ALB: Adaptive Load Balancing Based on Accurate Congestion Feedback for Asymmetric Topologies

Qingyu Shi[1], Fang Wang[1,2], Dan Feng[1], and Weibin Xie[1]

[1]*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*
[2]*Shenzhen Huazhong University of Science and Technology Research Institute*
Corresponding author: wangfang@hust.edu.cn

***Abstract*—In datacenter networks, multipath exists to facilitate parallel data transmission. Taking deployment challenges into account, some optimized alternatives (e.g. CLOVE, Hermes) to ECMP balance load at the virtual edge or hosts. However inaccuracies of congestion detection and reaction exist in these solutions. They either detect congestion through ECN and coarse-grained 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 on multiple paths, ACKs with congestion feedback of other paths can improperly influence the current sending rate.**

Therefore, we explore how to balance load by solving above inaccuracy problems while ensuring good adaptation to commodity switches and existing network protocols. We propose ALB, an adaptive load-balancing mechanism based on accurate congestion feedback running at end hosts, which is resilient to asymmetry. ALB leverage a latency-based congestion detection to precisely route flowlets to lighter load paths, and an ACK correction method to avoid inaccurate flow rate adjustment. In large-scale simulations ALB achieves up to 7% and 40% better flow completion time (FCT) than CONGA and CLOVE-ECN under asymmetry.

***Index Terms*—datacenter networks, load balancing, congestion feedback**

I. 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. Equal Cost Multiple Path (ECMP) forwarding [1], as the the standard strategy used today for load balancing in datacenters, randomly assigns flows to different paths according to a hash function using certain tuples from the packet header. But ECMP performs poorly because of hash collisions and the lack of adaptability to asymmetric topologies.

Despite causing traffic conflicts, ECMP is widely implemented because it is readily deployed with standard unmodified TCP/IP stacks and commodity datacenter switches. And under a symmetric network topology, if all flows are small, ECMP can provide near optimal transmission performance [2]. However, (1) on the one hand the datacenter presents a network environment with mixed traffics [3], where applications which are sensitive to bandwidth (e.g. MapReduce) and sensitive to flow completion time (e.g. Memcached) exist. (2) On the other hand, network asymmetry is common for modern datacenters in practice [4], where different paths between one or more source/destination pairs have different amounts of available bandwidth. Because the datacenter evolvement adding racks and switches can cause coexistence of heterogenous switches and cutting links can also create asymmetries [5] [6]. We will analyze below that asymmetry exacerbates the inaccuracy of congestion feedback. ECMP not accounting for either flow size or current network utilization overwhelms switch buffers and degrades link bandwidth utilization.

Prior solutions have made a great deal of effort to improve performance under the situation described above. Taking deployment challenges into account, some optimized alternatives (e.g. CLOVE-ECN [7], Hermes [4]) to ECMP balance load at the virtual edge or hosts to handle asymmetry and keep practical. However, they depend too much on the rough congestion feedback (e.g. ECN and coarse-grained RTT measurements) to balance load, and so performance can be degraded. CLOVE-ECN learns congestion along network paths using ECN signals and uses a weighted round-robin (WRR) algorithm to dynamically route flowlets on multiple paths. Hermes also exploits ECN signals and coarse-grained RTT measurements (e.g. introducing end host network stack delay) to decide the flow path at the host side. Inaccurate ECN signals and coarse-grained RTT measurements may 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 can be believable if and only if a small RTT is discovered. Furthermore, ECN is a passive and delayed mechanism for informing congestion level in multiple paths, and so it can hardly achieve timely load balancing. Therefore, current load-balancing solutions cannot accurately sensing network congestion under asymmetry without custom switches.

Actually the end-to-end latency effectively reflects 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 [8]) have made the measurement of end-to-end latency possible with microsecond accuracy. [9] shows that latency-based implicit feedback is accurate enough to reveal path congestion. DPDK now supports all major CPU architectures and NICs from multiple vendors (e.g. Intel, Emulex, Mellanox, and Cisco). [10] reveals that a tuned DPDK solution such as TRex [11] only introduces 5μs to 10μs overhead. Several latency-based congestion control protocols for datacenter networks have emerged (e.g. Timely [12], DX [9]). With the help of DPDK, the end-to-end latency can be measured with sufficient precision.

