Computer Networks 157 (2019) 133–145

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Adaptive load balancing based on accurate congestion feedback for asymmetric topologies

Qingyu Shi[a], Fang Wang[a,b,*], Dan Feng[a], Weibin Xie[a]

[a] Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China, China
[b] Shenzhen Huazhong University of Science and Technology Research Institute, China

ARTICLE INFO

Article history:
Received 5 November 2018
Revised 1 April 2019
Accepted 15 April 2019
Available online 23 April 2019

Keywords:
Datacenter network
Load balancing
Congestion feedback
Low latency

ABSTRACT

Datacenter load balancing schemes exist to facilitate parallel data transmission with multiple paths under various uncertainties such as traffic dynamics and topology asymmetries. Taking deployment challenges into account, several optimized schemes (e.g. CLOVE, Hermes) to ECMP balance load at end hosts. However, inaccurate congestion feedback exists in these solutions. They either detect congestion through Explicit Congestion Notification (ECN) and coarse-grained Round-Trip Time (RTT) measurements or are congestion-oblivious. These congestion feedbacks are not sufficient enough to indicate the accurate congestion status under asymmetry. And when rerouting events occur, outdated ACKs carrying congestion feedback of other paths can improperly influence the current sending rate. After our observations and analyses, these inaccurate congestion feedback can degrade performance.

Therefore, we explore how to address above problems while ensuring good adaptation to existing switch hardware and network protocol stack. We propose ALB, an adaptive load balancing mechanism based on accurate congestion feedback running at end hosts, which is resilient to asymmetry. ALB leverages a latency-based congestion detection to precisely reroute new flowlets to the paths with lighter load, and an ACK correction method to avoid inaccurate flow rate adjustment. In large-scale simulations, ALB achieves up to 13% and 48% better average flow completion time (FCT) than CONGA and CLOVE-ECN under asymmetry, respectively. And compared with other schemes ALB improves the average and the 99th percentile FCTs for small flows under high bursty traffic by 43–174% and 75–129%. Under the situation of dynamic network changes, ALB also provides competitive overall performance and maintains stable performance for small flows.

1. Introduction

Datacenter networks typically adopt multi-rooted topologies, such as fat tree and leaf spine, to provide high bisection bandwidth. The multipath existing in these topologies provides several alternative routing paths between any two end hosts which are connected by different switches. Balancing load in multiple paths to fully utilize the network resource can improve throughput and reduce latency for datacenter applications. But various uncertainties, such as traffic dynamics and topology asymmetries, pose great challenges for designing efficient load balancing schemes. Production datacenters present a network environment with dynamic traffics [1], where applications that are sensitive to bandwidth (e.g. MapReduce) and sensitive to flow completion time (e.g. Memcached) exist. And asymmetry is common in datacenter networks [2] because of adding racks, heterogenous network devices, cutting links and switch malfunctions [3,4]. Efficient load balancing mechanisms usually adapt to above uncertainties, which should accurately detect path conditions and distribute traffic among multipaths based on path conditions.

However, Equal Cost Multiple Path (ECMP) forwarding [5], as the standard strategy used today for load balancing in datacenter networks, performs poorly. It randomly assigns flows to different paths permanently according to a hash function using certain tuples from the packet header. Because it accounts for neither path conditions nor flow size, it can waste over 50% of the bisection bandwidth [6].

Therefore, prior solutions (e.g. CONGA [7], CLOVE-ECN [8], FlowBender [9], Hermes [2]) have made a great deal of effort to improve performance, but they still have some drawbacks. Some distributed load balancing schemes (e.g. CONGA, HULA [10], LetFlow [11]) residing in custom switches are hard to deploy in

* Corresponding author.
E-mail addresses: qingyushi@hust.edu.cn (Q. Shi), wangfang@hust.edu.cn (F. Wang).

https://doi.org/10.1016/j.comnet.2019.04.006
1389-1286/© 2019 Elsevier B.V. All rights reserved.

general datacenter networks, although they achieve significant improvement for throughput and latency. Centralized solutions (e.g. Hedera [6]) globally schedule large flows by collecting network information in a controller. But they have long scheduling intervals, which are not adaptive to the traffic volatility of datacenter networks and harmful for small flows.

The last category of solutions (e.g. CLOVE-ECN, Hermes) are deployed at network edges (e.g. hypervisor) or end hosts to keep practical. Some of them are designed to be congestion-aware. However, they depend too much on the rough congestion feedback (e.g. ECN and coarse-grained RTT measurements) to sense congestion. CLOVE-ECN learns congestion along network paths from ECN signals and uses a weighted round-robin (WRR) algorithm to dynamically route flowlets [12] on multiple paths. Hermes also exploits ECN signals and coarse-grained RTT measurements to decide the flow path at the host side. RTT measurements lump latencies in both directions along the network path. In order to use RTTs to capture congestion of the forward path, prior mechanisms (e.g. Hermes, TIMELY [13]) classify pure ACK packets in the reverse path into the higher priority queue. Inaccurate ECN signals and coarse-grained RTT measurements can degrade the performance gains in asymmetric topologies though they schedule flows using excellent algorithms. The ECN-based congestion detection cannot accurately characterize the degree of congestion among multiple paths in an asymmetric network due to its inherent oversimplified feedback. The coarse-grained RTT measurement introduces end host network stack delay, which can be believable if and only if a small enough RTT is discovered. Thus it cannot accurately represent the degree of path congestion. Furthermore, ECN is a passive and delayed mechanism for informing congestion level in multiple paths. Thus it can hardly help achieve timely load balancing.

Actually the end-to-end latency effectively indicates whether the path has been congested. Fortunately with the rapid growth of cloud computing and network functions virtualization (NFV), the advances in widely used NIC hardware and efficient packet IO frameworks (e.g. DPDK [14]) have made the measurement of end-to-end latency possible with microsecond accuracy. Latency-based implicit feedback is accurate enough to reveal path congestion [15]. DPDK now supports all major CPU architectures and NICs from multiple vendors (e.g. Intel, Emulex, Mellanox and Cisco). A tuned DPDK solution (e.g. TRex [16]) only introduces 5–10μs overhead [17]. With the help of DPDK, the end-to-end latency can be measured with sufficient precision to sense path conditions. Several latency-based congestion control protocols for datacenter networks have emerged (e.g. TIMELY, DX [15]). But latency-based implicit feedback has hardly been applied to load balancing schemes.

Moreover, current load balancing solutions also create new inaccurate congestion feedback in transport protocols. The end hosts in present datacenters commonly adopt ECN-based transport protocols (e.g. DCTCP [18]). Congestion control algorithms of transport protocols usually adjust the rate (window) of a flow based on the congestion state of the current path. When rerouting events happen, outdated ACKs with no ECE mark of the other path may improperly increase the sending rate (window), while the ones with an ECE mark will mistakenly decrease the sending rate. This problem hinders the utilization of link bandwidth especially under asymmetric topologies, because network asymmetry creates different network conditions more easily among different routing paths.

According to above observation, we find inaccurate congestion feedback causes inaccurate detection to path conditions and incorrect flow rate adjustment. This problem is bound to affect the performance of load balancing (Sections 2.2 and 2.3). Therefore, we ask the following question: can we design a congestion-aware load balancing scheme that can achieve accurate congestion feedback and keep practical? Finally we present ALB to answer this question, which is an adaptive load balancing solution implemented at end hosts. ALB employs accurate latency-based measurement to detect network path congestion. The latency-based congestion detection enables ALB to accurately reroute flows. And an ACK correction method is used to avoid blindly adjusting the flow rate at source hosts.

We make following contributions in this paper:

- We analyze that inaccurate congestion feedback can degrade performance under asymmetry in load balancing.
- We present ALB, an adaptive load balancing mechanism based on accurate congestion feedback running at end hosts, which is resilient to asymmetry and readily-deloyable with commodity switches in large-scale datacenters.
- In large-scale simulations we show that ALB achieves up to 13% and 48% better flow completion time than CONGA and CLOVE-ECN under asymmetry, respectively. Under the impact of dynamic network changes, ALB improves the overall average FCT by 5–42% compared to CLOVE-ECN. And ALB always keeps the best and stable performance for small flows under high bursty traffic. Compared with Hermes, ALB requires no complicated parameter settings and provides competitive performance.

Some preliminary results of this paper were published in the Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS, 2018) [19]. In this paper, we describe our motivation with more detailed theoretical and empirical analyses, improve the latency-based congestion detection mechanism (Section 3.2) and extend the evaluations for dynamic datacenter network changes (Section 4.2.2).

The rest of this paper is organized as follows. In next section, we introduce the background and motivation of designing ALB. Then we detail ALB in Section 3. And we evaluate ALB and show the superiority of ALB compared to other solutions in Section 4. Finally we briefly introduce the related work in Section 5 and summarize our work in Section 6.

2. Background and motivation

In this section, we describe network asymmetries and traffic dynamics pose challenges to load balancing and inaccurate congestion feedback exacerbates the performance loss. These problems motivate us to design ALB.

