AG: Adaptive Switching Granularity for Load Balancing with Asymmetric Topology in Data Center Network

Jingling Liu, Jiawei Huang, Weihe Li, Jianxin Wang
School of Computer Science and Engineering, Central South University, ChangSha, China 410083
Email: {jinglingliu, jiaweihuang, weiheli, jxwang}@csu.edu.cn

***Abstract*—Modern data center topologies often take the form of a multi-rooted tree with rich parallel paths to provide high bandwidth. However, various path diversities caused by traffic dynamics, link failures and heterogeneous switching equipments widely exist in production datacenter network. Therefore, the multi-path load balancer in data center should be robust to these diversities. Although prior fine-grained schemes such as RPS and Presto make full use of available paths, they are prone to experience packet reordering problem under asymmetric topology. The coarse-grained solutions such as ECMP and LetFlow effectively avoid packet reordering, but easily lead to under-utilization of multiple paths. To cope with these inefficiencies, we propose a load balancing mechanism called AG, which adaptively adjusts switching granularity according to the asymmetric degree of multiple paths. AG increases switching granularity to alleviate packet reordering under large degrees of topology asymmetry, while reducing switching granularity to obtain high link utilization under small degrees of topology asymmetry. AG is deployed on the switches with negligible overhead, while making no modification on end-hosts. We evaluate AG through both Mininet testbed and large-scale NS2 simulations. The experimental results show that AG reduces the average and 99^{th} flow completion time by up to 51% and 56% over the state-of-the-art load balancing schemes, respectively.**

***Index Terms*—data center, load balancing, asymmetry**

I. Introduction

Modern data centers are commonly organized in multi-rooted tree topologies such as Clos [1] and Fat-tree [2] with multiple available paths. The end-hosts utilize the multiple paths to deliver the increasing traffic of distributed applications such as web service, social networking and online retail. Unfortunately, due to traffic dynamics, link failures and heterogeneous switching equipments, production datacenters operate under various path diversities, making the topology asymmetric.

To fully utilize the parallel paths in data centers, many load balancing approaches have been proposed. As the standardized scheme in modern data center, Equal Cost Multi-Path (ECMP) [3] uses flow hashing to transfer flows to available paths. LetFlow [4] and CONGA [5] reroute flowlets to avoid packet reordering. Random packet spraying (RPS) [6], Presto [7] and DRILL [8] split flow at a finer granularity and then spray the fixed switching units across available paths to make full use of link resource.

However, most existing schemes are not aware of variation of network asymmetry. On the one hand, the flow-level ECMP randomly maps each flow to only one of the paths. The flowlet-level schemes are unable to timely reroute flowlets when necessary. Due to their coarse switching granularities, both flow-level and flowlet-level schemes easily suffer from the under-utilization of link resource, even when the topology is symmetric. On the other hand, fine-grained schemes such as RPS, Presto and DRILL are prone to experience packet reordering under asymmetric topology.

In this paper, to address the above inefficiencies, we propose a load balancing mechanism called AG, which adaptively adjusts switching granularity under different degrees of topology asymmetry. To achieve flexibility and resiliency to asymmetry, AG is sensitive to path latency and periodically adjusts its switching granularity. Specifically, AG increases the switching granularity to alleviate packet reordering under high degrees of topology asymmetry, and reduces the switching granularity to obtain high link utilization otherwise. We implement AG on the switches with negligible overhead, while making no modifications on the TCP/IP protocol stack of the end-hosts.

In summary, our major contributions are:

- We conduct extensive simulation-based studies to analyze the problems of current load balancing mechanisms under asymmetric topology, including the packet reordering of fine-grained load balancing schemes and the under-utilization of coarse-grained ones.
- We propose a load balancing mechanism AG, which adjusts the switching granularity according to the degree of topology asymmetry. We present the theoretical analysis on the optimal switching granularity and the design of granularity adjustment algorithm.
- We evaluate AG in the Mininet implementation and large-scale NS2 simulations under different realistic workloads. The results show that AG effectively reduces the average flow completion time by up to 51% over the state-of-the-art load balancing schemes. Furthermore, the implementation overhead of AG is acceptable.

The rest of the paper is organized as follows. We investigate the problems of load balancing with different granularities under asymmetric topology in Section II. We present the design overview and details of AG in Section III-V. We evaluate AG performance with NS2 simulations and Mininet implementation in Section VI and VII, respectively. We discuss the related works in Section VIII and then offer concluding remarks in Section IX.

978-1-7281-2700-2/19/$31.00 2019 © IEEE

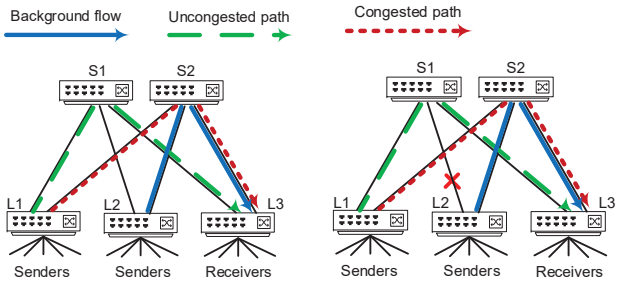
II. Background and Motivation

In this section, we firstly describe the topology asymmetry of data center networks and then elaborate the drawbacks of load balancing schemes with different granularities.

A. Topology Asymmetry

Data center networks enable multiple paths between host pairs to concurrently transfer traffic. However, the latency of multiple paths might be different due to traffic dynamics and link failures [9], making the topology asymmetric. We use an example to show the causes of topology asymmetry in a typical Leaf-Spine data center network.

Traffic variations change the degree of asymmetric topology [10]. As shown in Fig. 1(a), a dynamic background flow is transmitted by ECMP from leaf switch L2 to leaf switch L3. Since the link from spine switch S2 to leaf switch L3 is blocked by the background flow, the two paths from L1 to L3 becomes asymmetric. Fig. 1(b) shows the asymmetric topology caused by link failures. When the link between L2 and S1 breaks down, any traffic from L2 to L3 is forced to be transferred on path L2-S2-L3. Consequently, traffic from leaf switch L1 to L3 will go through two paths with different latencies. Besides, switch failures such as random packet dropping and heterogeneous switches with different link speed can also induce topology asymmetry [9].



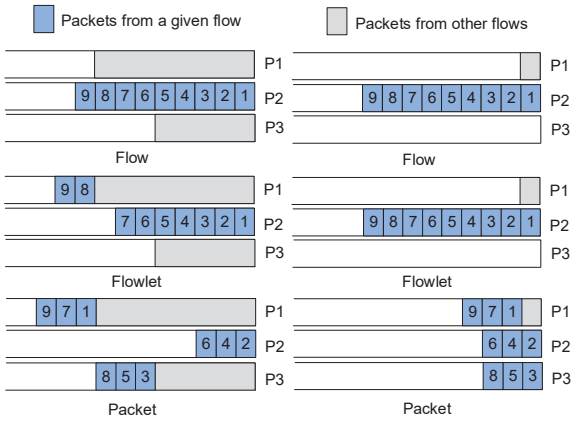
(a) Traffic Dynamics (b) Link Failures

Fig. 1. Topology Asymmetry.

B. Load Balancing Schemes with Different Granularities

Generally, prior load balancing schemes work at the flow, flowlet and packet level. In this section, we illustrate the working mechanisms of load balancing schemes with different granularities under different degrees of topology asymmetry.