Moreover, current load-balancing solutions also create new inaccurate congestion feedbacks in transport protocols under asymmetry. And this phenomenon is called congestion mismatch first unveiled in [4]. Due to that transport protocols do not perceive multipathing to adjust the flow rate, rerouting events can cause a mismatch between the sending rate and the state of the new path. This problem hinders the utilization of link bandwidth especially under asymmetric topologies. When the problem occurs, the 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. Therefore, current ECN-based congestion control mechanism cannot avoid the chaos of congestion feedbacks and can mistakenly adjust flow rate (we called it inaccurate rate adjustment problem in this paper).

According to above observation, we find that inaccurate congestion feedback under asymmetry exists in current load-balancing schemes and this motivates us to solve the problem to improve performance. Finally we present ALB, which achieves accurate congestion feedback at end hosts with commodity switches and provides competitive performance with those schemes requiring custom switches (e.g. LetFlow [13], CONGA [14]).

We make the following contributions in this paper:

- We analyze that the inaccuracy of congestion feedback can degrade performance under asymmetry in load balancing above.
- 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. Compared with Hermes, ALB requires no complicated parameter settings and provides competitive performance.
- In large-scale simulations we show that ALB achieves up to 7% and 40% better flow completion time than CONGA and CLOVE-ECN under asymmetry.

II. Design

A. Overview

We present ALB's framework in Fig. 1. ALB contains two modules, which are MDCTCP and ALB core. We design MDCTCP by slightly modifying DCTCP [15]. 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.

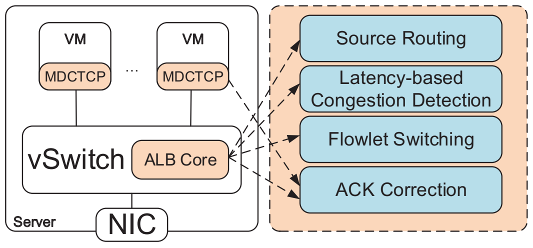


Fig. 1. The design of ALB

The source routing is used for path discovery. Then, the latency-based congestion detection serves to provide the accurate queuing delay for each path. So the source vSwitch can timely obtain the end-to-end latency and RTTs for every flow. Moreover, ALB divides each flow into flowlets to balance load with the latency-based congestion detection in a fine-grained way. Finally, the ACK correction function correcting the inaccurate rate adjustment needs the cooperation of MDCTCP and ALB core.

B. The Detailed Design of ALB

a) Source Routing: In order to route flows in all paths on the source side, ALB uses the traceroute mechanism in source vSwitch similar to CLOVE. The commodity datacenter switches inherently implement ECMP and the overlay network generally exists. By sending probes with varying source ports in static ECMP-based datacenter networks, ALB routing module can find a subset of source ports that lead to distinct paths. And the routing module can modify the source port of tunnel encapsulation (e.g. using Stateless Transport Tunneling (STT) protocol) to control transmission path of every flow.

b) Latency-based Congestion Detection: ALB measures the RTT and one-way delay for every flow under the clock without synchronization. Compared with ECN-based congestion detection, our algorithm can characterize the degree of path congestion more accurately through the end-to-end queuing delay detection, and feedback congestion metrics more quickly by leveraging other normal connections.

ALB timestamps packets in the option fields of TCP header at the DPDK-based device driver so as to calculate the RTT and one-way delay. For example in Fig. 2 we record NIC time t1, t2, t3 and t4 into the option fields of TCP header. We calculate the one-way delay by subtracting the baseline one-way delay, which is measured by picking the minimum among enough samples, from the current one-way delay under the clock without synchronization. We update the baseline delay in every few RTTs to avoid clock drifts [9] problems. In addition to calculating the one-way delay and measuring the 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. The feedback metric includes