2.1. Network asymmetries and traffic dynamics

Network asymmetry is common for modern datacenter networks in practice, where different paths between one or more source/destination pairs have different amounts of available bandwidth. Link failures and heterogeneous network equipment (e.g. different link speeds and different number of forwarding ports) can be common and cause asymmetry [2,7,20,21]. Fig. 1 shows an example of asymmetric topology caused by link failure, where the link capacity between Spine 1 and Leaf 1 is reduced. As shown in previous studies [3], traffic dynamics in production datacenters are also common. Path congestion can quickly occur because of burst traffic and can disappear as some flows finish. In the following we will analyze that asymmetry exacerbates the inaccuracy of congestion feedback, which causes performance degradation particularly for dynamic traffic workloads.

2.2. Inaccurate congestion detection

ECN-based network protocols have been widely used in datacenter networks. The congestion detection in many load balancing schemes deployed at end hosts uses ECN signals as congestion

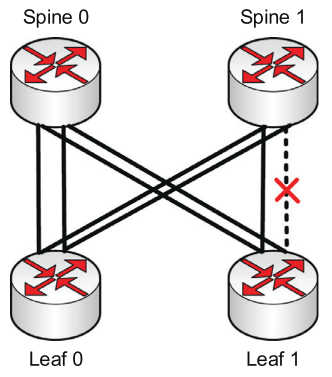


Fig. 1. An example of asymmetric topology caused by link failure. All links run at 10 Gbps.

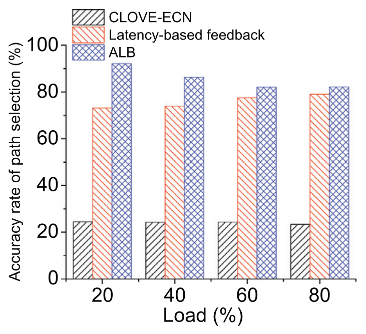


Fig. 2. The accuracy rates of different schemes.

feedback (e.g. FlowBender, CLOVE-ECN). FlowBender keeps track of the fraction of ECN-marked ACKs out of the total ACK packets every RTT at source hosts. If this fraction is larger than a certain threshold for any flow, this means that this flow is congested and should be rerouted. CLOVE-ECN also similarly judges the degree of congestion based on path weights at source hosts, where path weights are calculated based on piggybacked ECN signals. However, the simple ECN-based feedback cannot accurately show the exact level of congestion on multiple paths under asymmetry. Because an ECN mark only indicates congestion at a single switch and cannot describe the congestion extent of multiple switches. With experiments and theoretical analysis, [15] has observed that ECN as the congestion feedback is low accuracy and coarse granularity for congestion control. Although outstanding, load balancing algorithms can be influenced by inaccurate congestion detection on multiple paths and cause unsatisfactory performance.

In addition, several latency-based congestion control protocols (e.g. DX [15], TIMELY [13]) have been designed for datacenter networks to improve network utilization, which have shown that one-way delay provides richer and faster information about the state of network switches than ECN. We can also employ latency metrics to detect path congestion in load balancing. However, latency feedback measured at end hosts is still outdated for load balancing though it provides more accurate congestion metrics compared to ECN. This should have a non-negligible impact on performance. Therefore, how to provide more timely congestion feedback is essential.

To quantify the impact of congestion detection on load balancing, we run an ECN-based scheme CLOVE-ECN, a latency-based feedback mechanism and ALB based on the web-search workload [18] under a 4 × 4 leaf-spine topology with 10 Gbps links and 32 servers in NS3 [22]. The latency-based feedback mechanism collects RTTs on multiple paths from every ACKs to detect path congestion, where the RTTs only contain delays in the network and pure ACKs are classified into the higher priority queue than data packets. We make the topology asymmetric through reducing the capacity from 10 Gbps to 2 Gbps for 20% of randomly selected leaf-to-spine links. Because one-way delay directly signal the extent of end-to-end congestion [15], we measure one-way delay to represent path congestion level. We send probes on each routing path to timely update one-way delay at every source host. After referring to the same simulation environment configuration of Hermes and our many experimental verifications, we find that a probing interval of 100–500µs brings good visibility when the one hop delay is configured with 80µs. In this way, every time a flow is rerouted, we can know if it is switching to the least congested path. If it does switch to the least congested path, we record it as a correct path selection. As shown in Fig. 2, the accuracy rate of CLOVE-ECN at all different load levels only accounts for approximately 24%, while the latency-based feedback mechanism can increase the accuracy to up to 79%. This shows that the path congestion detection based on latency provides greater accuracy than the one based on ECN signals in load balancing. And although latency metrics measured at end hosts are outdated, they track closely the correct path selection in our experiment. Besides, ALB shows the highest accuracy at every load level in Fig. 2. This is because ALB employs a novel latency-based congestion detection (Section 3.2) mechanism to achieve faster latency feedback. In a word, we should adopt accurate congestion detection for load balancing schemes deployed at end hosts to improve network utilization.

2.3. Inaccurate rate adjustment

Generally, congestion control algorithms in source hosts adjust the flow rate according to congestion feedback piggybacked in ACKs. The feedback should represent the congestion state of the current path. But after rerouting events happen between asymmetric paths, outdated ACKs of the old path owning the different link capacity with the current path will improperly affect the flow rate. Under the situation with ECN-based transport protocols, the outdated ACK with no ECE mark of the other path may improperly increase the sending rate (window), while the one with an ECE mark will mistakenly decrease the sending rate. The Fig. 3 shows a simple example, where flow A is switched from path 1 to path 3 because of the decision of load balancing. The outdated ACKs of flow A denotes the ACKs which reply the data sent at path 1 before rerouting, while the new ACKs denotes the ACKs which reply the data sent at path 3 after rerouting. Hence, the congestion states of different paths are mixed together for the flow A. The sender will mistakenly use ACKs not belonging to the current state to adjust the sending rate. This phenomenon is also called congestion mismatch [2], which causes the inaccurate rate adjustment problem under asymmetry. In order to reveal this problem, we count the proportion of outdated ACKs of all flows under the experiment for CLOVE-ECN in Section 2.2. We find the proportion is always around 5% on different load level. We next reveal the performance degradation caused by inaccurate rate adjustment with empirical study.

To quantify the impairment of inaccurate rate adjustment in load balancing, we implement a per-packet scheme DRB [23] under an asymmetric topology in NS3. We use a simple 2 × 2 leaf-spine topology and a heterogenous network with 2 and 10 Gbps paths shown in Fig. 4a. And a reordering buffer is implemented in the DRB to mask packet reordering. DCTCP is used as the default transport protocol like most test scenarios in prior load balancing schemes. As shown in Fig. 4b, flow A only achieves around 3.7 Gbps overall throughput. This is because when rerouting happens the congestion feedback in ACKs belonging to the path with 2 Gbps bandwidth (the left path) constrains the congestion window, causing the throughput loss in the path with 10 Gbps bandwidth

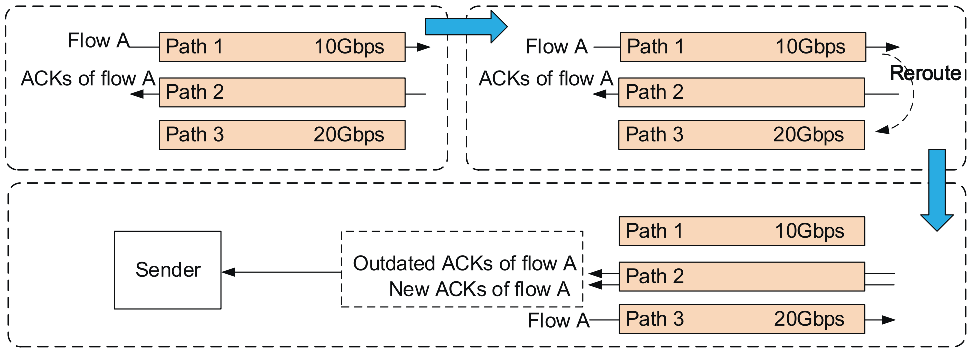


Fig. 3. An example of inaccurate rate adjustment.

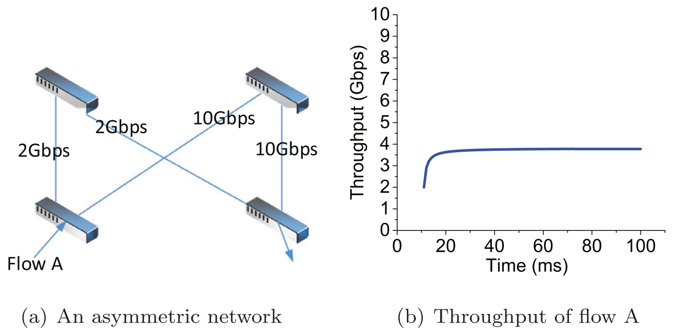


Fig. 4. Inaccurate rate adjustment causes severe throughput loss.

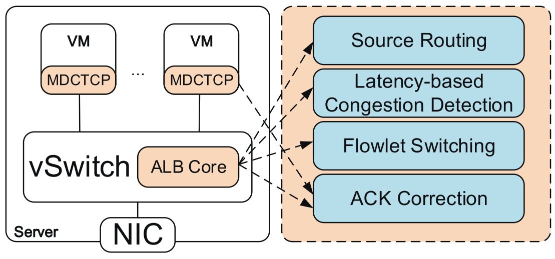


Fig. 5. The design of ALB.