In Fig. 2, three parallel paths with different queueing build-ups are provided for a single flow, which may be split with different switching granularities. Our design focuses on the switch granularity, not the way choosing path. Thus, we select RPS, ECMP and LetFlow as representatives due to their simplicity and randomness in choosing path. In fact, ECMP may choose the best or worst path. Here, we just show the best case to compare the impact of switching granularity size. As shown in Fig. 2(a), under large difference among paths, per-flow scheme such as ECMP may pick the path with shortest queue to reduce queueing delay and avoid packet reordering. For flowlet-based schemes, since the gap between two flowlets is always larger than the maximum latency difference among paths, the packet reordering is also



(a) Large Difference (b) Small Difference

Fig. 2. Motivation Examples.

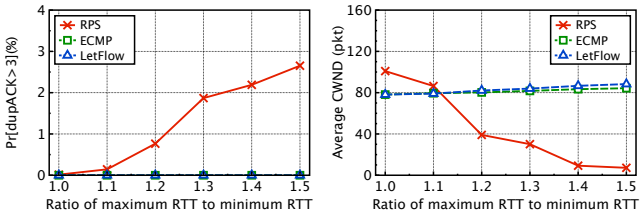
alleviated. Though packet-based schemes (e.g., RPS) utilize all available paths, they are prone to induce significant packet reordering, resulting in duplicate ACK packets (DUPACKs) and reduction of congestion window at end-hosts. Thus, under large degrees of topology asymmetry, coarse-grained schemes may achieve better performance than fine-grained ones.

However, as shown in Fig. 2(b), when the difference between three paths is small, coarse-grained schemes cause under-utilization of link resource. For example, path P3 is unutilized in flow-based schemes. Since the gap between consequent packets is not large enough to trigger rerouting flowlet, the flowlet-based scheme is also unable to fully utilize all paths. On the contrary, the packet-based scheme makes full use of all available paths. Moreover, since the path difference is very small, the impact of packet reordering is negligible. Therefore, the fine-grained packet-based scheme performs well under small degrees of topology asymmetry.

C. Network Performance under Fixed Switching Granularity

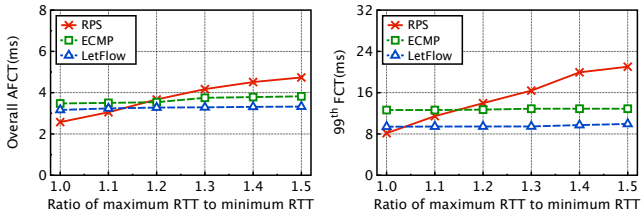
In this section, we conduct the NS2 simulations to investigate the performances of coarse-grained and fine-grained load balancing schemes. We compare the performances of ECMP, RPS and LetFlow. The test topology is a typical Leaf-Spine topology [11] with 3 Core switches and 2 ToR switches. There are 3 paths with 250μs propagation delay between any pair of hosts. The bandwidth of each link between source and destination leaf switch is 1Gbps. At the leaf level, the bandwidth oversubscription ratio is 4:3. The buffer size of each switch is 100 packets. We generate 2 flows between random pairs of hosts. The gap threshold of LetFlow is set as 500μs [4]. Each test result is the average value of 10 runs.

1) ***Packet reordering in fine-grained schemes*:** In order to evaluate the impact of asymmetric degree, we increase the propagation delay of one path. Among three paths, the ratio of maximum RTT to minimum RTT varies from 1 to 1.5. Fig. 3(a) shows the ratio of 3-dupack events caused by out-of-order packets to all packets. The coarse-grained schemes ECMP and LetFlow barely encounter out-of-order problem. On the contrary, the fine-grained scheme RPS inevitably experiences packet reordering. Under RPS, the larger difference of path latency leads to more disordered packets. Therefore, as shown



(a) Packet Reordering (b) The Average Congestion Window

Fig. 3. Packet reordering with Asymmetry.

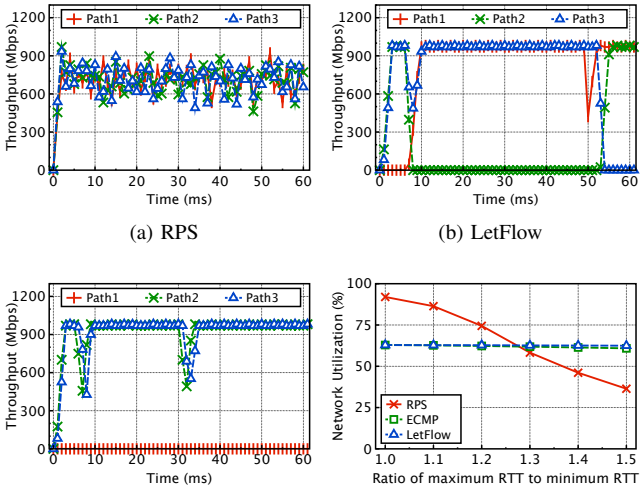


(a) Overall AFCT (b) 99^{th} FCT

Fig. 5. The Impact of Asymmetric Topology.

in Fig. 3(b), the TCP congestion window of end-host is also reduced under large degrees of topology asymmetry, greatly degrading the RPS performance.

2) ***Under-utilization of link resource in coarse-grained schemes*:** We measure the link utilization of 3 paths under three schemes. We measure the throughputs of 3 paths with the ratio of maximum RTT to minimum RTT as 1.1. RPS makes full use of the link resource and obtains high throughput as show in Fig. 4(a). However, coarse-grained schemes such as ECMP and LetFlow always lead to suboptimal link utilization. Fig. 4(b) and Fig. 4(c) show that, with ECMP and LetFlow, some links experience congestion, while the throughputs of other ones are as low as zero. Fig. 4(d) shows the total network link utilization when the ratio of maximum RTT to minimum RTT increases from 1 to 1.5. When the asymmetric degree is small, RPS achieves the highest link utilization since it scatters packets across all available paths. However, under large degrees of topology asymmetry, the link utilization decreases sharply due to the packet reordering.



(a) RPS (b) LetFlow

(c) ECMP (d) Network Utilization

Fig. 4. Under-utilization of Link Resource with Asymmetry.

Finally, we test the network performance under realistic workload. We generate 50 flows following a Poisson process with flow size distribution of Data Mining workload [2]. There are 8 paths with the propagation delay as 250μs between any pair of hosts. Fig. 5(a) and Fig. 5(b) show the average flow completion time (AFCT) and the 99^{th} percentile flow completion time (99^{th} FCT), respectively. When the asymmetric degree is low, RPS achieves the lowest AFCT and 99^{th} FCT due to the benefit of high network utilization. Under the high degree of topology asymmetry, however, since the positive effect of alleviating packet reordering dominates, ECMP and LetFlow obtain lower flow completion time than RPS.

D. Summary

The analysis of the load balancing schemes with different granularities leads us to conclude that: (i) the coarse-grained schemes are unable to make full use of link resource and may lead to under-utilization of network resource. (ii) though the fine-grained schemes utilize all available paths, they induce significant packet reordering under asymmetric topology. Therefore, we propose an asymmetry-aware load balancing mechanism, which adaptively adjusts switching granularity under different asymmetric degrees to mitigate packet reordering and obtain high link utilization. In the rest of the paper, we will elaborate on our design details.