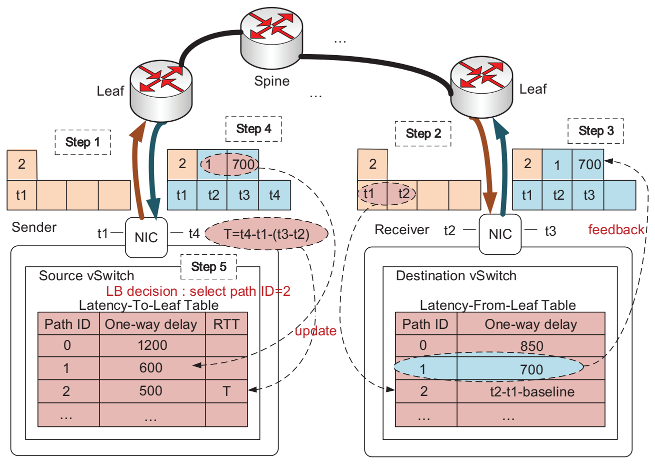


Fig. 2. The framework of latency-based congestion detection

two path IDs and a one-way delay of the feedback path, which requiring only no more than 6 bytes can be encoded in the fields of tunnel encapsulation Stateless Transport Tunneling (STT) context. The first 2 bytes are for two path IDs and remaining 4 bytes are used for storing the feedback one-way delay. The STT context containing 64 bits provides sufficient fields for our design. Referring to Fig. 2 for an example, we describe the process as the following five steps:

1) ALB selects the 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 only when $RTT_{new} \leq RTT_{old}$ (RTT_{old} represents the old RTT value). Then we update one metric in Latency-To-Leaf table with the feedback metric in tunnel encapsulation.

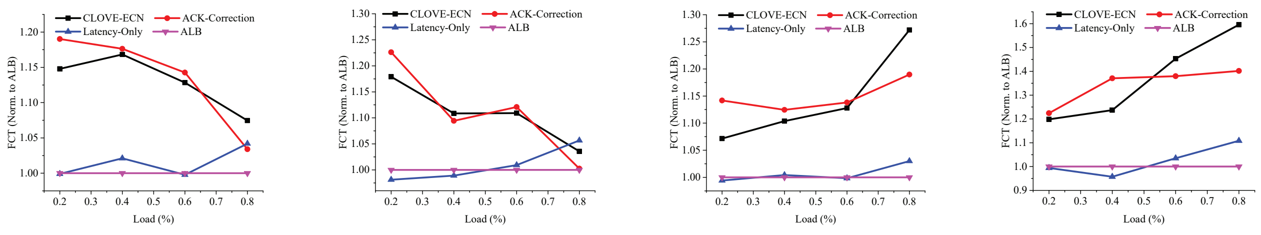
It is important to emphasize that we describe a simplified process of exchanging latency metrics above. Only the RTT measurement needs to wait the ACK packet to convey timestamps, and actually every packet simultaneously carries both timestamps to calculate forwarding one-way delay and a feedback metric. 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. Furthermore, it is possible that the metrics become unused if there is not enough feedback packets because of delayed ACKs and the lack of other connections to piggyback them. So we add a UDP-based daemon in vSwitch to improve the visibility of latency. If the metric in Latency-From-Leaf table has not been transferred more than β (β is a tunable parameter in ALB) RTTs, the UDP-based daemon should send it to destination vSwitchs which are under the corresponding leaf switch. To balance the bandwidth cost with performance gain we suggest to set β to 2 after our many experiments.

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. So we update the baseline one-way delay according to the change of RTTs. Each time the value of RTT in mapping table is updated, we need to remeasure the baseline one-way delay to keep it fresh. Finally the latency-based congestion detection enables ALB to accurately reroute flowlets to the least congested paths.

c) Accurate Flowlet Switching: Flowlet switching [16] 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. Compared to previous congestion-aware mechanisms ALB leverages the latency-based congestion detection to achieve more accurate flowlet switching while working with commodity switches. This method is a trade-off between performance improvement and difficulties in deployment for load balancing. Unlike prior schemes relying on ECN-based congestion detection, ALB does not need to rely on ECN feedback. The flowlet switching in ALB is simple that a new flowlet always selects the least congested path which has the smallest one-way delay referring to the Latency-To-Leaf table in source vSwitch (Fig. 2).