(the right path). Similarly, the congestion feedback from the right path can cause incorrect rate adjustment in the left path, which may lead to sudden congestion because the left path cannot immediately absorb such a burst. In other load balancing schemes vigorous rerouting between two paths also can result in inaccurate rate adjustment as long as the available bandwidth of this two paths are different. Some solutions (e.g. Presto [24], WCMP [20]) distribute traffic proportionally to path capacity at the system initialization, but this static setting still cannot avoid performance loss caused by the inaccurate rate adjustment. The root cause is the chaos of congestion feedback because of load balancing.

To handle issues caused by the inaccurate congestion feedback, an ideal solution should avoid inaccurate congestion detection and inaccurate rate adjustment in asymmetric topologies. Existing load balancing schemes either requires custom switches for load balancing in the network or suffers performance degradation from inaccurate congestion detection. And current schemes do not solve inaccurate rate adjustment. This motivates us to design ALB, an adaptive congestion-aware load balancing solution under asymmetry, which achieves accurate congestion feedback at end hosts with commodity switches. In large-scale simulations we show that ALB provides competitive performance with those schemes requiring custom switches (e.g. LetFlow, CONGA).

3. Design

3.1. Overview

We present ALB's framework in Fig. 5. ALB contains two modules, which are MDCTCP and ALB core. We design MDCTCP by slightly modifying DCTCP. MDCTCP is an ECN-based network protocol. And other three functions, namely source routing, latency-based congestion detection and accurate flowlet switching, work in the ALB core. The ALB core is implemented in software in hypervisor vSwitch (e.g. Open vSwitch), which is common for current multi-tenant datacenters to manager numerous virtual machines.

The source routing is used for path discovery. The latency-based congestion detection provides accurate one-way delay measurement for each path to detect path congestion. Moreover, ALB divides each flow into flowlets in a fine-grained way while leveraging latency-based congestion detection to select the least congested path for new flowlets. And the ACK correction function with the cooperation of MDCTCP and ALB core is used to correct inaccurate rate adjustment.

3.2. The detailed design of ALB

Source Routing. Because we need to reroute flows in different routing paths at the source side, the source routing is designed for routing path discovery. For this purpose, we implement an existing traceroute mechanism [8] in the source vSwitch to detect multiple routing paths, which has been used in several other schemes (e.g. CLOVE). As we know, commodity datacenter switches inherently implement ECMP, and network overlays have been recently widely adopted in multi-tenant datacenter networks. By sending probes with varying source ports, which are encapsulated by overlay network protocols in static ECMP-based datacenter networks, the source routing in ALB can find a subset of source ports that lead to distinct paths. Besides, each probe contains multiple packets with the same transport protocol source port but with the TTL value incremented. We can obtain the list of IP addresses of switch interfaces along that path to distinguish different routing paths. ALB modifies the source port of tunnel encapsulation (e.g. using Stateless Transport Tunneling (STT) protocol) to control transmission path of every flow. The probes are sent periodically to adapt to the changes in the network topology. Because the virtual switch in an overlay network typically generates Bidirectional Forwarding Detection (BFD) probes to all other overlay destinations in every few hundred milliseconds with negligible overhead, if the frequency of sending probes is about a few hundred milliseconds like BFD, the overhead in our design should be negligible too [8]. The probes in ALB are used to find multiple paths in system initialization,

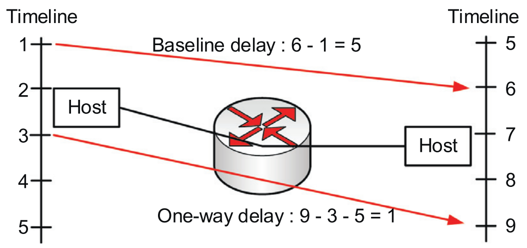


Fig. 6. An example of measuring one-way delay without time synchronization.

and the probe frequency only determines the reaction time to a change in network topology.

Latency-based congestion detection. The latency-based congestion detection enables ALB to accurately reroute flowlets to the least congested paths. ALB measures the RTT and one-way delay for every flow to judge the degree of path congestion. Compared with ECN-based congestion detection, our algorithm can characterize the degree of path congestion more accurately through the end-to-end delay detection, and feedback congestion metrics more quickly by leveraging normal connections of other flows. It is worth mentioning that to the best of our knowledge, though several congestion control protocols (e.g. DX, TIMELY) have adopted one-way delay as congestion signal, ALB first uses it in load balancing. The former adjusts the flow sending rate based on changes of the one-way delay on the current routing path, while the latter uses one-way delays of multiple routing paths to detect path conditions for flow rerouting.

Firstly, we introduce the method to accurately measure latency metrics. Because RTT contains the delay on the reverse path, which introduces noise for determining congestion level on the forward path, we use one-way delay to judge path congestion. The main problem of acquiring accurate one-way delay is the clock synchronization between the source and destination host. Although Precision Time Protocol (PTP) can support clock synchronization with sub-microseconds, it still requires periodic synchronization to compensate clock drifts. The periodic synchronization may affect the delay measurement in microsecond level. And PTP also requires hardware support and possibly switch support. Therefore, ALB leverages a technique to measure one-way delay under the clock without synchronization, which is used in some latency-based congestion control protocols (e.g. DX [15]). The principle is that we calculate the real one-way queuing delay by subtracting the baseline delay, which is the one-way delay measured without clock synchronization when the link is idle (no queueing in switches). The baseline delay can be measured by picking the minimum among enough samples from the current one-way delay. For example, as the Fig. 6 shows, the baseline delay is measured to be 5 s when link is idle. This value includes a clock difference, propagation delay (no queuing) and other times due to path states. When the link is not idle, a sample value is measured to be 6 ($9 - 3 = 6$) seconds because of queuing delay. Thus, we get 1 ($6 - 5 = 1$) second one-way queuing delay by subtracting 5 from 6.

Next we detail how to exchange latency metrics for path selection. As Fig. 7 shows, we record NIC time $t1$–$t4$ into the option fields of TCP header at the DPDK-based device driver. We update the baseline delay in every few RTTs to avoid clock drifts problems. In addition to calculating one-way delay and measuring RTT, ALB uses a feedback loop between the source and destination vSwitch to populate remote metrics in the Latency-To-Leaf Table at each vSwitch. And we assign IDs to different transmission paths with the help of source routing. Referring to Fig. 7 for an example, we describe the process as the following five steps:

1. ALB selects transmission path with the smallest one-way delay in Latency-To-Leaf Table for every new flowlet, and write the path ID into the fields of tunnel encapsulation. The source server writes NIC time into $t1$ right before the TX.
2. Right after the RX, the destination server records NIC time into $t2$. The destination vSwitch stores a mapping from the expected ACK of the packet to t1, t2 and the path ID in the tunnel encapsulation when it forwards the packet to the destination host. The destination vSwitch updates the one-way delay ($t2 - t1 - baseline$) to Latency-From-Leaf Table.
3. When the returned ACK goes through the destination vSwitch, $t1$ and $t2$ are filled into the ACK header according to the timestamp mapping, while the path ID is piggybacked in the tunnel encapsulation. And right before the TX we fill $t3$ with the NIC time in destination server. One metric from Latency-From-Leaf Table is inserted in the tunnel encapsulation.
4. When the ACK arrives at the source server, right after the RX $t4$ is filled into the ACK header.
5. Finally, when the ACK goes through the source vSwitch, we can calculate the new RTT value RTT_{new} as $t4 - t1 - (t3 - t2)$ and update this value of path 2 only when $RTT_{new} \le RTT_{old}$ (RTT_{old} represents the old RTT value) on this path. Then we update one metric in Latency-To-Leaf Table with the feedback metric in tunnel encapsulation (for path 1).

It is important to emphasize that we describe a simplified process of exchanging latency metrics above. Actually the feedback metrics are piggybacked in every packet and one-way delays are updated after each packet arrives. Besides, we make load balancing decisions for each flowlet and ACKs. We have improved the algorithm of the first version in [19] in two ways. On the one hand, we reselect the least congested path for ACKs that encounter a tuned flowlet timeout to accelerate the return of ACKs. On the other hand, in addition to data packets, ACKs are also used to calculate one-way delays.

We show the packet header format in an example in Fig. 8 for a clear description. The data packet carries a path ID (2), which denotes its current path, and one feedback metric (3, 500) from local Latency-From-Leaf Table. When it arrives destination vSwitch, the one-way delay ($t2 - t1 - baseline$) for path ID 2 and the feedback metric are recorded in Latency-From-Leaf Table and Latency-To-Leaf Table, respectively. Similarly, when an ACK arrives destination vSwitch, it should carry one path ID (2), which indicates the transmission path of data previously acked, one feedback metric (1, 700) and its transmission path ID (4). The feedback metric are recorded in Latency-To-Leaf Table and we calculate a new RTT value from $t4 - t1 - (t3 - t2)$. The one-way delay ($t4 - t3 - baseline$) for path ID 4 is recorded in Latency-From-Leaf Table. As this process shows, ALB timestamps packets in the option fields of TCP header so as to calculate accurate RTT and one-way delay. Two path IDs (e.g. the path ID 2 and the path ID 4 residing in data packets and ACKs in Fig. 8) and one feedback metric are encoded in the fields of tunnel encapsulation Stateless Transport Tunneling (STT) context. One byte is used to encode one path ID, while 5 bytes are used to encode one feedback metric which contains one path ID and a one-way delay requiring 4 bytes. Therefore, we need a total of 7 bytes to encode two path IDs and one feedback metric in the STT context, which containing 64 bits provides sufficient fields for our design. We use a round robin fashion to convey feedback metrics in Latency-From-Leaf Table and the metrics whose values have been updated since last feedback will be preferentially chosen.

