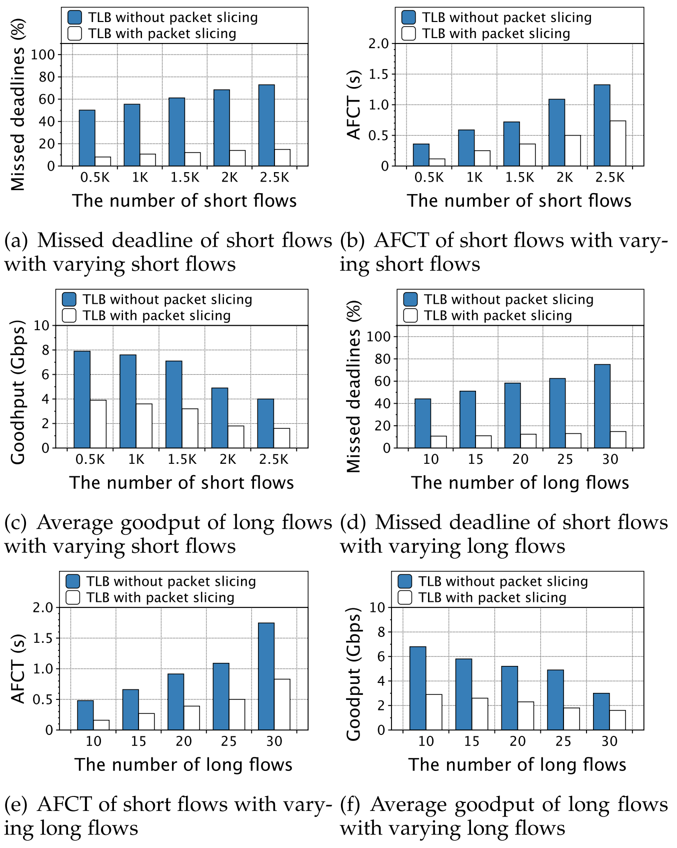
D. Performances Under Extremely Bursty Scenario

To evaluate the performance of TLB under highly concurrent flows, we conduct testbed experiments to compare the deadline missing ratio and AFCT of short flows, the throughput of long flows without and with packet slicing. The testbed consists of 5 senders and 3 receivers with 10 cores Intel Xeon W-2255 CPU, 64GB memory, Mellanox ConnectX-5 EN 100GbE dual-port QSFP28 NICs and Ubuntu 18.04 (Linux version 4.15.0-1090). These servers connect to the corresponding leaf switch with 100Gbps links. Between two leaf switches, there are 10 parallel paths, each of which has 40Gbps bandwidth. The other experimental settings are as same as that in Section VII-A.

According to the measurement results of datacenter traffic in [12] and [42], there are generally hundreds of concurrent flows on a server. In this test, the number of short flows generated by each server increases from 100 to 500, resulting in the total number of short flows changing from 500 to 2500. The size of short flows varies randomly from 10KB to 100KB, and the size of long flows varies randomly between 10MB and 30MB. The bursty short flows and long flows generated between random pairs of servers arrive in Poisson distribution. When multiple long flows arrive at a same receiver, the 100Gbps link connected with the receiver is the bottleneck. We set the deadline of short flows as 25^{th} percentile value according to the deadline distribution. The default numbers of short and long flows are 2000 and 25, respectively.

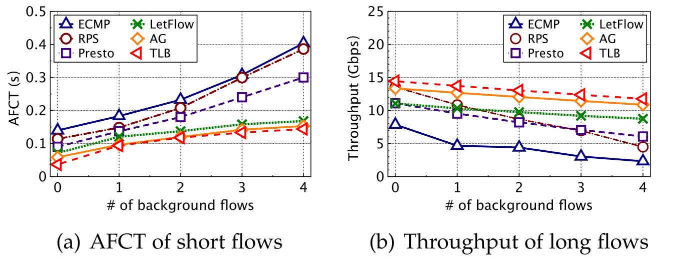
Fig.22 (a) and (b) show that, when the packet slicing scheme is enabled, the deadline missing ratio and AFCT are decreased by up to 80% and 44% compared with the cases without packet slicing, respectively. The reason is that, if even the largest switching granularity of the long flows cannot protect the very large number of short flows, the long flows will cut their packets to provide more spare bandwidth for helping short flows to meet their deadlines. As for long flows, however, the average goodput of each long flow is inevitably degraded due to packet overhead as shown in Fig.22 (c).