III. AG Overview

In this section, we present the architecture of our design AG. Our goal is to design a load balancing mechanism adjusting switching granularity based on the latency difference among multiple paths to achieve the tradeoff between packet reordering and link utilization. When the difference in path latency is small, the switching granularity is fine-grained. Therefore, the flows are able to efficiently make full use of multiple paths and quickly complete their transmissions. In contrast, when the latency difference becomes large, the switching granularity adaptively increases to be robust and resilient to packet reordering.

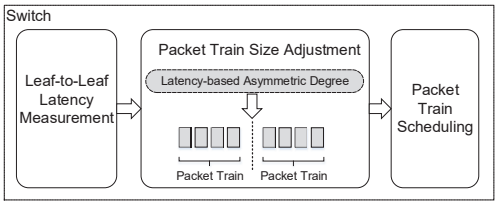


Fig. 6. AG Overview.

In our design, the two key points of AG are how to measure the latency difference among multiple paths and how to adjust switching granularity according to the asymmetric degree of multiple paths. Fig. 6 shows the architecture of AG, which is deployed on ToR switches and has three modules.

1) ***Leaf-to-Leaf Latency Measurement*:** A congestion-aware load balancing approach has advantages over congestion-agnostic ones. However, handling congestion in asymmetric topology essentially requires global knowledge about each path. In our design, to obtain the accurate path information with very small overhead, AG periodically

measures the latency of each path from the source leaf switch to the destination one. To control the measurement overhead, only a few data packets are selected as the probe packets. Moreover, AG uses the timestamp of TCP header in probe packet to piggyback the path latency information, without any modifications on existing TCP protocol.

2) ***Packet Train Adjustment*:** The switching granularity is a key point of our design. On the one hand, too small granularity exacerbates the adverse effect of packet reordering under asymmetric topology. On the other hand, too large granularity is prone to cause load imbalance and link under-utilization. The switching granularity of AG is called as packet train, which is a burst of packets belonging to one congestion window in a single flow. The packet train adjustment module adaptively adjusts the size of packet train according to the latency-based asymmetric degree of all paths measured by Leaf-to-Leaf latency measurement module. We develop a rational model to select the optimal packet train size based on asymmetric degree in Section IV-A.

3) ***Packet Train Scheduling*:** The packet train scheduling module picks the forwarding path for each packet train. In order to avoid synchronization effect of herd behavior when multiple packet trains choose the same path, AG simply chooses a path at random for each packet train. For a new packet train, the first packet is scheduled to a random path and the subsequent packets are assigned to the same path until the total number of packets sent exceeds the packet train size calculated by the packet train adjustment module.

IV. Packet Train Adjustment

In this section, we firstly analyze the optimal size of packet train, and then give the packet train adjustment algorithm.

A. Packet Train Size Optimization

The packet train size affects both the TCP reordering probability and network utilization under different asymmetric degrees. We give the analysis on how to get the optimal value of packet train size as following.

Let W and $gran$ denote the TCP congestion window and packet train size, respectively. Then the number of packet trains X in the congestion window W is $\frac{W}{gran}$.

When the packet trains are transferred on multiple paths, a packet train is out-of-order only when at least one packet train sent later arrives at the destination earlier. We assume that the X packet trains may select M parallel paths, which consist of M_b congested paths and M_g uncongested paths with the propagation delay RTT_b and RTT_g, respectively. In this case, the out-of-order event occurs when one packet train is transmitted on congested path and at least one packet train sent later is transmitted on uncongested path. Therefore, the reordering probability P of X packet trains is calculated as

$$P = \sum_{i=1}^{X-1} P_g^{i-1} \times P_b \times (1 - P_b^{X-i}), \tag{1}$$

where P_g and P_b are the probabilities that the packet train selects the uncongested and congested paths, respectively. Specifically, P_g and P_b may be various under different load balancing schemes. In our design, in order to avoid synchronization effect, each packet train is randomly assigned to one of the available paths. Thus, the probability P_b that a congested path is selected is calculated as $\frac{M_b}{M}$.

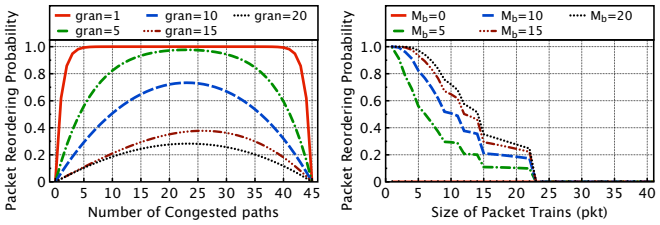
Then, we get the probability P_g that an uncongested path is selected as

$$P_g = 1 - \frac{M_b}{M}. \tag{2}$$

Substitute P_b and P_g into Equation (2), we get the reordering probability P as

$$P = \sum_{i=1}^{X-1} (1 - \frac{M_b}{M})^{i-1} \times \frac{M_b}{M} \times (1 - (\frac{M_b}{M})^{X-i}). \tag{3}$$

Fig. 7 shows the reordering probability P with increasing numbers of congested paths M_b and packet train size $gran$. The total number of paths is 45. W is set as 64KB (i.e., about 44 packets) due to the limitation of the Advertised Window field in TCP header. Since the packet train is not larger than a congestion window, the packet train size is in [1, 44]. Fig. 7(a) shows that the adverse effect of packet reordering is exacerbated under the larger asymmetric degree. When half of paths are congested, the most serious packet reordering occurs. As depicted in Fig. 7(b), the reordering probability decreases with the increasing packet train size under different asymmetric degrees. For instance, a small packet train easily leads to significant reordering, while a large one does not induce out-of-order problem.



(a) Increasing M_b (b) Increasing $gran$

Fig. 7. Packet Reordering Probability

When detecting the out-of-order packet, the TCP sender reduces its congestion window by half. Thus, with packet reordering probability P and maximum congestion window W, we get the average congestion window W_0 as

$$W_0 = (1 - P) \times W + P \times \frac{W}{2}. \tag{4}$$

Though the small size of packet train leads to large packet reordering probability, the small packet trains could utilize more paths, increasing the total utilized bandwidth. Given the link bandwidth C for each path, since each packet train randomly picks its transmission path, the total bandwidth for X packet trains is $X \times C$.

Typically, the end-to-end latency mainly consists of the queueing and propagation delay. We obtain the average end-to-end round-trip time RTT as

$$RTT = \frac{W}{X \times C} + P_g^X \times RTT_g + (1 - P_g^X) \times RTT_b. \tag{5}$$

With the average RTT and congestion window W_0, we get the average rate r as

$$r = \frac{W_0}{RTT} = \frac{(1-P)\times W + P \times \frac{W}{2}}{\frac{W}{X \times C} + P_g^X \times RTT_g + (1 - P_g^X) \times RTT_b}. \quad (6)$$

Finally, we choose the optimal granularity $gran^*$ to obtain the maximum value of average rate r as

$$gran^* = \underset{gran \in [1,44]}{\arg\max} \quad \|r(gran)\|. \quad (7)$$

The numeric comparison of optimal granularity is shown in Fig. 8. With the increasing number of congested paths M_b, the optimal granularity becomes larger due to more adverse effect of packet reordering. We also conduct the NS2 simulation experiment in Leaf-Spine topology. There is one 1MB TCP flow using 45 paths to transmit its packets. The bandwidth oversubscription ratio is 10:9 at the leaf level. Other settings are the same as that in Section II-C. Fig. 8 shows that the simulation test result is consistent with the result of numeric analysis.