Through above steps we maintain the corresponding one-way delay mapping for each path. But over time the baseline one-way delay may be influenced by the clock drift between two hosts. We update the baseline delay according to the change of RTTs to solve

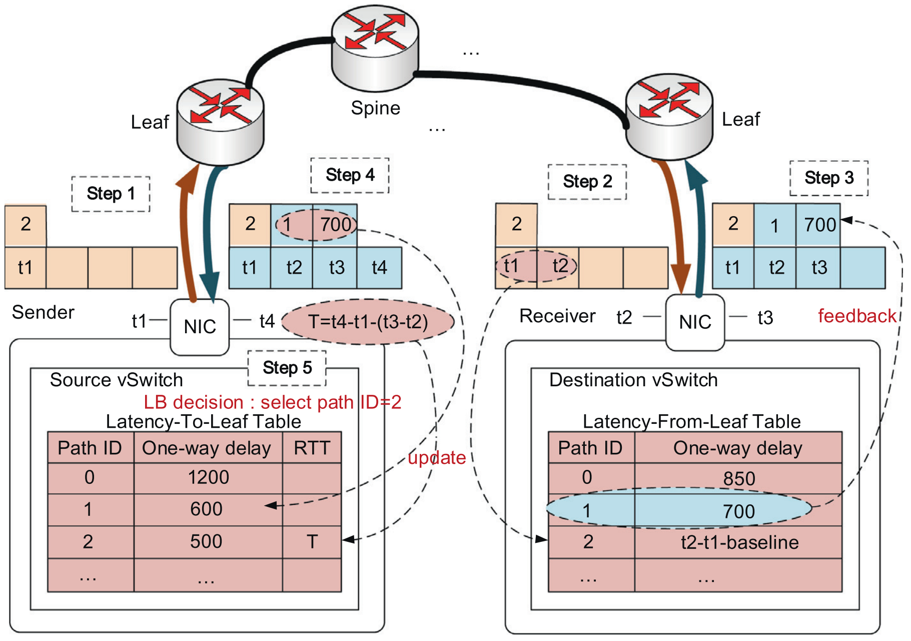


Fig. 7. The framework of latency-based congestion detection.

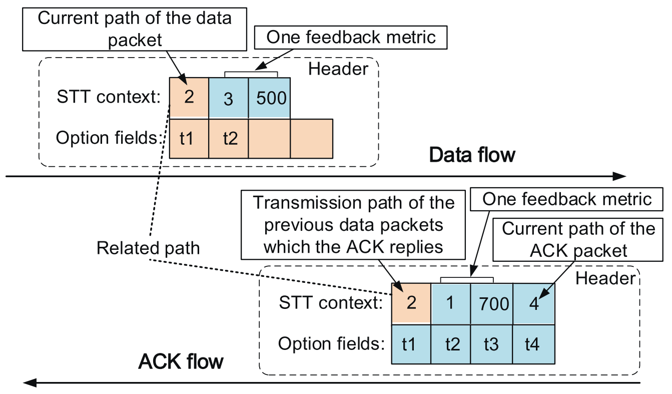

Current path of the data packet
One feedback metric
Header
STT context:
Option fields:
Data flow
Transmission path of the previous data packets which the ACK replies
Current path of the ACK packet
Related path
ACK flow

Fig. 8. The Packet header format in latency-based congestion detection.

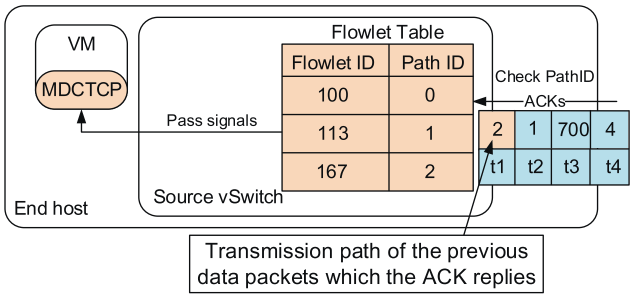


Fig. 9. An example of ACK correction.

the problem. Each time the RTT in mapping table is updated with a smaller or the same value, we need to remeasure the baseline one-way delay on this path according to previous studies [15].

Accurate flowlet switching. Flowlet switching is a widely-used and fine-grained load balancing strategy, which splits a flow into many flowlets to route over multiple paths without causing much packet reordering. Accurate flowlet switching here means that every time we choose a path for a new flowlet, we always choose the least congested path according to latency-based congestion detection. Different from prior schemes relying on ECN-based congestion detection, when encountering new flowlets ALB always selects the least congested routing path which has the smallest one-way delay referring to the Latency-To-Leaf Table in source vSwitch (Fig. 7). This enables ALB to make more accurate rerouting decisions for load balancing, which has been shown in Fig. 2.

ACK correction. Because congestion control protocols always adjust flow rate based on the congestion feedback of the current path, rerouting events can cause a mismatch between the sending rate and the state of the new path. This is the inaccurate rate adjustment problem as we described in Section 2.3. ACK correction is used to handle this problem.

ACK correction needs to pass the signal of state mismatch to MDCTCP to adjust flow rate. Firstly, the returned ACK should carry a path ID in the encapsulation header to indicate that previously transmitted data of the flow is transmitted on this path (referring to the path ID 2 in Fig. 8). As the example in Fig. 9 shows, the path ID is 2, which means that the congestion information carried by the ACK belongs to the path with ID 2. Then, when the returned ACK reaches the source vSwitch, ALB calculates the flowlet ID of this flow according to the fields of protocol header. Afterwards ALB gets the current mapped path ID according to the flowlet ID in the Flowlet Table, which is used to record the current transmission paths of flows. ALB compares whether the mapped path ID is equal to 2. If they are not equal, which means that the flow has been switched to a new mapped path and the ACK is outdated, ALB modifies a reserved bit in the TCP header to 1, otherwise sets the bit to 0. In this way, the reserved bit can indicate whether the congestion feedback belongs to the current state. ALB can pass this signal to MDCTCP through the reserved bit. We call it Path Change Notification and *PCN* for short.

MDCTCP is a congestion control protocol that is slightly modified based on DCTCP. In MDCTCP the *PCN* in TCP header joins the flow rate control. Because zeroing the *PCN* means that this is a matched feedback, MDCTCP only adjusts the congestion window and threshold when *PCN* is 0, otherwise we maintain these values unchanged. Other features of DCTCP are left unchanged. While DCTCP always uses an estimate α to resize it's window size,

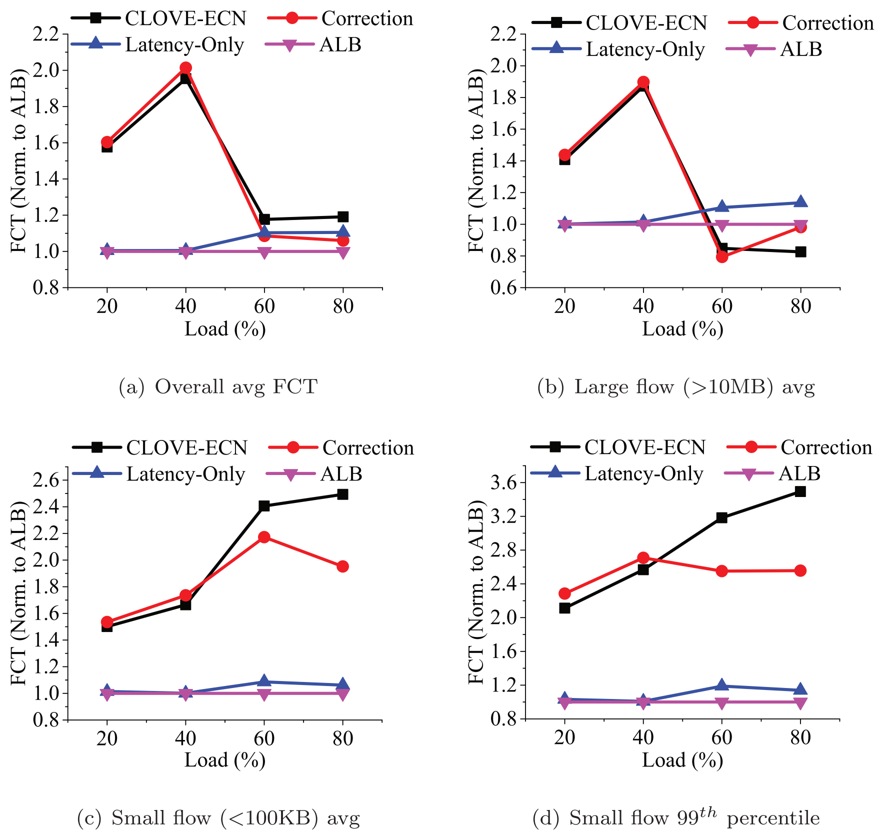


Fig. 10. FCT of different methods (normalized to ALB).