Next, we set the number of short flows to 2000 and change the number of long flows. Fig.22 (d), (e) and (f) show that, with more long flows, the packet slicing scheme significantly improves the performances of short flows. When the number of long flows increases from 10 to 30, for 2000 short flows, the deadline missing ratio is reduced to less than 15% and the AFCT remains very small. Fig.22 (f) shows the average goodput of each long flow. For example, in the case of 25 long flows in Fig.22 (f), the average goodput is about 5Gbps without packet slicing. For each link connected with the receiver, the average link utilization ratio is $\frac{25\times5}{100\times3} = 41.6\%$ when packet slicing is disabled.



(a) Missed deadline of short flows with varying short flows (b) AFCT of short flows with varying short flows (c) Average goodput of long flows with varying short flows (d) Missed deadline of short flows with varying long flows (e) AFCT of short flows with varying long flows (f) Average goodput of long flows with varying long flows

Fig. 22. Varying the number of short and long flows.



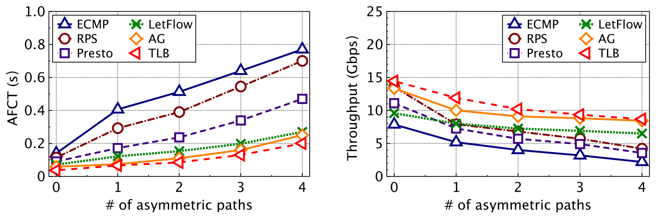
(a) AFCT of short flows (b) Throughput of long flows

Fig. 23. Varying delay in asymmetric scenario.

E. Performances Under Asymmetric Scenario

We test TLB performance in the asymmetric scenarios. The default experimental settings are as same as that in Section VII-A. We respectively vary the number of background flows from 0 to 4 on one randomly selected parallel paths and vary the number of paths from 0 to 4 with bandwidth of 40Gbps to change the degree of topology asymmetry.

As shown in Fig.23 and Fig.24, under the greater delay or bandwidth asymmetry, TLB achieves more performance



(a) AFCT of short flows (b) Throughput of long flows

Fig. 24. Varying bandwidth in asymmetric scenario.

improvement compared with ECMP, RPS and Presto. Under ECMP, the short and long flows experience large tailed delay and throughput degradation once being hashed to bad paths. For Presto and RPS, the flows suffer the packet reordering and perform badly due to the delay or bandwidth diversity under the asymmetric scenarios. Since using flowlet switching and adaptive switching, LetFlow and AG are resilient to network asymmetry and perform well especially under the high degree of asymmetry. Overall, TLB outperforms all other schemes because it is able to perceive traffic heterogeneity and flexibly reroute short and long flows with different granularities.

VIII. Related Work

In this section, we classify prior work on loading balancing for heterogeneous traffic in DCNs into flow-based, flowlet-based, flowcell-based and packet-based schemes below.

Flow-Based Schemes: ECMP forwards each flow by using flow hash. To avoid hash collision in ECMP, Hedera [45] dynamically schedules flows by a central controller to alleviate hotspots. MicroTE [46] schedules flows by leveraging the partial predictability of traffic matrix through OpenFlow switches. FlowBender [47] reroutes the flow when the congestion or link failure is detected. The authors in [48] propose a congestion-aware algorithm to assign flows to the available paths online.

Some other flow-based multi-path transmission schemes are proposed to schedule heterogeneous flows in different ways. Karuna [22] balances the interests of mix-flows with and without deadlines by using rate-control and priority-based flow scheduling. RepFlow [20] simply replicates each short flow to reap multi-path diversity in data centers. Freeway [49] adaptively partitions the transmission paths into low latency paths and high throughput paths respectively for the short and long flows. In [50], each flow is assigned to an available path with the minimum marginal network cost, achieving a good load balancing performance.