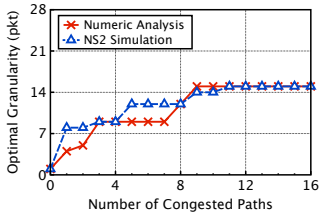


Fig. 8. Optimal Size of Packet Train with Varying M_b.

B. Packet Train Adjustment Algorithm

To cope with the packet reordering of fine-grained load balancing schemes and under-utilization issue of coarse-grained ones, AG adaptively adjusts the switching granularity according to asymmetric degree.

When the timer expires after timeout T, AG firstly estimates the latencies of all paths between the source leaf and destination leaf switch. Then AG updates the number of congested path whose latency is larger than the average RTT. With the path latency information, AG chooses the switching granularity such that the average rate r is maximal. Here, we use the binary searching to quickly find the optimal switching granularity. The initial search boundaries are 1 and 44. We set the maximum optional granularity as the maximum congestion window size, which is 64KB (i.e., about 44 packets). Note that the computational time complexity of the packet train adjustment algorithm is O($\log_2 n$) for n (i.e, 44) optional switching granularities. It only requires 6 iterations to find the optimal switching granularity. We enforce the optimal granularity to all active flows. When the number of arriving packets from a given flow is less than the optimal granularity, other flows do not follow the previous packet train to ensure in-order transmission. Though it may lead to a little traffic imbalance, since the maximum granularity is not large, the performance loss is still acceptable. We set the empirical timeout T as 500μs [9].

V. Leaf-to-Leaf Latency Measurement

To make correct forwarding decisions, the load balancing schemes should gather the congestion states from multiple paths. In our design, AG measures the latency from the source leaf switch to the destination one. However, to obtain the accurate global information in real-time manner with small overhead is a challenging issue. To address this issue, we utilize two reserved bits and the option fields of TCP header to collect the path delay with very small overhead. AG periodically measures the global latency of each path between the source and destination leaf switches.

As shown in Algorithm 1, when the timer expires after timeout T, AG marks the 1st reserved bit of packets sent from the source leaf switch to destination one. AG only marks one packet as probe packet on each path to limit overhead. Specifically, when a probe packet is sent from port p_s to the destination leaf switch, the source leaf switch firstly updates the timestamp in TCP header of the probe packet with current time, and meanwhile writes the subnet address of source leaf switch into the option field of the TCP header. Note that servers under the same ToR switch are assigned addresses with the same IP subnet [12].

When the destination leaf switch receives the probe packet from its port p_d, the leaf-to-leaf latency information will be updated by subtracting the probe packet's timestamp from the current time. Then, a data or ACK packet is selected to sent from port p_d to the source leaf switch which is specified by the subnet address in the probe packet header. The selected piggyback packet feedbacks the latency information. The destination leaf switch marks the piggyback packet with the 2nd reserved bit. The subnet address of destination leaf switch is also written into the option field of piggyback packet. Moreover, the destination leaf switch resets the 1st reserved bit and the option field of the probe packet to 0.

Finally, when receiving a piggyback packet from port p_s, the source leaf switch firstly records the port p_s, subnet address of destination leaf switch and one-way delay of corresponding path. Then, source leaf switch resets the 2nd reserved bit and option field of piggyback packet to 0. Since the packet trains are randomly transferred across all paths, the source leaf switch can collect the path latency from any output port to all destination ones. We calculate the one-way delay based on the assumption that the network devices are synchronized. Though Precision Time Protocol [13] can synchronize clocks, it requires hardware support and may lost precision with the increasing traffic load. To deal with the asynchronous problem, we calculate the real one-way queueing delay by subtracting the base delay from the measured one-way delay. The base delay is the one-way delay with zero queueing and is obtained by recording the minimum history delay. Therefore, even when the network devices are actually asynchronous, we still get real one-way queueing delay without time synchronization.

AG brings about limited overhead, since it only measures the one-way-delay by using source and destination leaf switches. Moreover, to reduce overhead and enhance

Algorithm 1: Measuring Leaf-to-Leaf Latency

Parameters:

$OWD[]$: One-way delay;
$r1$: The first reserved bit in the TCP header;
$r2$: The second reserved bit in the TCP header;
$time$: Timestamp in the option field of TCP header;
$saddr$: Source IP subnet in the option field of TCP header;
ToR_s: Source leaf switch;
ToR_d: Destination leaf switch;

Initialization:

$OWD[] \leftarrow \{\}$; $T \leftarrow 500\mu$s;

Function SOURCE_LEAF_SWITCH()
begin
  if *the timer expires after* T **then**
    select a packet $Spkt$ *sent from port* p_s *to* ToR_d *as probe packet*;
    $Spkt.r1$ = 1;
    $Spkt.time$ = *current_time*;
    $Spkt.saddr$ = $ToR_s.subnet$;
    reset the timer;
  On receiving piggyback packet $Rpkt$ *from port* p_s:
  if $Rpkt.r2$ == 1 && $Rpkt.saddr$ == $ToR_d.subnet$ **then**
    $pathID$ = *the path from port* p_s *to* ToR_d;
    $OWD[pathID]$ = $Rpkt.time$;
    $Rpkt.r2$ = 0;
    $Rpkt.saddr$ = 0;

Function DESTINATION_LEAF_SWITCH()
begin
  On receiving probe packet $Spkt$ *from port* p_d:
  if $Spkt.r1$ == 1 && $Spkt.saddr$ == $ToR_s.subnet$ **then**
    select a packet $Rpkt$ *sent from port* p_d *to* ToR_s *as piggyback packet*;
    $Rpkt.r2$ = 1;
    $Rpkt.time$ = *current_time* - $Spkt.time$;
    $Rpkt.saddr$ = $ToR_d.subnet$;
    $Spkt.r1$ = 0;
    $Spkt.saddr$ = 0;

scalability, AG periodically uses few data and ACK packets to carry the delay information in the option field of TCP header. Thus, the path delay is collected with very small traffic and deployment overhead. Meanwhile, it is worthy to note that we use only 10 bytes to record the state information of each path in the design of AG. Since the number of path is usually not large [14], given the 50-100MB SRAM available in the modern switching ASICs for storing path states [15], the deployment overhead on leaf switches can be neglected.

VI. Simulation Evaluation

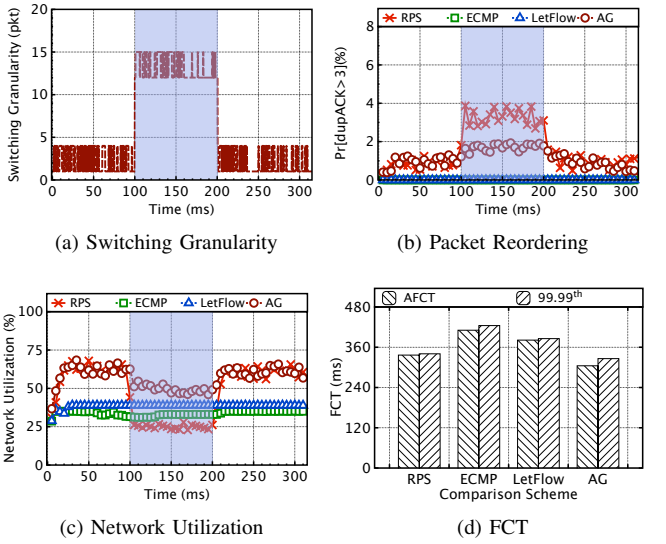
In this section, we conduct the NS2 simulation tests to evaluate the performance of AG. We firstly test the basic performance of AG, and then compare AG with the state-of-the-art schemes under datacenter workloads in large-scale test. We also evaluate the accuracy of leaf-to-leaf latency measurement. We finally test AG performance under different sample intervals. We use flow completion time (FCT) as the primary performance metric.