MDCTCP adds a conditional statement:

$$cwnd = \begin{cases} cwnd \times (1 - \alpha/2), & if\ PCN\ is\ 0 \\ cwnd, & if\ PCN\ is\ 1 \end{cases} \tag{1}$$

Thus, we eliminate the influence of congestion feedback that does not belong to the current state to avoid the inaccurate rate adjustment. When the load is not heavy, there are not many flow rerouting events so the performance of MDCTCP may get close to DCTCP. When the load becomes heavy, more flow rerouting events occur. The value α becomes high and even causes the congestion window to be mistakenly cut in half in DCTCP (e.g. $\alpha = 1$) due to outdated ACKs. Therefore the flow sending rate can be limited incorrectly. Because of solving this problem MDCTCP can significantly improve performance at heavy load. The evaluation result from Fig. 10 in the next section also validates our analysis, in which ACK correction works better at high loads.

4. Evaluation

We evaluate ALB via the discrete-event network simulator NS3 [22]. Our evaluation seeks to answer the following questions:

How does each design component contributes to performance? (Section 4.1) ALB implements a novel latency-based congestion detection and an ACK correction method at end hosts. We evaluate the benefits brought by these two methods separately. Results show that both of them contribute to around 10% overall performance improvements under heavy loads.

How does ALB perform under traffic dynamics and asymmetries? (Section 4.2.1) For the web-search workload, ALB is within 1–45% of CONGA and achieves 15–48% better average FCTs than CLOVE-ECN. Besides, ALB obtains up to 4× better FCTs for small flows at high loads compared to LetFlow and CLOVE-ECN. For the data-mining workload, ALB outperforms CONGA and CLOVE-ECN by up to 13% and 20%, respectively. All in all, compared with schemes requiring custom switches (e.g. CONGA, LetFlow), ALB shows very competitive performance. And ALB greatly outperforms the schemes which are implemented at end hosts (e.g. CLOVE-ECN).

How effective is ALB under dynamic network changes? (Section 4.2.2) We simulate the experiment under dynamic network changes, and results show that ALB can effectively adapt to the network dynamics, which improves the overall average FCT by 5–42% compared to CLOVE-ECN. And ALB keeps the best and stable performance for small flows under high bursty traffic.

4.1. Functional verification

As we analyze in Sections 2.2 and 2.3, inaccurate congestion feedback (including inaccurate congestion detection and inaccurate rate adjustment) can degrade performance in load balancing. In order to demonstrate ALB's ability to solve these problems, we run a trace-driven simulation for functional verification, which is based on the web-search workload in a 8 × 8 leaf-spine topology with 10 Gbps links and 128 servers in NS3. We make the topology asymmetric through reducing the capacity from 10 Gbps to 2 Gbps for 20% of randomly selected leaf-to-spine links.

ALB employs ACK correction to avoid inaccurate rate adjustment, while using latency-based congestion detection to improve the accuracy of congestion detection on multiple paths. In order to reveal performance benefits of these two optimized methods, we evaluate each of them separately compared with CLOVE-ECN. In Fig. 10 Latency-Only indicates a solution where only latency-based congestion detection is implemented, while Correction indicates a modified CLOVE-ECN where ACK correction is implemented. We use flow completion time (FCT) as the performance metric (Note that we normalize the flow completion time to ALB to better visualize the results). From Fig. 10, we can see that Latency-Only always performs better than CLOVE-ECN, while Correction works better at high loads ($\geq$ 60% load). This because compared to CLOVE-ECN, Latency-Only can always find a less congested path for a new flowlet more accurately. When the load gets more aggravated, more paths become congested, therefore the performance improvement of Latency-Only decreases. Besides, because more flowlets are created at high loads, more path switching events occur. Therefore, at high loads ACK correction solves more events of inaccurate rate adjustment so as to obtain more performance gains. By combining these two methods, ALB achieves 15–48% better overall average FCT and 50–149% better 99th-percentile FCT for small flows than CLOVE-ECN in Fig. 10. In other words, the inaccuracies do exist, and the combination of latency-based congestion detection and ACK correction can provide significant performance improvements.

4.2. Performance comparison

We evaluate ALB in this subsection and compare its performance with other representative solutions in large-scale simulations.

Workloads: We use two widely-used realistic workloads including web-search [18] and data-mining [25], which are obtained from production datacenters. These two workloads are both heavy-tailed and most flows generated are small, but the small fraction of large flows contributes to a great portion of total bytes. Particularly, the data-mining workload is more skewed with 95% of all data bytes belonging to around 3.6% of flows that are larger than 35MB, which makes it more challenging for load balancing [7].

Metrics. Similar to previous work, we use flow completion time (FCT) as the primary performance metric. In addition to the overall average FCT, we also take the FCT for small flows ($<$100KB) and large flows ($>$10MB) into consideration for better understanding of performance. And the 99th percentile FCT for small flows is also an important performance metric.

Topology. We build a 8 $\times$ 8 leaf-spine topology with 10Gbps links and 128 servers with NS3. Therefore we simulate a 2:1 oversubscription at the leaf level to meet the typical deployment of current datacenter networks [7].

Methodology: In order to show the performance gain from solving the inaccuracy problems, besides ECMP we compare ALB with the following state-of-the-art solutions using DCTCP [18] as the default transport protocol:

- **Presto.** We implement an idealized variant of Presto to provide best-case performance for Presto, where we employ per-packet load balancing and a reordering buffer to put all packets of every flow in order.
- **LetFlow.** LetFlow picks paths at random for flowlets in custom switches.
- **CONGA.** CONGA employs global utilization-aware flowlet switching in custom switches.
- **CLOVE-ECN.** CLOVE-ECN leverages ECN-based feedback to route flowlets at end hosts. Because CLOVE-INT is shown to be outperformed by CONGA [8], we do not simulate CLOVE-INT.

Note: We do not compare all previous solutions. Because Hermes [2] does not have better performance than CONGA in all cases when there is no switch failure and it introduces a large number of parameter settings that make it hard to achieve optimal performance for us, we have not simulated it. Additionally we do not compare against MPTCP [26] because of performance instability and worse performance gains compared with CONGA in many scenarios.

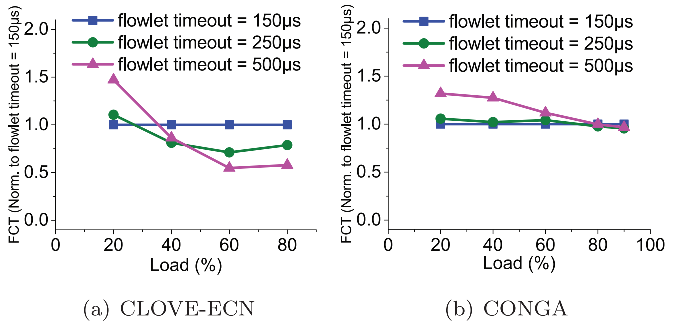


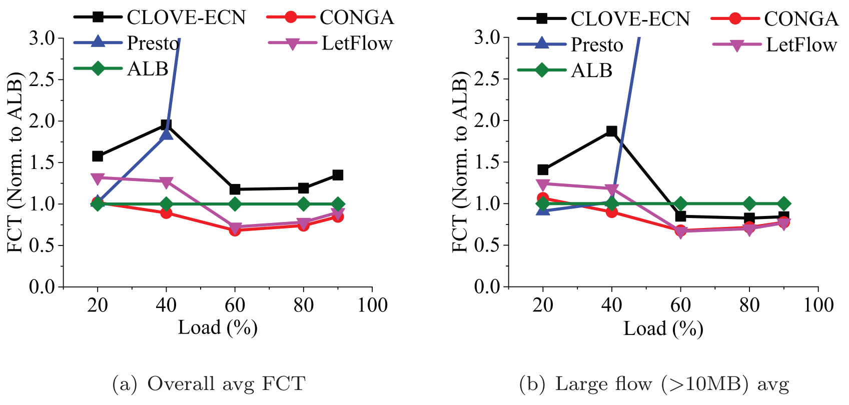
Fig. 11. Overall avg FCT in different flowlet timeout (normalized to flowlet timeout = 150μs).

We should adopt a consistent and proper flowlet timeout value in all schemes for fairness. But setting it too small can risk reordering issue, while setting it too large can reduce flowlet opportunities. Because DCTCP is less bursty than TCP, the default 500μs flowlet timeout value in CONGA is too big [2]. Therefore we try several different flowlet timeout values and adopt the most appropriate one (250μs) in our simulations. Fig. 11 shows the average FCT of CLOVE-ECN and CONGA in different flowlet timeout values under the web-search workload. At low loads the 150μs flowlet timeout achieves best performance. This is because reducing the flowlet timeout creates more rerouting opportunities. However when load increases, too small flowlet timeout obtains degraded performance. CLOVE-ECN with 150μs flowlet timeout achieves the worst performance at 40–80% loads and setting the flowlet timeout from 150μs to 250μs improves FCT in CONGA at 80–90% loads. This because as load increases packets may arrive out of order due to congestion under asymmetry and more congestion mismatch events occur. Therefore, after considering the performance of different load levels, we use a modest flowlet timeout (250μs) in our simulation.

4.2.1. Impact of asymmetric topology

To compare ALB with above schemes under an asymmetric topology, we reduce the link capacity from 10 Gbps to 2 Gbps for 20% of randomly selected leaf-to-spine links. We normalize the FCT to ALB in order to better visualize the results.