Recently, some work focuses on data center network features to solve the dynamic load balancing problem. In [51], the proposed load balancing algorithm divides a multi-tier fat-tree topology into multiple routing domains with independent routing processes for better efficiency and scalability. In [52], a set of specific converters are designed for multiple topologies in hybrid switching data center networks to reduce the maximum link utilization and achieve better load balancing. To improve scalability for mega data centers, the devolved controllers are used to balance workload among controllers and alleviate reconfiguration complexities in dynamic situations [53].

In brief, the above flow-based scheduling schemes have no packet reordering, but easily lead to the long-tailed latency or low-utilization problems under the highly dynamic traffic since the long flows are not able to switch flexibly.

Flowlet-Based Schemes: CONGA [1] uses in-network congestion feedback to estimate load and switches flowlets with the global congestion information to achieve good performance. LetFlow [9] randomly reroutes flowlets to naturally balance the traffic since the flowlet size changes automatically based on the traffic conditions on their paths. Flowtune [54] allocates an optimal transmission rate for each flowlet by using a centralized allocator. Clove [55] picks paths for flowlets in a weighted round-robin manner at the end-hosts. Expeditus [56] collects local congestion information and employs a two-stage path selection scheme to route flows or flowlets to improve network performance. HULA [57] is a data-plane load-balancing scheme for programmable switch. It directs flowlets to the best path according to the hop-by-hop congestion state. However, since the flowlet-based schemes only perform rerouting when the flowlets emerge, if the flowlet timeout value is large, they cannot always make full use of multiple paths due to rare flowlet opportunities [2].

Flowcell-Based Schemes: Presto [8] performs load balancing at the same granularity of fixed-size flowcell (i.e., 64KB) for both short and long flows, which are rerouted in a congestion-oblivious way. Presto needs to use the TCP offload functionality at the receiver to reassemble the out-of-order flowcells to prevent the reordered packets from being pushed up the networking stack. Luopan [58] periodically samples two random paths between the source and destination switches and forwards flowcells to the least congested one. As a sampling-based load balancing scheme with low overhead, Luopan performs well in asymmetric topologies and obtains lower flow completion time than Presto.

Packet-Based Schemes: RPS [6] randomly sprays all packets to multiple paths to achieve high network utilization. MMPTCP [21] randomly spreads the packets of short flows to reduce the FCT and transmits the long flows by MPTCP [59] to improve their throughputs. DRILL [3] switches each packet quickly and flexibly based on the local queue information by using a method similar to the power of two choices paradigm [60]. However, the above packet-based load balancing schemes potentially incur reordering especially under the asymmetric topology. AG [41] adaptively adjusts switching granularity according to the asymmetric degree of all paths to be robust to various path diversities. Hermes [2] timely and cautiously makes rerouting decisions only when it will be benefit. Compared with Hermes, instead of using ECMP, TLB flexibly routes short flows to all non-congested paths at packet granularity. Thus, the short flows avoid being blocked by the long ones and almost have no packet reordering.

Despite significant efforts, the above load balancing schemes lack the ability to dynamically adjust the switching granularity on different types of flows. Our solution TLB works through a new perspective: TLB adopts different granularities to switch short and long flows rather than all flows pick paths with the same granularity. The switching granularity of long flows is adaptively changed based on the traffic load of short flows. Thus, TLB successfully addresses the long-tailed queueing delay, low network utilization and packet reordering

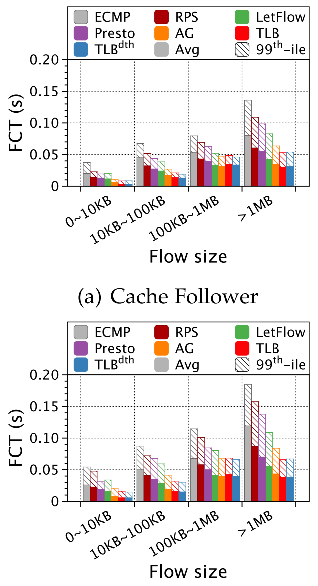


Fig. 25. Average and 99^{th}-ile FCT under realistic workloads.