A. Basic Performance

In this section, we test the basic performance of AG. We compare the packet reordering, link utilization and flow completion time of AG, RPS, ECMP and LetFlow. We use the Leaf-Spine topology with 4 paths. The round trip propagation delay is 250μs and the link bandwidth is 1Gbps. We set the value of TCP RTO as 10ms [9]. The buffer size of each switch is 100 packets. We generate 2 TCP flows with size of 20MB. There is a 500Mbps UDP background flow on each path to induce congestion. Meanwhile, to produce topology asymmetry, we increase the sending rate of UDP flow to 800Mbps on one randomly selected path from 100ms to 200ms. We set the sample interval as 500μs. Each result is the average value of 10 runs.

Fig. 9(a) shows the switching granularity of AG. When the multiple paths are symmetric before 100ms or after 200ms, AG maintains small switching granularities to obtain high link utilization. While the asymmetric degree of multiple paths becomes large, AG increases switching granularity to alleviate packet reordering. Note that AG exhibits good convergence in adjusting the switching granularity.

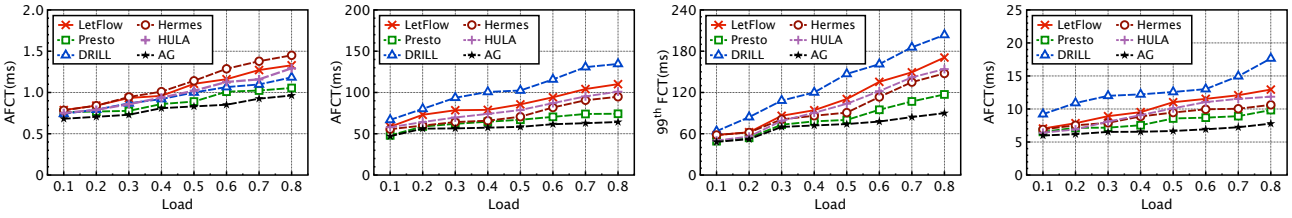
Fig. 9(b) shows the ratio of 3-dupack events caused by out-of-order packets to all packets. When there is no background flow, the packet reordering is not serious for all schemes, because the multiple paths are symmetric. After 100ms, however, the background flow makes the topology asymmetric. In this case, the flow-level scheme ECMP still totally avoids packet



(a) Switching Granularity (b) Packet Reordering

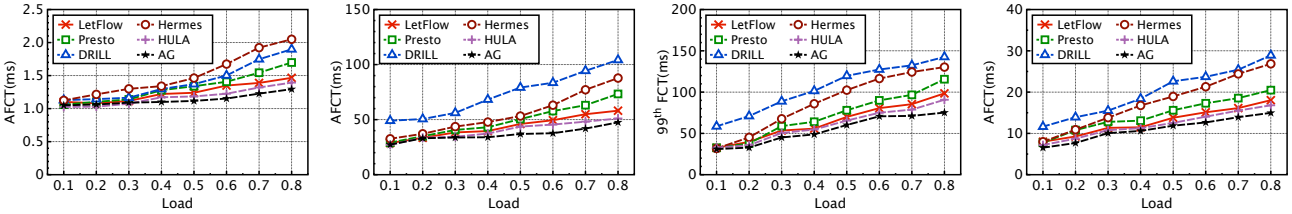
(c) Network Utilization (d) FCT

Fig. 9. The basic performance with different load balancing mechanisms.



(a) Small Flow (<100KB) AFCT (b) Long Flow (>1MB) AFCT (c) Overall 99^{th} FCT (d) Overall AFCT

Fig. 10. FCT for the Data Mining workload in the asymmetric topology.



(a) Small Flow (<100KB) AFCT (b) Long Flow (>1MB) AFCT (c) Overall 99^{th} FCT (d) Overall AFCT

Fig. 11. FCT for the Web Search workload in the asymmetric topology.

reordering. LetFlow has none out-of-order packets, because the flowlet timeout is larger than the maximum delay difference between the parallel paths. RPS experiences the most serious packet reordering exacerbated by the background flow, because it makes forwarding decisions for each packet with a random style. Compared with ECMP and LetFlow, AG experiences a little more packet reordering due to its random scheduling pattern. However, AG can sense the topology asymmetry and adjust the switching granularity. Therefore, the packet reordering ratio of AG is still acceptable (i.e., about 2%).

Fig. 9(c) shows the overall network utilization. Although RPS sprays packets to all paths, the network utilization of RPS is suboptimal due to the small sending rate caused by serious packet reordering. The network utilization of coarse-grained schemes such as ECMP and LetFlow is less than 50%, because 2 flows respectively almost fully utilize 2 paths at a time, while the other 2 paths are unused. AG achieves high network utilization, because it adaptively adjusts the switching granularity based on path asymmetry to make a tradeoff between the packet reordering and link utilization.

Fig. 9(d) shows the average and 99.99^{th} flow completion time. The reason that AG effectively reduces the FCT is twofold. On the one hand, AG selects an adaptable switching granularity based on the degree of topology asymmetry, alleviating packet reordering and avoiding the excessive reduction of TCP congestion window compared to the fine-grained scheme RPS. On the other hand, AG obtains higher link utilization by transmitting the packet trains on multiple paths, compared to the coarse-grained schemes such as ECMP and LetFlow.

B. Large-scale Simulation Test

In this test, we compare AG with other state-of-the-art solutions in large-scale simulations. We build an 8x8 leaf-spine topology with 1Gbps links and 256 servers. There are 8 equal cost paths with the propagation delay as 250μs between any pair of hosts. We set the round trip propagation delay of one randomly selected path as 300μs to produce delay asymmetry. The buffer size of each switch is 100 packets. We set the value of TCP RTO as 10ms [9]. We choose the realistic Data Mining workload [2] with mostly short flows and Web Search workload [17] with a mixture of short and long flows. We vary the overall workload from 0.1 to 0.8 to thoroughly evaluate the performance of AG. Moreover, we increase the asymmetric degree by changing the ratio of the maximum number to the average number of hosts under each ToR switch from 1 to 1.8 according to the increasing traffic load. We set the sample interval as 500μs. Each result is the average value of 10 runs.

We compare AG with the state-of-the-art load balancing schemes, such as Presto [7], LetFlow [4], DRILL [8], Hermes [9] and HULA [18].

Presto: Presto splits a flow at fixed flowcell granularity with the maximum TSO size (64 KB), and then chooses the paths in a round-robin way for the flowcells.