Under the web-search workload: As shown in Fig. 12, CONGA performs the best performance in most cases. ALB and LetFlow achieve similar performance. Because the web-search workload is more bursty and creates a large number of flowlets, in-network schemes (e.g. CONGA and LetFlow), which require modification of switches, can balance load more faster than other solutions deployed at end hosts. But at 20–40% load, ALB performs 21–24% better than LetFlow. This is because LetFlow is oblivious to congestion and the load is not heavy. ALB achieves accurate congestion detection so as to obtain better performance than LetFlow at light loads. When load gets heavy, LetFlow can quickly converge even without good visibility to congestion. Compared to CLOVE-ECN, ALB always improves the overall average FCT by 15–48%. This is because ALB, as the same as CLOVE-ECN deployed at end hosts, achieves more accurate congestion feedback than CLOVE-ECN. Besides, previous work [2] show that Hermes only achieves similar performance with CLOVE-ECN under the web-search workload. Moreover, as shown in Fig. 12c and d, the average and the 99th percentile FCTs for small flows grow dramatically for all of schemes except ALB. And as Fig. 13a shows, ALB reduces the average FCTs for small flows by a few milliseconds or more than ten milliseconds, which greatly improves the quality of delay-sensitive small request services in web search systems. This is because under high loads many small flows are broken



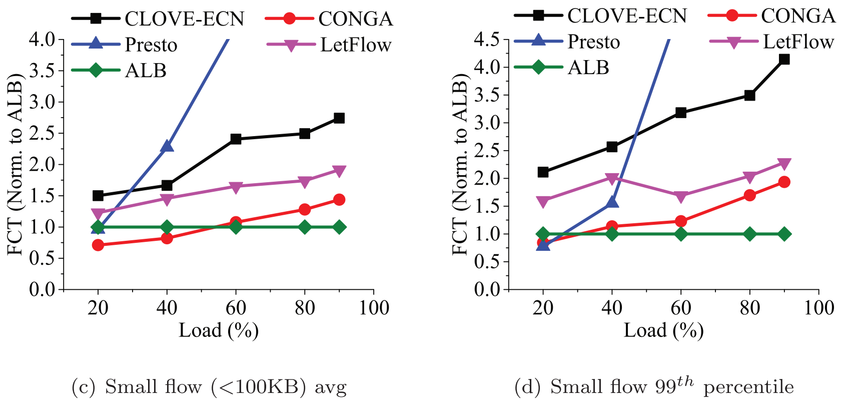


Fig. 12. FCT for the web-search workload in the asymmetric topology (normalized to ALB).

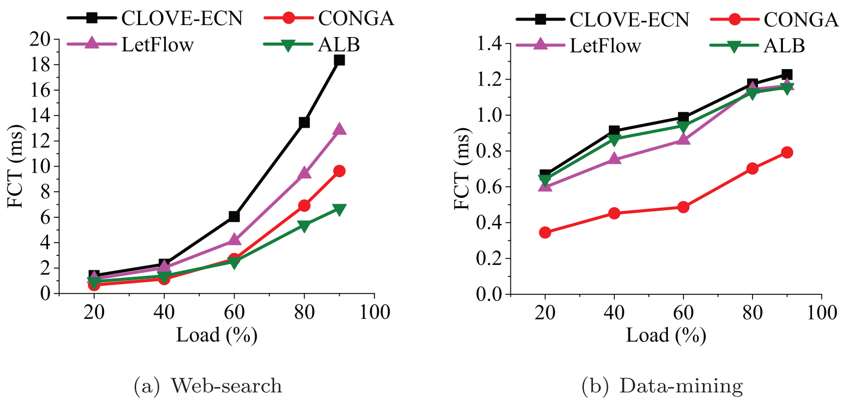
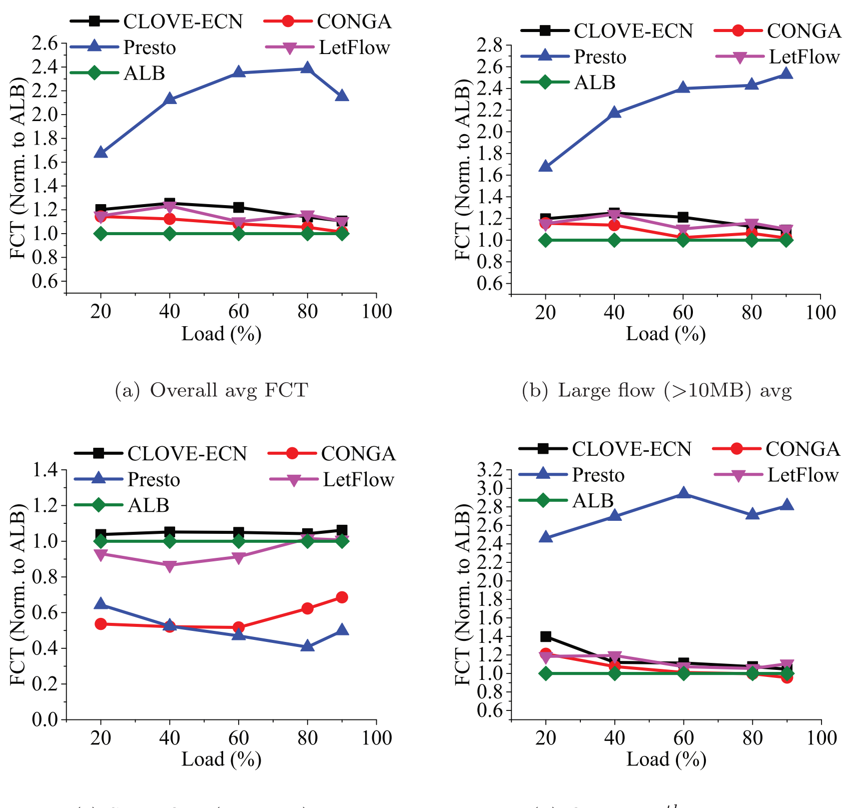


Fig. 13. Small flow (< 100KB) avg in the asymmetric case.

into several flowlets, where prior solutions are seriously affected by inaccurate rate adjustment (Section 2.3). In comparison, ALB can alleviate the performance degradation due to its accurate rate adjustment. Thus, compare with other schemes, at 90% load ALB improves the average and the 99th percentile FCTs for small flows by 43–174% and 75–129%, respectively.

Under the data-mining workload: As shown in Fig. 14, ALB achieves 2–13% better performance than CONGA for the overall average FCT. Note that the data-mining workload contains more large flows and has a much bigger inter-flow arrival time. This leads to much fewer flowlets than under the web-search workload. Under the circumstances, the visibility of network congestion becomes especially important for load balancing. As we know, LetFlow is oblivious to path conditions and CLOVE-ECN does not achieve accurate congestion detection. Thus, with accurate congestion detection ALB achieves 9–18% and 10–20% better performance than LetFlow and CLOVE-ECN, respectively. And because CONGA implemented in switches cannot handle the inaccurate rate adjustment problem, ALB can outperform CONGA slightly. Besides, ALB provides competitive performance on average FCTs for small flows and overall tail latency. As shown in Fig. 14c, ALB and LetFlow achieve similar performance on average FCTs for small flows. Actually as the Fig. 13b shows, these schemes only have differences in microsecond granularity for average FCTs for small flows. As shown in Fig. 14d, in most cases ALB achieves the minimum overall 99th percentile FCTs.

For Presto, we take into account the path asymmetry by using static weights (based on the topology) to make load balancing decisions [11]. However, Presto does not achieve comparable performance to ALB as Figs. 12a and 14 a show. This is because Presto cannot solve inaccurate rate adjustment problem. When load gets heavy under asymmetry, the congestion window in Presto



(a) Overall avg FCT (b) Large flow (>10MB) avg

(c) Small flow (<100KB) avg (d) Overall 99th percentile

Fig. 14. FCT for the data-mining workload in the asymmetric topology (normalized to ALB).

is constrained by the most congested path and the flow rate is adjusted in a chaotic way. This causes high FCTs under asymmetry.

4.2.2. Impact of dynamic network changes

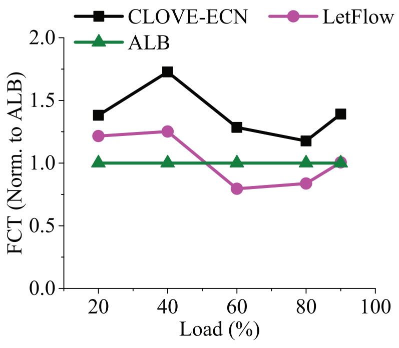
As link failures and some network device updates can occur in datacenters at anytime, the situation of dynamic network changes should be seriously considered. In order to demonstrate the ability of ALB to adapt to topology dynamics, we perform above experiments under dynamic network changes, where we reduce the link capacity from 10 Gbps to 2 Gbps for 20% of randomly selected leaf-to-spine links at a certain time when the experiment is still running. We consider LetFlow, CLOVE-ECN and ALB in this experiment. Since the link capacity is required by CONGA's DRE algorithm [7] to estimate network utilization on each link, it is difficult for CONGA to estimate link utilization with sufficient accuracy when the link capacity changes in the system running in our simulation. So we do not simulate CONGA under this situation. And LetFlow has shown very close performance to CONGA in our experiments(in Figs. 12 and14) and previous work [11] also have verified this. Presto is not considered here because of its high FCTs under asymmetry.