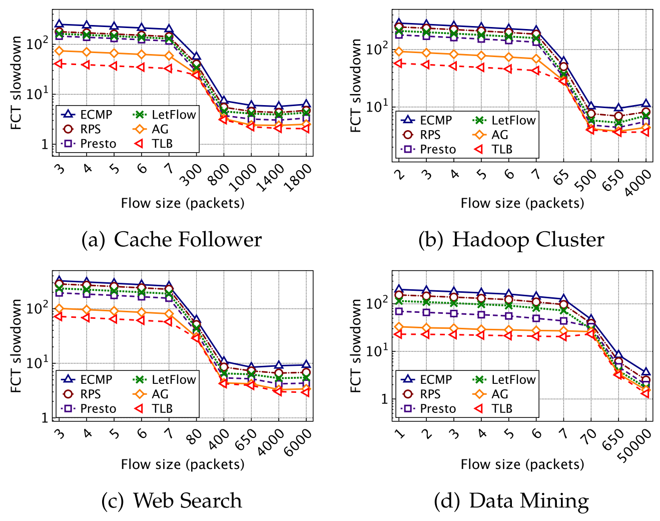


Fig. 26. FCT slowdown under realistic workloads.

problems to guarantee both the low latency of short flows and high throughput of long flows.

IX. Conclusion

We proposed a novel traffic-aware adaptive granularity load balancing design TLB that reduces flow completion time for short flows and simultaneously improves throughputs for long flows. Specifically, TLB adaptively adjusts the rerouting granularity of long flows according to the load strength of short ones. Therefore, the short flows have more opportunities to make full use of the less-congested paths unused by long flows. The long flows are also able to flexibly change the switching granularity to improve their throughputs. The results of testbed experiments and large-scale NS2 simulations demonstrate that TLB significantly reduces the AFCT of short flows by up to 67% under high workload over the state-of-the-art load balancing schemes and improves the throughput of long flows by up to 93% compared with flow-based load balancing schemes.

Appendix A
Performances Under Realistic Workloads

We evaluate TLB and TLB^{dth} over four realistic datacenter workloads, including cache follower, hadoop cluster, web search and data mining [2], [22], which cover the average flow sizes ranging from 701KB to 7.41MB and most flows are less than 100KB. In these four workloads, the flow size are heavy-tailed distributions. The traffic are randomly generated between 3 pair of hosts and arrive in Poisson process.

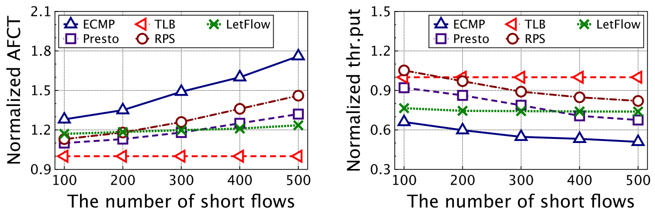
We measure the average and 99^{th}-ile FCTs across different flow sizes. As shown in Fig.25, TLB with the static 100KB threshold and dynamic threshold performs better than the other schemes. Since TLB^{dth} measures the flow size distribution and adaptively adjusts the threshold to keep up with dynamic traffic, TLB^{dth} achieves the best performance under four workloads. Moreover, TLB still works better under different workload patterns than the other schemes based on static threshold for the following reasons. Firstly, only a few short flows being mistakenly distinguished as long ones are adversely affected. Secondly, even if long flows are mistaken for short ones at the beginning, the negative impact is very small due to the small number of long flows and small threshold compared with the large size of long flows.

To quantify system overhead introduced by the TLB^{dth} for detecting and tagging long flows, we deploy it on a Dell PRECISION TOWER 5820 desktop with a Mellanox ConnectX-5 EN 100GbE dual-port QSFP28 NIC and measured CPU usage. We started 4 long flows and achieved ~96Gbps throughput. The extra CPU usage caused by TLB^{dth} is less than 2% compared with the case where the TLB^{dth} is not enabled.