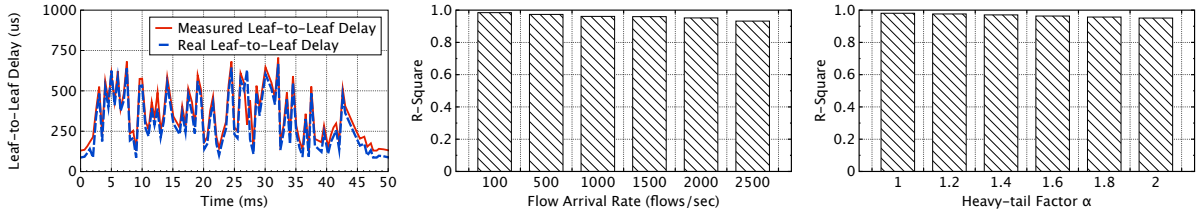
LetFlow: LetFlow simply picks a random path for each flowlet. In order to avoid packet reordering, the gap between flowlets is usually larger than the maximum delay difference between the parallel paths. In the test, the gap threshold is set as 500μs.

DRILL: DRILL is a packet-level load balancing scheme and makes forwarding decision for each packet based on the local congestion information at switch. It compares the queueing lengths of two random output ports and last selected port, and selects the one with the minimum queueing length to send the arriving packet.

Hermes: Hermes detects comprehensive path conditions and makes deliberate rerouting decisions when there is a path with better condition. The flow is rerouted only when its sending size exceeds a threshold (i.e., 600KB) and its sending rate is lower than a threshold (i.e., 30% of the link capacity). We set the parameters as the same as that in Ref [9].

HULA: HULA periodically tracks congestion of the best path using special probe packets, and reroutes the flowlets to the best next hop. In HULA, we set the flowlet gap as 500μs and the probe frequency as 200μs.

We show the simulation results of Data Mining workload and Web Search workload in Fig. 10 and Fig. 11, respectively. In Fig. 10(a) and Fig. 11(a), the average FCT of short flows increases with larger traffic load. Moreover, the FCT



(a) Leaf-to-Leaf Latency (b) Poisson Distribution (c) Heavy-tail Distribution

Fig. 12. Accuracy of Estimating Leaf-to-Leaf Latency.

improvement of AG also increases under large degrees of topology asymmetry. We observe that Hermes performs poorly for small flows because of its conservative rerouting decision. Under Hermes, the flow is rerouted only when its size exceeds a threshold (i.e. 600KB), resulting in low link utilization and large AFCT for short flows. Presto splits flow into a fixed granularity (i.e. 64KB). Though mitigating the issues of low link utilization and packet disordering to a certain extent, the fixed granularity of Presto is not adaptable to all asymmetric degrees. Due to the adaptive switching granularity, AG reduces the average FCT of short flows by up to 37%.

Fig. 10(b) and Fig. 11(b) show the average FCT of long flows under different traffic loads. Since DRILL makes forwarding decisions only according to the local information at switch buffer, it is prone to experience packet reordering under highly asymmetric topology. Therefore, DRILL displays the poor performance because of the packet reordering problem. Like LetFlow, HULA reroutes the flowlets only when the flowlets emerge. Since the Web Search workload contains more long flows, HULA has more chances to reroute flowlets, thus showing better adaptability and obtaining lower average FCT than that in the Data Mining workload. Meanwhile, HULA reroutes the flowlets to the best next hop, thereby obtaining better performance than LetFlow. Compared with the other schemes, AG alleviates the impacts of large queueing delay and out-of-order problem by adaptively adjusting the switching granularity of long flows according to the latency-based congestion conditions. Therefore, AG reduces the average FCT of long flows by up to 55% over other schemes.

Overall, AG significantly improves tail FCT and AFCT performances compared with the other four schemes. Fig. 10(c) and Fig. 11(c) show that AG reduces the tail FCT by around ∼23%-56% and ∼23%-53% under Data Mining and Web Search load, respectively. As shown in Fig. 10(d) and Fig. 11(d), AG has about ∼22%-51% and ∼18%-48% mean FCT improvements compared with other four schemes under Data Mining and Web Search load, respectively. Note that under Data Mining workload, since the flow sizes of 80% flows are less than 100KB [2], the overall average FCT and short flow's FCT are lower than that of Web Search workload. Meanwhile, as the long flow size in Data Mining workload is larger, the long flow's average FCT and overall 99^{th} FCT are higher than that of Web Search workload.

C. Measurement Accuracy of Leaf-to-Leaf Latency

In this test, we evaluate the measurement accuracy of leaf-to-leaf latency. We generate 10,000 flows according to Poisson processes. The flow sizes vary from 10KB to 5MB. The flows are sent from randomly selected senders to receivers. The topology settings are as same as that in Section VI-B. In order to analyze the measurement accuracy, we compare the measured leaf-to-leaf latency with the real one.

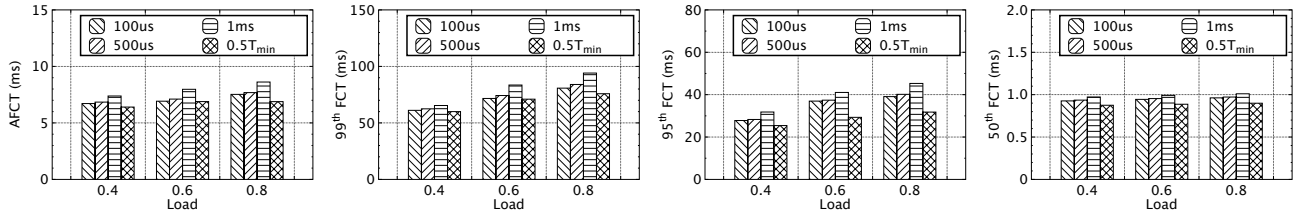
Firstly, the flows arrive at 100 flows/sec and the flow size obeys heavy-tail distribution [16] with parameter α as 1.5. Fig. 12(a) shows that the estimated one-way delay is very approximate to the real one measured from the NS2 trace file. Furthermore, we change the flow distribution and use R-Square as evaluation metric. R-Square is between 0 and 1, and the values closer to 1 indicates more accurate results. Fig.12(b) shows the results under Poisson distribution with the mean arrival rate from 100 flows/sec to 2500 flows/sec. Fig.12(c) shows R-Square under various power α of heavy-tail distribution of flow size. The larger power means more heavy-tail flow size. As shown in Fig.12, all R-Squares between the estimated leaf-to-leaf latency and the real one are more than 0.94, meaning that AG obtains accurate leaf-to-leaf latency with very small estimation error.

D. Impact of Sample Intervals

To capture the accurate global congestion states on multiple paths, AG needs to periodically measure the path latency to differentiate between congested and uncongested paths. A proper value of timeout T will obtain good tradeoff between high accuracy and low overhead. A simple method is to use a fixed and empirical value, such as 500μs. However, under real network, an optimal T should enable the switch to catch every change of congestion state, such as from congested to uncongested state or vice versa. Thus, we can use the statistical method to adaptively obtain a more accurate T.