d) ACK Correction: ACK correction needs to pass the signal of state mismatch to MDCTCP. When one returned ACK reaches the source vSwitch, ALB calculates the flowlet ID of this flow according to the fields of protocol header and get the mapped path ID in mapping table. Another path ID identifying which path the congestion feedback belongs to is piggybacked by the ACK (referring to Fig. 2). Then ALB compares whether the mapped path ID and the one piggybacked in tunnel encapsulation of this ACK are equal. If these two path IDs are not equal, ALB modifies a reserved bit in the TCP header to 1, otherwise sets the bit to 0. Therefore, the reserved bit can indicate whether the congestion feedback belongs to the current state. ALB passes 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 flag in



(a) Overall avg FCT (b) Large flow (>10MB) avg (c) Small flow (<100KB) avg (d) Small flow 99^{th} percentile

Fig. 3. FCT of different methods (normalized to ALB).

TCP header joins the management of sending rate. The only difference between MDCTCP and DCTCP is in how the sender reacts to receiving an ACK. Because zeroing the PCN flag means that this is a matched feedback, MDCTCP only adjusts the congestion window and threshold when the PCN flag 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, 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 avoid the influence of congestion feedback that does not belong to the current path. In this way we avoid the inaccurate rate adjustment. But our algorithm can reduce the total amount of packet statistics at the sender, because those ACKs whose PCN flag marked with 1 will be removed from the packet statistics. 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, the value α becomes high ($\alpha = 1$) so as to the congestion window can even be mistakenly cut in half because of inaccurate rate adjustment, and more flow rerouting events occur in this situation. Therefore MDCTCP significantly improves performance at heavy load. The evaluation result in Fig. 3 also validates our analysis, in which ACK-Correction works better at high load.

III. Evaluation

A. Functionality Verification

We run a trace-driven simulation to expose the performance degradation due to inaccuracies, which is based on the web-search workload in a 4×4 leaf-spine topology with 10Gbps links and 32 servers in NS3. We make the topology asymmetric through reducing the capacity from 10Gbps to 2Gbps for 20% of randomly selected leaf-to-spine links.

In ALB, we use ACK correction to avoid inaccurate rate adjustment, while using latency-based congestion detection to improve accuracy of congestion detection on multiple paths. We show the performance improvement due to fixing these inaccuracies in Fig. 3. In Fig. 3 Latency-Only indicates the solution where only latency-based congestion detection is implemented, while ACK-Correction means that the effect of inaccurate ACK feedbacks is ruled out from CLOVE-ECN. 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). ACK-Correction works better at the heavy load (at 80% loads), while Latency-Only always performs better than CLOVE-ECN. 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 occurs. So 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 up to 14% better overall average FCT and 37% better 99th-percentile FCT for small flows than CLOVE-ECN. The combination of latency-based congestion detection and ACK correction can provide significant performance improvements.

B. Performance Comparison

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

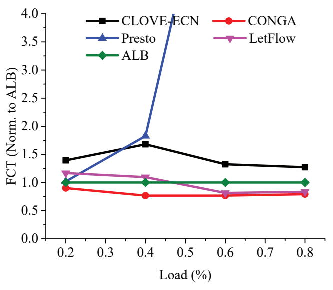
Workloads: We use two widely-used realistic workloads observed from deployed datacenters: web-search [15] and data-mining [17].

Metrics: Similar to previous work, we use flow completion time (FCT) as the primary performance metric. We normalize the FCT to ALB to better visualize the results.

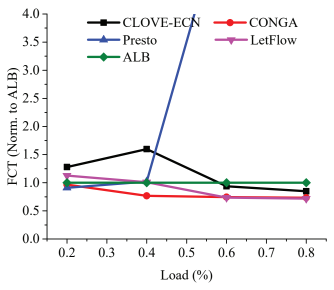
Topology: We build a 8×8 leaf-spine topology with 10Gbps links and 128 servers with NS3. Therefore we simulate a 2:1 oversubscription at the leaf level to meet the typical deployment of current datacenters [14]. To compare ALB with above schemes under asymmetry, we reduce the link capacity from 10Gbps to 2Gbps for 20% of randomly selected leaf-to-spine links.