Fig.26 shows the test results of slowdown, which is defined as the ratio of the measured FCT to the ideal FCT without any congestion. The X-axis are linear in the total number of flows (the first tick is 10% of all flows and each tick is increased by 10% of all flows). The results show that TLB achieves lower slowdown than other schemes for short flows less than 100KB across all workloads. For example, in Hadoop Cluster workload, TLB reduces slowdown by 77%, 72%, 66%, 58% and 29% with the flow size less than 10 packets on average over ECMP, RPS, Presto, LetFlow and AG, respectively. For medium long flows with size between 100KB and 1MB, TLB achieves the similar FCT performance as LetFlow and AG. For long flows larger than 1MB, TLB also obtains the best performance.

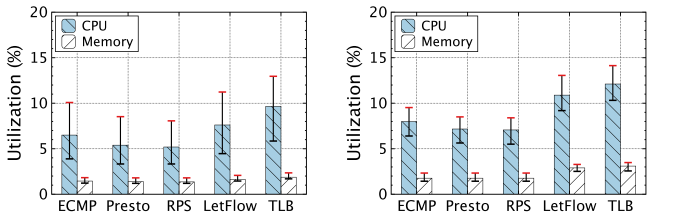
Appendix B
Performances Under P4 Software Switch

To show the feasibility of TLB in software, we conduct experiments based on P4 software switch and Mininet, which is a high fidelity network emulator for software-defined networks based on Linux kernel [20], [23]. We implement the packet processing pipeline of TLB with P4_{16} 1.0. We use Mininet 2.3.0 to create a leaf-spine topology with 10 equal-cost paths between the leaf and spine switches on a server, which is a Dell PRECISION TOWER 5820 desktop



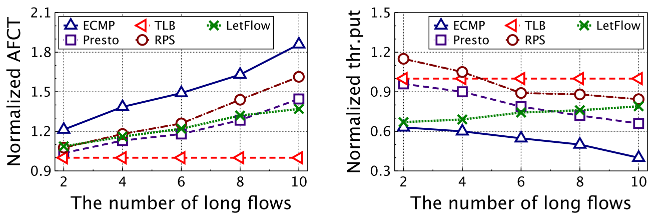
(a) AFCT of short flows (b) Throughput of long flows

Fig. 27. Varying number of short flows.



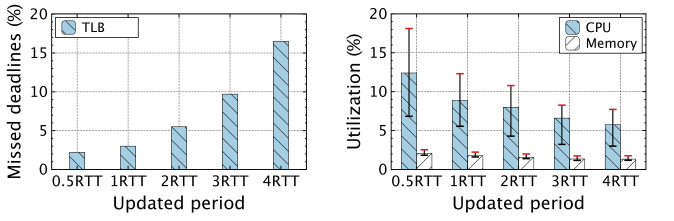
(a) Varying number of short flows (b) Varying number of long flows

Fig. 29. Overhead of the leaf switch.



(a) AFCT of short flows (b) Throughput of long flows

Fig. 28. Varying number of long flows.



(a) Missed deadline of short flows (b) CPU and memory utilization

Fig. 30. Varying updated period of the shortest queue.

with 10 cores Intel Xeon W-2255 CPU, 64GB memory, Mellanox ConnectX-5 100GbE NICs and Ubuntu 16.04. We set the link bandwidth to 200Mbps due to the limited switching ability on a single machine and delay to 1ms [43]. The behavioral model version 2 (BMv2) is installed as the software programmable switch with the buffer size of 256 packets. The default numbers of short flows less than 100KB and long flows larger than 5MB are 300 and 6, respectively. The overall traffic obeys the heavy-tailed distribution in web search workload as illustrated in [14] and the deadlines of short flows are randomly distributed between [2s, 6s]. We still use the 25^{th} percentile deadline (i.e., 3s) to calculate the switching threshold of queue length for rerouting long flows. The updating interval of the switching granularity of long flows and the flowlet timeout are set to 15ms.

We normalize the results of ECMP, RPS, Presto and LetFlow to that of TLB. Fig. 27 (a) shows that TLB reduces the AFCT of short flows by 19%-42%, 12-32%, 11-25%, 15-20% with the increasing number of short flows over ECMP, RPS, Presto and LetFlow, respectively. As shown in Fig. 27 (b), TLB improves the average throughput of long flows by 49%-95%, 8-48%, 28-36% over ECMP, Presto and LetFlow, respectively. With more long flows, Fig. 28 (a) and (b) show that TLB achieves better performance compared with the other schemes by adaptively adjusting the switching granularity of long flows. Due to the inability to flexibly reroute flows, ECMP and LetFlow suffer from the long-tailed delay and low link utilization. For RPS and Presto, the packet reordering degrades their performances especially under the heavy workload.