As shown in Fig. 15, LetFlow achieves the best average FCT in most load levels under the web-search workload. But in order to distribute traffic efficiently, it requires custom switches, which are not widely supported by commodity switches. ALB, which is implementable on existing networks, achieves average FCTs that are only slightly higher than Letflow at high loads: within 19% at 80% load. Compared to CLOVE-ECN, ALB always improves the overall average FCT by 15–42%. This is because ALB leverages accurate congestion feedback to timely react to dynamic network conditions. Moreover, under bursty arrivals of new flows the average and the 99th percentile FCTs for small flows grow dramatically for LetFlow and CLOVE-ECN as shown in Fig. 15c and d, while ALB achieves the best and stable performance at all load levels because it alleviates inaccurate rate adjustment problem. ALB performs increasingly better (18–45% and 26–52% respectively for average and the 99th percentile FCTs) for small flows compared to Letflow as the load increases. Moreover, because the traffic under the data-mining workload is less bursty, the favorable visibility to congestion and opportune reaction to asymmetries are more crucial for load balancing. Under this circumstance Letflow achieves suboptimal performance with congestion-oblivious rerouting and blind flow rate adjustment. As we can see from Fig. 16a, ALB improves the overall average FCT by 5–10% compared to LetFlow that requires advanced hardware, while ALB performs 5–15% better than CLOVE-ECN. We can find that even under dynamic network changes and the extremely bursty workload, ALB can timely adapt to these dynamics to balance load efficiently. When the network becomes asymmetric under a relatively smooth workload, ALB always maintains excellent performance and surpasses several previous solutions.

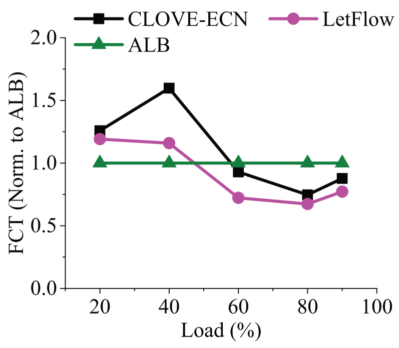
4.3. Complexity of ALB

From the perspective of system architecture, ALB and CLOVE-ECN are very similar because they are both mainly implemented in the hypervisor at end hosts. Thus, in this subsection we discuss the complexity of ALB by comparing with CLOVE-ECN. ALB still differs from CLOVE-ECN in that ALB modifies the congestion control algorithm in VMs, but this take very little computation overhead for VMs since we only add several conditional statements based on DCTCP using one extra reserved bit in the TCP header. Therefore, we focus on the memory and processing overhead of ALB in the hypervisor.

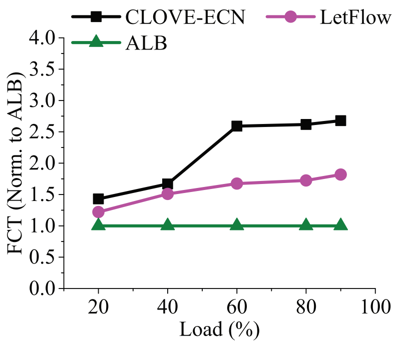
Memory overhead: ALB needs to keep congestion state at every hypervisor for multiple paths, which is similar to the requirements of CLOVE-ECN. Firstly they both need to use a Flowlet Table at each



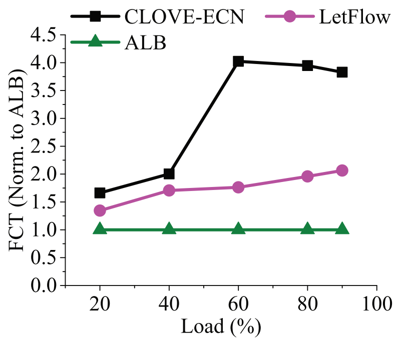
(a) Overall avg FCT



(b) Large flow (>10MB) avg

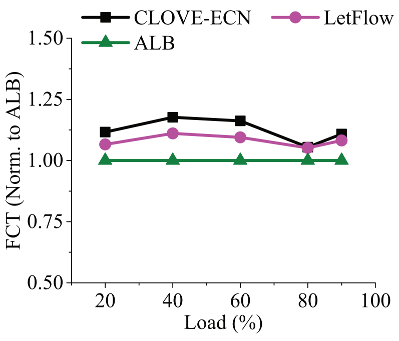


(c) Small flow (<100KB) avg

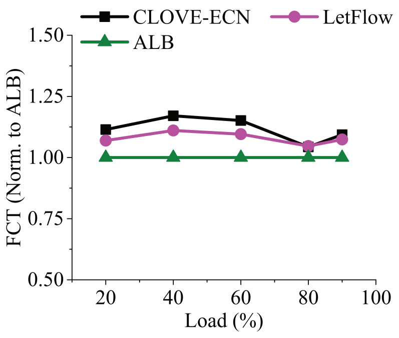


(d) Small flow 99th percentile

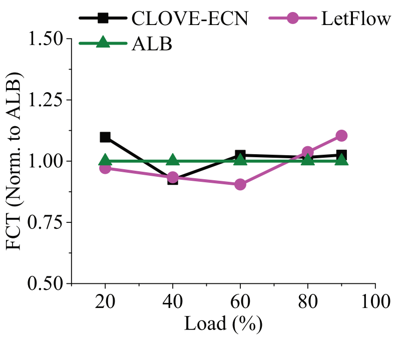
Fig. 15. FCT for the web-search workload under dynamic network changes (normalized to ALB).



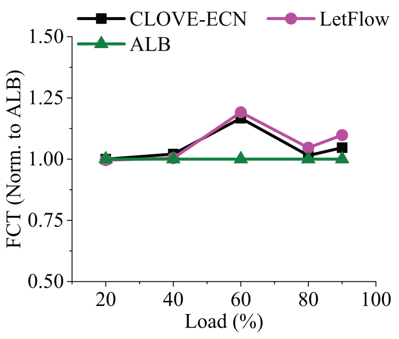
(a) Overall avg FCT



(b) Large flow (>10MB) avg



(c) Small flow (<100KB) avg



(d) Small flow 99th percentile

Fig. 16. FCT for the data-mining workload under dynamic network changes (normalized to ALB).

Table 1
Memory usage of CLOVE-ECN and ALB in the simulation.

Scheme	Load level (%)			
	20	40	60	80
CLOVE-ECN	1.51%	2.16%	3.99%	6.30%
ALB	1.44%	2.11%	4.06%	6.70%

hypervisor to track flowlets. Maintaining such a table has proven to be at low cost [7]. Moreover, CLOVE-ECN uses a path weight table at each source hypervisor to distinguish different congestion degrees of paths. The weights associated with the distinct paths are continuously adapted based on the congestion feedback obtained from ECN messages. But ALB uses latency metrics to sense path conditions. ALB employs two tables, which are Latency-From-Leaf Table and Latency-To-Leaf Table, to keep latency metrics at each hypervisor. Obviously, ALB needs to use more memory space. We use 1 byte to store path ID and 4 bytes to store the one-way delay or RTT in ALB. Therefore, storing one metric for one path takes 5 bytes and 9 bytes in Latency-From-Leaf Table and Latency-To-Leaf Table, respectively. Assuming a typical 8×8 leaf-spine topology in datacenters, we at least need to store congestion state for 56 paths in every hypervisor, which means that we need 784 bytes of extra memory ($(5+9) \times 56 = 784$). Even at a 16×16 leaf-spine topology, ALB only uses 3360 bytes of extra memory. This shows that the memory overhead for storing congestion state in ALB is negligible for servers in modern datacenters. Finally, we use simulations to support the analysis. We run CLOVE-ECN and ALB at a same server based on the web-search workload under a 8×8 leaf-spine topology with 10Gbps links and 128 servers in NS3 to measure their memory usage separately. As Table 1 shows, compared to CLOVE-ECN, ALB consumes the close amount of memory on a sever with a total memory of 32GB. Although the memory management in NS3 is different from that in real system, this can at least explain that ALB does not cause too much memory overhead in maintaining per-path congestion information.

Processing overhead: ALB updates RTTs and one-way delays based on timestamp of packets in every hypervisor to maintain per-path congestion state, while CLOVE-ECN updates path weights based on ECN signals. The CPU overhead brought by these two methods does not make much difference since they are both simple operations for extracting values and updating local records. ALB can leverage the same efficient locking mechanisms such as Read-Copy-Update (RCU) [27] locks as CLOVE-ECN to minimize blocking of threads when updating state, which is a mechanism already used for updating per-connection state in the Open vSwitch. The updates to these data structures for maintaining per-path congestion state happen in the datapath while maintaining the line rate throughput of at least 40Gbps per hypervisor [8]. Furthermore, ALB timestamps packets on top of DPDK, which is a mature technology and can be implemented on a wide variety of CPU architectures. Besides, ALB adopts a small number of header fields in the overlay transport protocol STT to convey congestion information. We know that network overlay has been widely used in modern datacenters to optimise device functions or reduce the complexity of the network devices. The CPU overhead caused by timestamping and using STT protocol should be normal to the system. Through comparison with CLOVE-ECN and analyse for other additional operations, the processing overhead introduced by ALB should be tolerable for servers in modern datacenters.

5. Related work

We briefly discuss related work that has informed and inspired our design.

Hedera [6], MicroTE [28] and FastPass [29] use a centralized scheduler to monitor global network state and schedules flows evenly in multiple paths. They cannot achieve timely reaction to latency-sensitive application requests and have difficulties handling traffic volatility.