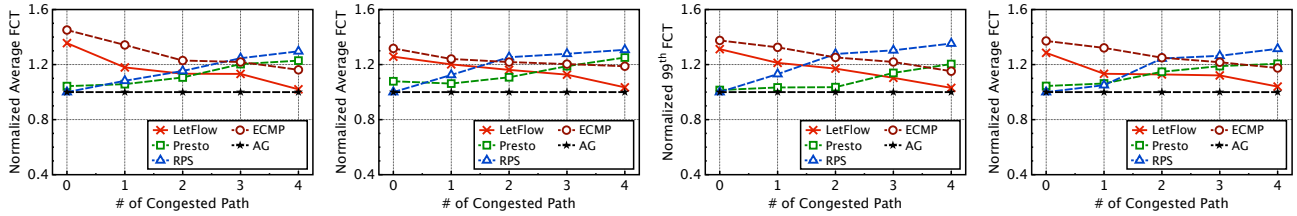
Specifically, we periodically collect a certain number of RTT samples of each path during the predefined time interval. For example, we measure the RTT of each packet in the 1st second in every one hour. If a RTT sample at one time point is larger than the average RTT, the path at this time is considered as congested. Otherwise the path is uncongested. Then, we calculate the average time period T_{avg} between state change (i.e., the path changes from congested to uncongested state, or vice versa). For multiple paths, we pick out the minimum value T_{min} from T_{avg} in all paths to represent the change rate of network state. Finally, according to Nyquist's theorem, the sampling period T is set as $0.5T_{min}$.

Next, we conduct the test to evaluate the parameter setting of sampling period T. We use a fixed T as 100μs, 500μs, 1ms, and $0.5T_{min}$. In this test, we use variation of congestion state at the beginning 10ms to determine T_{min}. In Fig.13, as the sampling timeout becomes greater, the interval between two adjacent sampling gets larger, AG misses some variations of network state, thus leading to the performance losses. On the



(a) Overall AFCT (b) Overall 99^{th} FCT (c) Overall 95^{th} FCT (d) Overall 50^{th} FCT

Fig. 13. Parameter Setting Analysis.



(a) Small Flow (<100KB) AFCT (b) Long Flow (>1MB) AFCT (c) Overall 99^{th} FCT (d) Overall AFCT

Fig. 14. Performance with Varying Number of Congested Paths.

contrary, our method can obtain a more accurate sampling timeout through statistic analysis, hence AG can perceive more variations of network state, thus achieving the best performance as a whole.

Nonetheless, the overhead still exists, although the statistic method gets the best performance compared to the fixed T. Therefore, in this paper, we use the sampling timeout of 500us to run the evaluation tests to reduce the overhead with few performance losses.

VII. Testbed Evaluation

In this section, we test the performance of AG through Mininet which is a network emulation system with high fidelity on Linux kernel [19], [20]. Compared to production data center networks, Mininet has smaller test scale and only supports tens of Mbps link bandwidth due to the limitation of single-machine CPU [20]. However, Mininet has been shown to faithfully reproduce implementation with high fidelity. Moreover, its codes and test scripts can be deployed into a real production network [19]. Thus, it is widely used as a flexible testbed for networking experiments [21], [22]. Therefore, we develop a Mininet-based prototype of AG with P4, which is a high-level language for programming protocol-independent packet processors [23]. P4 offers APIs for the software switch to parse packets and match header fields, and guides the switch to operate corresponding behaviors such as dropping and forwarding packet.

Specifically, we use header $ipv4\_t$ to declare an IPv4 header and record five-tuple information of the arrival packet. Each flow is identified by $hash()$ function, which adopts CRC16 algorithm to calculate the hash value of five-tuple information. We use the built-in metadata $ingress\_global\_timestamp$ to obtain the time when packet arrives at source leaf switch. We add an optional field $hdr.tcp.optional$ into TCP header to record the arrival time. Similarly, when receiving the packet, the destination leaf switch reads the recorded time in the TCP header and subtracts it from the current time. Finally, the calculated path latency is reordered in the header of data or ACK packet on the reverse direction.

With the measured path delay, the source leaf switch calculates the optimal switching granularity. We use two registers to record and compare the optimal switching granularity and the packet train length. If the packet train length is smaller than the optimal switching granularity, the packet train is not rerouted. Otherwise, the flow is rerouted to another random output port specified in $egress\_spec$ by the $random()$ function.

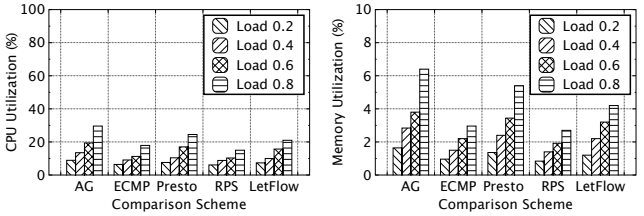
In this test, we implement AG on P4_16. We use Mininet 2.1.0 on a Ubuntu 16.04 LTS virtual machine to create a Leaf-Spine topology with 24 hosts connected by 2 leaf switches and 8 spine switches. There are 8 equal-cost paths between the leaf and spine switches. BMv2 is installed as the software programmable switch. Each port has a 100 packet buffer. We set link bandwidth to 20Mbps and the round trip propagation delay to 1ms, respectively [20]. We generate 80 flows with 90% small flows and 10% large flows following a Poisson process. We compare the performance of AG with ECMP [3], RPS [6], Presto [7] and LetFlow [4] with the number of congested paths varying from 0 to 4. We set the basic RTT of the congested path as 4ms to produce delay asymmetry.

We normalize the test results of ECMP, RPS, Presto and LetFlow to that of AG. Fig. 14(a) and Fig. 14(b) show the normalized AFCT of short flows and long flows under different numbers of congested paths, respectively. AG reduces the AFCT of short flows and long flows by up to 31% and 24% over the other schemes, respectively. The performance of the fine-grained schemes such as RPS and Presto deteriorates with the increasing number of congested path, because more number of congested paths aggravates the packet reordering, which triggers the reduction of congestion windows and leads to more spurious retransmission and bandwidth wastage. For the coarse-grained schemes such as ECMP and LetFlow, the FCT performances become worse due to the under-utilization of link resource under less number of congested paths. ECMP and LetFlow suffer from the randomness and inflexibility in rerouting flows. For LetFlow, once a flow is hashed to the path with large delay, the flow may be hard to be switched to other good paths due to small time gap between packets.

Fig. 14(c) shows that, compared with other four schemes,

the improvement of AG in tail FCT is around ∼18%-29%. Overall, AG has about ∼17%-27% AFCT improvements as shown in Fig. 14(d). The reason that AG reduces the average and tail FCT is two-fold. On the one hand, AG makes switching decisions according to the global network congestion. On the other hand, AG calculates an optimal granularity in load balancing for different congestion conditions, effectively alleviating packet reordering and link under-utilization.

AG is implemented on the switch with computing overhead. To evaluate the system overhead of AG, we measure the CPU and memory utilization ratio at the leaf switch with various traffic load as shown in Fig. 15. We conduct a Leaf-Spine topology with 24 hosts connected by 2 leaf switches and 8 spine switches. We generate 100 flows following a Poisson process with different arrival rates to change the traffic load. Fig.15 (a) shows the CPU utilization of the leaf switch. RPS achieves the lowest CPU utilization due to its simplicity in spraying packets to all paths. Although AG needs to conduct the packet train adjustment, it does not incur excessive CPU overhead compared with other schemes. Fig.15 (b) shows that even at 80% load, AG's memory utilization is only 6.5%.



(a) CPU Utilization (b) Memory Utilization

Fig. 15. Overhead of the Leaf Switch with Different Schemes.

VIII. Related Works

To achieve high bisection bandwidth in modern data centers organized in multi-rooted tree topologies, many load balancing approaches are proposed. These mechanisms can be divided into four categories according to different granularities.