To evaluate the system overhead of TLB under the above two scenarios, we measure the average CPU and memory utilization introduced by TLB on P4 software switch as shown in Fig. 29. Specifically, we firstly measure the average utilizations of CPU and memory every base RTT, and then calculate the average, maximum and minimum values of average utilizations of all base RTTs. For ECMP, RPS and Presto, due to their simple operations at switches, the CPU utilization is lower than LetFlow and TLB. The short flows provide only about 20% of datacenter traffic, which incur a small amount of extra CPU usage. Therefore, TLB does not incur excessive CPU overhead and brings about negligible memory utilization compared with other schemes.

To forward the packets of small flows on the path with the minimum delay, TLB needs to timely update the shortest queue length. On the one hand, it is the most accurate to update the shortest queue length as soon as it changes, but it will increase the CPU overhead. On the other hand, increasing the update interval will reduce CPU overhead, but TLB cannot obtain the accurate shortest queue length in time. A proper value of the updated time interval will make a good compromise between high accuracy and low overhead.

Next, we conduct new simulation tests to show the deadline missing ratio of short flows and CPU utilization with varying updated period of the shortest queue. We use a fixed period as $0.5\times$RTT, $1\times$RTT, $2\times$RTT, $3\times$RTT and $4\times$RTT. In this simulation, traffic and experimental settings are the default values. As shown in Fig. 30 (a) and (b), with the increase of the updated period, although the overheads of CPU and memory are decreased, TLB is difficult to get the shortest queue length in time and the queueing delay of short flows increases, resulting in larger missed deadline ratio. Therefore, we use base RTT as the updated period of the shortest queue length to obtain good tradeoff between high accuracy and low overhead. In the future work, we will continue to improve TLB by optimizing the updated period.

However, as the compiled target for the data plane behavior, BMv2 software switch still degrades TLB performance compared with P4 commodity hardware switch. The reason behind this is as follows. Firstly, the data is read and written from a virtual Ethernet interface in the software switch, limiting the packet dequeueing rate from the shared buffer to the egress pipeline [44]. Secondly, BMv2 is used as a tool for developing, testing and debugging P4 data and control planes, the throughput and latency performances of

BMv2 are significantly worse than that of a production-grade switch. Moreover, since Mininet is originally designed to emulate a software-defined network, the network traffic is strictly controlled by a centralized controller, which makes Mininet not a perfect tool for high-speed network emulation [43].

REFERENCES

[1] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.

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

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

[4] G. Michelogiannakis, K. Z. Ibrahim, J. Shalf, J. J. Wilke, S. Knight, and J. P. Kenny, "APHiD: Hierarchical task placement to enable a tapered fat tree topology for lower power and cost in HPC networks," in *Proc. IEEE/ACM CCGrid*, May 2017, pp. 228–237.

[5] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, document RFC 2992, Internet Engineering Task Force, 2000.

[6] 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*, Apr. 2013, pp. 2130–2138.

[7] J. Hu, J. Huang, W. Lv, W. Li, J. Wang, and T. He, "TLB: Traffic-aware load balancing with adaptive granularity in data center networks," in *Proc. ACM ICPP*, 2019, p. a18.

[8] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *Proc. ACM SIGCOMM*, 2015, pp. 465–478.

[9] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. USENIX NSDI*, 2017, pp. 407–420.

[10] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, 2014.

[11] L. Zhou, C.-H. Chou, L. N. Bhuyan, K. K. Ramakrishnan, and D. Wong, "Joint server and network energy saving in data centers for latency-sensitive applications," in *Proc. IEEE IPDPS*, May 2018, pp. 700–709.

[12] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 63–74.

[13] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.

[14] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC*, 2010, pp. 267–280.

[15] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in cloud data stores via adaptive replica selection," in *Proc. USENIX NSDI*, 2015, pp. 513–527.