Methodology: Besides ECMP we compare ALB with four schemes using DCTCP as the default transport protocol: an idealized variant of Presto with a reordering buffer, LetFlow, the state-of-the-art scheme CONGA and CLOVE-ECN.

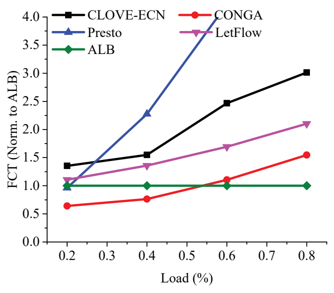
Note: We do not compare all previous solutions. We first should choose schemes using the ECN-based congestion control protocol. Then because Hermes's performance is close to CONGA when there is no switch failure and Hermes introduces a large number of parameter settings that make it hard



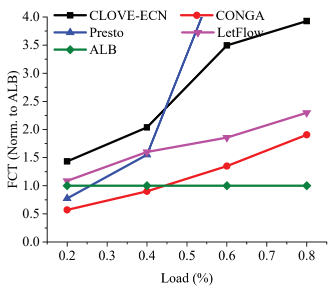
(a) Overall avg FCT



(b) Large flow (>10MB) avg

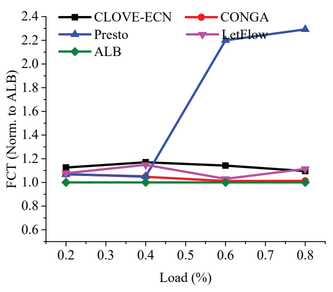


(c) Small flow (<100KB) avg

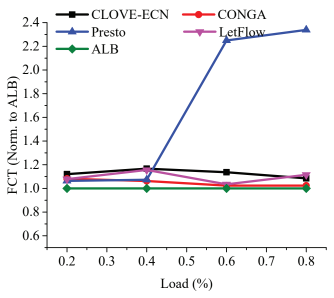


(d) Small flow 99^{th} percentile

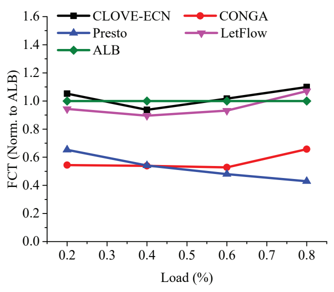
Fig. 4. FCT for the web-search workload in the asymmetric topology (normalized to ALB).



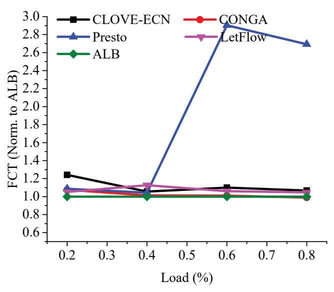
(a) Overall avg FCT



(b) Large flow (>10MB) avg



(c) Small flow (<100KB) avg



(d) Overall 99^{th} percentile

Fig. 5. FCT for the data-mining workload in the asymmetric topology (normalized to ALB).

to set up to achieve optimal performance for us, we have not simulated it. Additionally we do not compare against MPTCP because of performance instability and worse performance gains compared with CONGA in many scenarios. We try several different flowlet timeout values in CLOVE-ECN and CONGA, and adopt the most appropriate one (250μs) in our simulations for all flowlet-based solutions.

Under the web-search workload: As shown in Fig. 4, 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) can balance load more faster than other solutions deployed at end-hosts. But at 0.2-0.4 load, ALB performs 8-14% 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 21-40%. This is because ALB based on accurate congestion feedback can reroute new flowlets more accurately and avoid the inaccurate rate adjustment at end-hosts. In [4], Hermes only achieves similar performance with CLOVE-ECN under the web-search workload. Moreover, as shown in Fig. 4(c) and Fig. 4(d), the average and the 99th percentile FCTs for small flows grow dramatically for all of schemes except ALB. This is because under high loads more flowlets are created by small flows and they are seriously affected by congestion mismatch. Because of accurate rate adjustment, ALB can alleviate the congestion mismatch problem. In comparison, at 0.8 load ALB improves the average and the 99th percentile FCTs for small flows by 52-201% and 74-129% respectively.