The first category is the flow-based load balancing mechanisms. ECMP picks paths according to the five-tuple in packet header. WCMP [24] adds weights to different paths according to the path conditions. Hedera [21] and MicroTE [25] use a central controller to reroute the long flows to non-congested paths. Freeway [26] schedules the short and long flows separately on parititioned paths. Hermes [9] makes the switching decisions at packet level for long flows according to congestion conditions, and reroutes the short flows at a flow level. Due to the inflexibility feature, however, the flow-based mechanisms may lead to congestion and load imbalance.

To address the above problems, many packet-based schemes are proposed to make full use of multiple paths. RPS [6] randomly forwards packets of a flow to all paths at switches. DRB [12] chooses a round-robin method to transmit the arriving packets. Detail [27] performs the packet-level load balancing to reduce the tail latency. DRILL [8] compares two random output ports and the least-loaded port in the last round, and chooses the one with least queueing length to forward a packet. Though the above packet-based load balancing mechanisms can effectively improve the link utilization, they easily incur high degree of packet reordering under asymmetric topology. QDAPS [28] selects the output port for each packet based on the queueing delay of the last arriving packet for the same flow to avoid packet reordering. CAPS [29] encodes the short flows and spreads the packets of short flows to all path.

The third one is to balance load at the level of TCP Segmentation Offload (TSO) unit, called flowcell. Presto [7] is a typical flowcell-based mechanism which sprays the flowcells to all available paths in a round-robin style. Luopan [14] also operates at flowcell granularity. It periodically samples paths for each destination switch and selects the least congested one for flowcells. Since a fixed switching granularity still lacks enough flexibility and adaptability, it is hard to achieve the optimal performance under highly dynamic network.

The last category balances traffic based on the flowlet, which is a burst of packets separated from other bursts by a fixed time gap. CONGA [5] estimates the real-time congestion on each path and reroutes flowlets to the least load one. HULA [18] tracks global congestion by using programmable data planes. Clove [30] uses the vitual switch to dictate the path of each flowlet. LetFlow [4] picks a random path for each flowlet. ALB [31] reroutes flowlets based on accurate one-way delay. Unfortunately, the proper value of flowlet gap is hard to choose in advance. The multiple paths may be under-utilized with a large flowlet gap, while the packet reordering problem will occur frequently with a small one.

All above load balancing schemes with different granularities perform well in a certain network scenario, but none of them takes the asymmetric degree of multiple paths into consideration. Compared with these schemes, AG is a proactive and adaptive load balancer that adjusts switching granularity based on the asymmetric degree of multiple paths, achieving good performance under different network scenarios.

IX. Conclusion

We present the design and evaluation of AG, a simple load balancing scheme that adaptively adjusts switching granularity under asymmetric topology according to the latency-based congestion conditions of all paths. AG achieves high performance by alleviating packet reordering under large degrees of topology asymmetry and obtaining high link utilization under low degrees of topology asymmetry. We present the model analysis on how to obtain the optimal switching granularity according to the asymmetric degree. We evaluate and compare AG with the state-of-the-art load balancing schemes through both NS2 simulations and small-scale Mininet testbed. The experimental results demonstrate that AG reduces mean and tail FCT significantly without any changes to the TCP/IP protocol stack at end-hosts.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61872387, 61572530, 61872403), CERNET Innovation Project (Grant No. NGII20170107).

REFERENCES

[1] M. Al-Flare, A. Loukissas, A. Vahdat. A scalable, commodity data center network architecture, in Proc. ACM SIGCOMM, 2008.

[2] 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 Proc. ACM SIGCOMM, 2009.

[3] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm, in RFC 2992.

[4] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, T. Edsall. Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, in Proc. USENIX NSDI, 2017.

[5] 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 Proc. ACM SIGCOMM, 2014.

[6] A. Dixit, P. Prakash, Y. C. Hu, R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks, in Proc. IEEE INFOCOM, 2013.

[7] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akellay. Presto: Edge-based Load Balancing for Fast Datacenter Networks, in Proc. ACM SIGCOMM, 2015.

[8] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, A. Firoozshahian. DRILL: Micro Load Balancing for Low-latency Data Center Networks, in Proc. ACM SIGCOMM, 2017.

[9] H. Zhang, J. Zhang, W. Bai, K. Chen, M. Chowdhury. Resilient Datacenter Load Balancing in the Wild, in Proc. ACM SIGCOMM, 2017.

[10] S. Kandula, D. Katabi, S. Sinha, A. Berger. Dynamic Load Balancing Without Packet Reordering, ACM SIGCOMM Computer Communication Review, 2007, 37(2): 53-62.

[11] M. Alizadeh, T. Edsall. On the Data Path Performance of Leaf-Spine Datacenter Fabrics, in Proc. IEEE HOTI, 2013.

[12] 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 Proc. ACM CoNEXT, 2013.

[13] K. Lee, J. Eidson. IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems, in 34th Annual Precise Time and Time Interval (PTTI) Meeting, 2002.

[14] P. Wang, G. Trimponias, H. Xu, Y. Geng. Luopan: Sampling based Load Balancing in Data Center Networks, IEEE Transactions on Parallel and Distributed Systems, 2019, 30(1): 133-145.

[15] R. Miao, H. Zeng, C. King, J. Lee, M. Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs, in Proc. ACM SIGCOMM, 2017.

[16] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y. Chen, G. Zhang. SketchVisor: Robust Network Measurement for Software Packet Processing, in Proc. ACM SIGCOMM, 2014.

[17] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan. Data Center TCP (DCTCP), in Proc. ACM SIGCOMM, 2010.

[18] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes, in Proc. ACM SOSR, 2016.

[19] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown. Reproducible Network Experiments Using Container-Based Emulation, in Proc. ACM CoNEXT, 2012.

[20] H. Xu, B. Li. Repflow: Minimizing Flow Completion Times with Replicated Flows in Data Centers, in Proc. IEEE INFOCOM, 2014.

[21] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks, in Proc. USENIX NSDI, 2010.

[22] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P. B. Godfrey. Veriflow: Verifying Network-Wide Invariants in Real Time, in Proc. USENIX NSDI, 2013.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker. P4: Programming Protocol-Independent Packet Processors, ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[24] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, A. Vahdat. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers, in Proc. ACM EuroSys, 2014.

[25] T. Benson, A. Anand, A. Akella, M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers, in Proc. CoNEXT, 2011.

[26] W. Wang, Y. Sun, K. Salamatian, Z. Li. Adaptive Path Isolation for Elephant and Mice Flows by Exploiting Path Diversity in Datacenters, IEEE Transactions on Network and Service Management, 2016, 13(1): 5-18.

[27] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks, in Proc. ACM SIGCOMM, 2012.

[28] J. Huang, W. Lv, W. Li, J. Wang, T. He. QDAPS: Queueing Delay Aware Packet Spraying for Load Balancing in Data Center, in Proc. IEEE ICNP, 2018.

[29] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, T. He. CAPS: Coding-based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center, in Proc. IEEE INFOCOM, 2018.

[30] NN. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford. Clove: Congestion-Aware Load Balancing at the Virtual Edges, in Proc. ACM CoNEXT, 2017.

[31] Q. Shi, F. Wang, D. Feng, W. Xie. ALB: Adaptive Load Balancing Based on Accurate Congestion Feedback for Asymmetric Topologies, in Proc. IEEE IWQoS, 2018.