[16] S. Bak, H. Menon, S. White, M. Diener, and L. Kale, "Multi-level load balancing with an integrated runtime approach," in *Proc. IEEE/ACM CCGRID*, May 2018, pp. 31–40.

[17] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (D2TCP)," in *Proc. ACM SIGCOMM*, 2012, pp. 115–126.

[18] K. Zheng and X. Wang, "Dynamic control of flow completion time for power efficiency of data center networks," in *Proc. IEEE ICDCS*, Jun. 2017, pp. 340–350.

[19] Y. Li *et al.*, "HPCC: High precision congestion control," in *Proc. ACM SIGCOMM*, 2019, pp. 44–58.

[20] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1581–1589.

[21] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[22] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 174–187.

[23] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He, "CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2338–2353, Dec. 2019.

[24] M. Alizadeh *et al.*, "PFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 435–446.

[25] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*. Hoboken, NJ, USA: Wiley, 2008.

[26] 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, pp. 2324–2352, 3rd Quart., 2018.

[27] J. Xia *et al.*, "Rethinking transport layer design for distributed machine learning," in *Proc. ACM APNet*, Aug. 2019, pp. 22–28.

[28] J. Huang, Y. Huang, J. Wang, and T. He, "Packet slicing for highly concurrent TCPs in data center networks with COTS switches," in *Proc. IEEE ICNP*, Nov. 2015, pp. 22–31.

[29] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Practical information-agnostic flow scheduling in data center networks," in *Proc. USENIX NSDI*, 2015, pp. 455–468.

[30] Q. Huang *et al.*, "SketchVisor: Robust network measurement for software packet processing," in *Proc. ACM SIGCOMM*, 2017, pp. 113–126.

[31] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. ACM SIGCOMM*, 2018, pp. 561–575.

[32] Q. Huang, P. P. C. Lee, and Y. Bao, "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference," in *Proc. ACM SIGCOMM*, 2018, pp. 576–590.

[33] Z. Liu *et al.*, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM SIGCOMM*, 2019, pp. 334–350.

[34] The P4.org Architecture Working Group. (2020). *P4$_{16}$ Portable Switch Architecture (PSA)*. [Online]. Available: https://p4.org/p4-spec/docs/PSA.html

[35] X. Jin *et al.*, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. ACM SOSP*, 2017, pp. 121–136.

[36] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *Proc. USENIX NSDI*, 2018, pp. 1–16.

[37] Z. Zhao, X. Shi, X. Yin, Z. Wang, and Q. Li, "HashFlow for better flow record collection," in *Proc. IEEE ICDCS*, Jul. 2019, pp. 1416–1425.

[38] T. Qu, R. Joshi, M. C. Chan, B. Leong, D. Guo, and Z. Liu, "SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks," in *Proc. IEEE ICNP*, Oct. 2019, pp. 1–12.

[39] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.

[40] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. ACM SIGCOMM*, 2017, pp. 15–28.

[41] J. Liu, J. Huang, W. Li, and J. Wang, "AG: Adaptive switching granularity for load balancing with asymmetric topology in data center network," in *Proc. IEEE ICNP*, Oct. 2019, pp. 1–11.

[42] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, Aug. 2015, pp. 123–137.

[43] S. Liu, H. Xu, L. Liu, W. Bai, K. Chen, and Z. Cai, "RepNet: Cutting latency with flow replication in data center networks," *IEEE Trans. Services Comput.*, vol. 14, no. 1, pp. 248–261, Feb. 2021, doi: 10.1109/TSC.2018.2793250.

[44] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, "DC.P4: Programming the forwarding plane of a data-center switch," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, pp. 1–8.

[45] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010, pp. 19–34.

[46] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. ACM CoNEXT*, 2011, pp. 1–12.

[47] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proc. ACM CoNEXT*, 2014, pp. 149–160.

[48] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 3670–3682.

[49] W. Wang, Y. Sun, K. Zheng, M. A. Kaafar, D. Li, and Z. Li, "Freeway: Adaptively isolating the elephant and mice flows on different transmission paths," in *Proc. ICNP*, Oct. 2014, pp. 362–367.

[50] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3670–3682, Dec. 2017.