Under the data-mining workload: As shown in Fig. 5, ALB achieves 2-7% 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. ALB can balance load accurately even though there are not too much bursty arrivals of new flows. And CONGA implemented in switches cannot handle the inaccurate rate adjustment problem. Therefore ALB can outperform CONGA slightly. Moreover, ALB achieves 3-13% and 9-14% better performance than LetFlow and CLOVE-ECN respectively. This is because the data-mining workload is less bursty and so the visibility of network congestion becomes especially important to balance load effectively. Based on accurate congestion feedback ALB achieves better visibility of network congestion than LetFlow and CLOVE-ECN.

For Presto, we take into account the network asymmetry by using static weights (based on the topology) to make load-balancing decisions [13]. However, Presto does not achieve comparable performance to ALB. This is because congestion mismatch problem. When load gets heavy under asymmetry, the congestion window in Presto is constrained by the most congested path and the flow rate is adjusted in a chaotic way. This causes high FCTs under asymmetry.

IV. Related Work

Centralized Mechanisms: The centralized scheduler used in centralized mechanisms (e.g. Hedera [18], MicroTE [19] and FastPass [20]) monitors global network state and schedules flows evenly in multiple paths. But they have long scheduling interval, which is not adaptive to the traffic volatility of datacenter networks. Therefore the latency-sensitive flows do not fit into these centralized mechanisms.

In-Network Distributed Mechanisms: Some in-network solutions based on local state (e.g. Flare [16], LocalFlow [21], Drill [22] and LetFlow [13]) route flowlets according to local link utilization. Due to the lack of global congestion detection, they perform poor with dynamic traffic changes and varying asymmetry. And some other mechanisms (e.g. CONGA [14], HULA [23] and CLOVE-INT [7]) which employ custom switches to balance load with global visibility of link state. However, the requirement of custom switches creates difficulties in deployment and scalability. Besides, the rerouting events at in-network devices cannot cooperate with the congestion control protocol at the sender, which leads to inaccurate rate adjustment. ALB can sense the inaccuracy problem at source vSwitch to inform the sender.

Host-based Mechanisms: MPTCP [24] creates more burstiness and performs poorly under incast [14]. Flowbender [25] reroutes flows blindly when congestion is detected based on ECN signals at end hosts. Presto [2] routes flowcells to balance load at network edge. CLOVE-ECN [7] 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 [4] exploits ECN signals and coarse-grained RTT measurements to sense congestion on multiple paths, while ALB use a more accurate congestion detection. ALB requires no complicated parameter settings and provides competitive performance (e.g. ALB achieves up to 7% FCT than CONGA under asymmetry).

V. Conclusion

We propose ALB, an adaptive load-balancing mechanism based on accurate congestion feedback running at end hosts, which is resilient to asymmetry. ALB leverage a 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 requiring custom swithes or complicated parameter settings, ALB can provide competitive performance.

Acknowledgment

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.61502190 and CERNET Innovation Project NGII20170120.

References

[1] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992*, 2000.

[2] K. He *et al.*, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, 2010.

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

[5] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[6] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM SIGCOMM*, 2015.

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

[8] "Intel dpdk. data plane development kit." [Online]. Available: http://dpdk.org/.

[9] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Dx: Latency-based congestion control for datacenters," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 1, pp. 335–348, 2017.

[10] M. Primorac, E. Bugnion, and K. Argyraki, "How to measure the killer microsecond," in *Proc. ACM SIGCOMM Workshop on KBNets*, 2017.

[11] "Cisco systems. TRex: Cisco's realistic traffic generator." [Online]. Available: https://trex-tgn.cisco.com.

[12] R. Mittal *et al.*, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[13] 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.

[14] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014.

[15] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM*, 2010.

[16] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[17] A. Greenberg *et al.*, "Vl2: a scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009.

[18] 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.

[19] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. ACM CoNEXT*, 2011.

[20] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proc. ACM SIGCOMM*, 2014.

[21] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proc. ACM CoNEXT*, 2013.

[22] 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.

[23] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proc. ACM SOSR*, 2016.

[24] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011.

[25] 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.