Presto [24], DRB [23] and Flowbender [9] are per-flowcell/packet/flow based, congestion-oblivious load balancing solutions. They cannot effectively balance traffic without visibility of dynamic network conditions on multiple paths.

CONGA [7], HULA [10] and CLOVE-INT [8] leverage specialized switches or advanced programmable switches to achieve global congestion-aware switching. Though they obtain great performance improvements, the switch hardware requirements from these schemes pose difficulties for deployment in large-scale datacenters. And LetFlow also relies on new switch hardware for implementation, which splits flows into flowlets in the network without awareness of path congestion. The rerouting events at in-network switches cannot cooperate with the congestion control protocol at the sender, which leads to inaccurate rate adjustment as shown in Section 2.3. Due to accurate rate adjustment, ALB performs up to 13% better than CONGA under an asymmetric topology.

Taking network congestion awareness and deployment challenges into account, some optimized schemes (e.g. CLOVE-ECN [8], Hermes [2]) are proposed. CLOVE-ECN leverages per-flowlet weighted round robin at end hosts to route flowlets. And these path weights are calculated according to ECN signals residing in ACKs. Hermes uses packet as the minimum switchable granularity by exploits ECN signals and coarse-grained RTT measurements to sense congestion on multiple paths. And it requires many parameter settings, which are hard to tune, to effectively sense path state and make efficient load balancing decisions. This kind of previous schemes suffer from inaccurate congestion feedback, which causes performance degradation as we showed in Section 4.

Finally, all the aforementioned schemes either lack accurate congestion feedback, which results in great performance degradation as we discuss in Section 2.3, or require custom switch hardware for implementation to obtain performance gains.

6. Conclusion

We propose ALB, an adaptive load balancing mechanism based on accurate congestion feedback running at end hosts with commodity switches, which is resilient to asymmetry. ALB leverages the latency-based congestion detection to precisely route flowlets to lighter load paths, and an ACK correction method to avoid inaccurate flow rate adjustment. We evaluate ALB through large-scale simulations. Our results show that compared to schemes which require custom switch hardware for implementation, ALB can provide competitive performance.

Acknowledgments

This work is supported in part by NSFC No. 61772216, National Defense Preliminary Research Project (31511010202), the National High Technology Research and Development Program (863 Program) of China under Grant no. 2013AA013203; Hubei Province Technical Innovation Special Project (2017AAA129), Wuhan Application Basic Research Project (2017010201010103), Project of Shenzhen Technology Scheme (JCYJ20170307172248636), Fundamental Research Funds for the Central Universities. This work is also supported by NSFC No. NSFC61502190 and CERNET Innovation Project NGII20170120.

Conflict of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceeding of the ACM IMC, 2010, pp. 267–280, doi:10.1145/1879141.1879175.
[2] H. Zhang, J. Zhang, W. Bai, K. Chen, M. Chowdhury, Resilient datacenter load balancing in the wild, in: Proceeding of the ACM SIGCOMM, 2017, pp. 253–266, doi:10.1145/3098822.3098841.
[3] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: Proceeding of the ACM SIGCOMM, 2011, pp. 350–361, doi:10.1145/2018436.2018477.
[4] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, V. Kurien, Pingmesh: a large-scale system for data center network latency measurement and analysis, pp. 139–152.
[5] C. Hopps, Analysis of an equal-cost multi-path algorithm, RFC 2992, 2000, doi:10.17487/RFC2992.
[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: Proceeding of the USENIX NSDI, 2010, pp. 89–92.
[7] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V.T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese, CONGA: distributed congestion-aware load balancing for datacenters, in: Proceeding of the ACM SIGCOMM, 2014, pp. 503–514, doi:10.1145/2619239.2626316.
[8] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford, Clove: congestion-aware load balancing at the virtual edge, in: Proceeding of the ACM CoNEXT, 2017, pp. 323–335, doi:10.1145/3143361.3143401.
[9] A. Kabbani, B. Vamanan, J. Hasan, F. Duchene, Flowbender: flow-level adaptive routing for improved latency and throughput in datacenter networks, in: Proceeding of the ACM CoNEXT, 2014, pp. 149–160, doi:10.1145/2674005.2674985.
[10] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, HULA: scalable load balancing using programmable data planes, Proceeding of the ACM SOSR, 2016, doi:10.1145/2890955.2890968.
[11] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, T. Edsall, Let it flow: resilient asymmetric load balancing with flowlet switching, in: Proceeding of the USENIX NSDI, 2017, pp. 407–420.
[12] S. Kandula, D. Katabi, S. Sinha, A. Berger, Dynamic load balancing without packet reordering, ACM SIGCOMM Comput. Commun. Rev. 37 (2) (2007) 51–62, doi:10.1145/1232919.1232925.
[13] R. Mittal, V.T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, TIMELY: RTT-based congestion control for the datacenter, in: Proceeding of the ACM SIGCOMM, 2015, pp. 537–550, doi:10.1145/2785956.2787510.
[14] Intel dpdk. data plane development kit, (Accessed 9 October 2010. [Online]. Available: http://dpdk.org/).
[15] C. Lee, C. Park, K. Jang, S. Moon, D. Han, DX: Latency-based congestion control for datacenters, IEEE/ACM Trans. Netw. 25 (1) (2017) 335–348, doi:10.1109/TNET.2016.2587286.
[16] Cisco systems. TRex: Cisco's realistic traffic generator, (Accessed 9 October 2010. [Online]. Available: https://trex-tgn.cisco.com).
[17] M. Primorac, E. Bugnion, K. Argyraki, How to measure the killer microsecond, in: Proceeding of the ACM SIGCOMM Workshop on KBNets, 2017, pp. 37–42, doi:10.1145/3098583.3098590.
[18] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center tcp (dctcp), in: Proceeding of the ACM SIGCOMM, 2010, pp. 63–74, doi:10.1145/1851182.1851192.
[19] Q. Shi, F. Wang, D. Feng, W. Xie, ALB: adaptive load balancing based on accurate congestion feedback for asymmetric topologies, in: Proceeding of the IEEE IWQoS, 2018, pp. 1–6.
[20] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, A. Vahdat, WCMP: weighted cost multipathing for improved fairness in data centers, Proceeding of the ACM EuroSys, 2014, doi:10.1145/2592798.2592803.
[21] S. Sen, D. Shue, S. Ihm, M.J. Freedman, Scalable, optimal flow routing in datacenters via local link balancing, in: Proceeding of the ACM CoNEXT, 2013, pp. 151–162, doi:10.1145/2535372.2535397.
[22] Ns3, (Accessed October 9, 2010. [Online]. Available: https://www.nsnam.org/).
[23] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, D. Maltz, Per-packet load-balanced, low-latency routing for clos-based data center networks, in: Proceeding of the ACM CoNEXT, 2013, pp. 49–60, doi:10.1145/2535372.2535375.
[24] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akella, Presto: Edge-based load balancing for fast datacenter networks, in: Proceeding of the ACM SIGCOMM, 2015, pp. 465–478, doi:10.1145/2785956.2787507.
[25] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: Proceeding of the ACM SIGCOMM, 2009, pp. 51–62, doi:10.1145/1592568.1592576.
[26] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, in: Proceeding of the ACM SIGCOMM, 2011, pp. 266–277, doi:10.1145/2018436.2018467.
[27] P.E. McKenney, J. Walpole, What is RCU, Fundamentally?, (Accessed 10 January 10 2019. [Online]. Available: https://lwn.net/Articles/262464/).
[28] T. Benson, A. Anand, A. Akella, M. Zhang, MicroTE: fine grained traffic engineering for data centers, Proceeding of the ACM CoNEXT, 2011, doi:10.1145/2079296.2079304.
[29] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, H. Fugal, Fastpass: a centralized "zero-queue" datacenter network, in: Proceeding of the ACM SIGCOMM, 2014, pp. 307–318, doi:10.1145/2619239.2626309.

Qingyu Shi He received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently a Ph.D. student majoring in Computer Architecture in Wuhan National Laboratory for Optoelectronics (WNLO). His current research interests include software-defined networking and load balancing for datacenter networks. He has a publication in international conference: IWQoS.

Fang Wang She received her BE degree and Master degree in computer science in 1994, 1997, and Ph.D. degree in computer architecture in 2001 from Huazhong University of Science and Technology (HUST), China. She is a professor of computer science and engineering at HUST. Her interests include distribute file systems, parallel I/O storage systems and graph processing systems. She has more than 50 publications in major journals and international conferences, including FGCS, ACM TACO, SCIENCE CHINA Information Sciences, Chinese Journal of Computers and HiPC, ICDCS, HPDC, ICPP.

Dan Feng She received the BE, ME, and Ph.D. degrees in Computer Science and Technology in 1991, 1994, and 1997, respectively, from Huazhong University of Science and Technology (HUST), China. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including IEEE-TC, IEEE-TPDS, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She serves on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of IEEE and a member of ACM.

Weibin Xie He received the BE degree in Energy and Power Engineering from the China University of Mining and Technology (CUMT), Xuzhou, China, in 2011. He is currently a Ph.D. student majoring in Computer Architecture in HUST. His current research interests include Computer networks and protocols and distributed storage systems. He has several publications in major journals and international conferences, including CN and IWQoS.