[51] F. Fan, B. Hu, and K. L. Yeung, "Routing in black box: Modularized load balancing for multipath data center networks," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 1639–1647.

[52] J. Zheng, Q. Zheng, X. Gao, and G. Chen, "Dynamic load balancing in hybrid switching data center networks with converters," in *Proc. ACM ICPP*, Aug. 2019, pp. 1–10.

[53] X. Gao, L. Kong, W. Li, W. Liang, Y. Chen, and G. Chen, "Traffic load balancing schemes for devolved controllers in mega data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 572–585, Feb. 2017.

[54] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet control for datacenter networks," in *Proc. USENIX NSDI*, 2017, pp. 421–435.

[55] N. Katta *et al.*, "Clove: Congestion-aware load balancing at the virtual edge," in *Proc. ACM CoNEXT*, 2017, pp. 323–335.

[56] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong, "Expeditus: Congestion-aware load balancing in clos data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3175–3188, Oct. 2017.

[57] N. Katta, M. Hiray, C. Kimz, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proc. ACM SOSR*, 2016, p. 10.

[58] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 133–145, Jan. 2019.

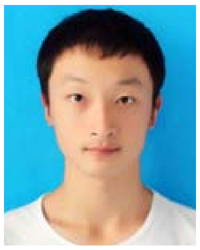
[59] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, Oct. 2011.

[60] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

Jinbin Hu received the B.E. and M.E. degrees from Beijing Jiao Tong University, China, in 2008 and 2011, respectively, and the Ph.D. degree from the School of Computer Science and Engineering, Central South University, China, in 2020. Her current research interests are in the area of data center networks.

Jiawei Huang (Member, IEEE) received the bachelor's degree from the School of Computer Science, Hunan University, in 1999, and the master's and Ph.D. degrees from the School of Computer Science and Engineering, Central South University, China, in 2004 and 2008, respectively. He is currently a Professor with the School of Computer Science and Engineering, Central South University. His research interests include performance modeling, analysis, and optimization for wireless networks and data center networks.

Wenjun Lyu received the B.E. and M.E. degrees in computer science and technology from Central South University in 2017 and 2020, respectively. He is currently pursuing the Ph.D. degree with Rutgers University. His research interests include data center networks and cyber-physical systems.



Weihe Li is currently pursuing the M.S. degree with the School of Computer Science and Engineering, Central South University, China. His research interests are video streaming and data center networks.

Zhaoyi Li received the B.S. degree from Central South University, China, in 2019, where he is currently pursuing the M.S. degree with the School of Computer Science and Engineering. His research interests are in the area of data center networks.

Wenchao Jiang received the B.S. and M.S. degrees from Shanghai Jiao Tong University, China, and the Ph.D. degree from the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. He is currently an Assistant Professor with the Pillar of Information Systems Technology and Design, Singapore University of Technology and Design, Singapore. His research interests include the Internet of Things, wireless networks, sensor networks, and mobile systems.

Jianxin Wang (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer engineering from Central South University, Changsha, Hunan, China, in 1992, 1996, and 2001, respectively. He is currently the Chair and a Professor with the School of Computer Science and Engineering, Central South University. His current research interests include algorithm analysis and optimization, parameraized algorithm, bioinformatics, and computer networks.

Tian He (Fellow, IEEE) received the Ph.D. degree from the University of Virginia, Charlottesville, in 2004, under the supervision of Prof. John A. Stankovic. He is currently a Professor with the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. He is the author or coauthor of more than 200 articles in journals and conferences with over 20 000 citations (H-Index 59). His publications have been selected as graduate-level course materials by over 50 universities in the USA and other countries. His research interests include wireless sensor networks, cyber-physical systems, intelligent transportation systems, real-time embedded systems, and distributed systems. His research was supported by the U.S. National Science Foundation, IBM, Microsoft, and other agencies. He has received a number of research awards in the area of networking, including five best paper awards. He was a recipient of the NSF CAREER Award 2009 and the McKnight Land-Grant Professorship. He served a few program chair positions in international conferences and on many program committees. He currently serves as an Editorial Board Member for six international journals, including IEEE TRANSACTIONS ON COMPUTER. He is an ACM Fellow.