2018 IEEE 26th International Conference on Network Protocols

QDAPS: Queueing Delay Aware Packet Spraying for Load Balancing in Data Center

Jiawei Huang[§], Wenjun Lv[§], Weihe Li[§], Jianxin Wang[§], Tian He[‡]
[§]School of Information Science and Engineering, Central South University, Changsha, China 410083
[‡]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA 55455
Email: {jiaweihuang,wenjunlv}@csu.edu.cn, weiheleecsu@gmail.com, jxwang@csu.edu.cn, tianhe@umn.edu

***Abstract*—Modern data center networks are usually constructed in multi-rooted tree topologies, which require the highly efficient multi-path load balancing to achieve high link utilization. Recent packet-level load balancer obtains high throughput by spraying packets to all paths, but it easily leads to the packet reordering under network asymmetry. The flow-level or flowlet-level load balancer avoids the packet reordering, while reducing the link utilization due to their inflexibility. To solve these problems, we design a Queueing Delay Aware Packet Spraying (QDAPS), that effectively mitigates the packet reordering for packet-level load balancer. QDAPS selects paths for packets according to the queueing delay of output buffer, and lets the packet arriving earlier be forwarded before the later packets to avoid packet reordering. We compare QDAPS with ECMP, LetFlow and RPS through NS2 simulation and Mininet implementation. The test results show that QDAPS reduces flow completion time (FCT) by ∼30%-50% over the state-of-the-art load balancing mechanism.**

***Index Terms*—Data center; Multi-path; Load balancing**

I. Introduction

With the rapid development of cloud computing and big data techniques, more and more distributed applications such as web search, social networking and online retail are migrating to data centers. These applications generate huge amounts of traffic in data center networks, whose performance directly affects the user experience and the revenue of data center providers. Modern data centers are commonly organized in multi-rooted tree topologies such as Fat-tree [1] and Clos [2] to provide multiple equal cost paths for communication between different servers. The multipath transmission becomes critical in determining the performance of data center applications.

Random Packet Spraying (RPS) [3] is a simple packet-level load balancing scheme in data center networks. RPS splits each flow into packets on the switch and randomly sprays packets to all available paths to the destination. RPS works on a finer granularity to make good use of the multiple paths. Moreover, RPS does not require any change on the hosts and has already been implemented on many commodity switches such as Cisco [4].

Unfortunately, RPS easily leads to TCP out-of-order problem under network asymmetry. When the packets in a single flow are sent through multiple paths with different latency, the packets arriving at the receiver have a great chance of out-of-order. Since the current transport layer protocols such as TCP and DCTCP are not able to distinguish the reordered packets from the lost ones, RPS unavoidably brings about the reduction of TCP congestion window, resulting in the suboptimal network performance.

As a flow-level load balancer, Equal Cost Multi Path (ECMP) [5] randomly assigns flows to different paths. Though being widely deployed in data center, ECMP suffers from hash collisions and the inability to adapt to network asymmetry. In recent years, some flowlet-level load balancers such as CONGA [6] and Letflow [7] are proposed to split a flow into multiple flowlets and route a flowlet to one of the shortest path to the receiver. The flowlet-based mechanisms mitigate the packet reordering, but are not flexible enough as the flowlet gap is set to a fixed value [8]. As a result, it is hard to fully utilize all transmission paths.

We propose a packet-level load balancing design called Queueing Delay Aware Packet Spraying (QDAPS), that is both flexible to achieve high utilization and resilient to mitigate packet reordering under network asymmetry. QDAPS makes load balancing decisions on a per-packet granularity to obtain high utilization, eliminating the restriction of fixed granularity or passive path selection. Meanwhile, in order to avoid the packet reordering, QDAPS selects the output path for the arriving packet according to the queueing delay of the last arriving packet in the same flow. We implement QDAPS on the switches with negligible overhead, while making no modifications on the TCP/IP protocol stack of the end-hosts.

We make the following contributions:

- We conduct an extensive simulation-based study to analyze the problems of the current load balancing mechanisms, including TCP out-of-order of packet-level load balancing and the low link utilization of flow-level and flowlet-level ones.
- We propose a packet-level load balancing mechanism QDAPS, which selects the output port for a packet based on the queueing delay of the last arriving packet in the same flow. Specifically, the packet arriving earlier is forwarded before the later packets to resolve out-of-order problem.
- We evaluate QDAPS extensively in the Mininet implementation and large-scale NS2 simulations under different realistic traffic patterns. The test results show that QDAPS greatly reduces the average flow completion time (AFCT) by ∼30%-50% over the state-of-the-art load balancing mechanisms.

The rest of the paper is organized as follows. We analyze the TCP out-of-order problem of RPS and low link utilization

978-1-5386-6043-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICNP.2018.00017

of existing flow-level and flowlet-level mechanisms in Section II. In Section III, we describe the design and implementation details of QDAPS. We evaluate the performance of QDAPS through NS2 simulation and Mininet implementation in Section IV and Section V, respectively. We discuss the related work in Section VI and conclude the paper in Section VII.

II. Design Motivation

To motivate our design, we first investigate the TCP out-of-order problem of RPS mechanism. Then we show the drawbacks of flow-level and flowlet-level load balancing mechanisms. Finally, we present the key insights underlying our design.

A. Load balancing scheme with different granularities

Existing load balancing designs make switching decisions at different levels. Specifically, packet-level load balancing mechanism RPS simply sprays each packet to all available shortest paths. ECMP randomly hashes flows to paths at per-flow level. The switching granularities of CONGA and LetFlow are flowlets, which are bursts of packets from a flow that are separated by large enough gaps to avoid packet reordering.

We illustrate the working mechanisms of RPS, ECMP and LetFlow in Fig.1. As shown in Fig.1 (a), 3 equal cost paths are provided for 2 flows. Each output port of the leaf switch has a buffer queue. Fig.1 (b) shows that the packets are randomly scattered to the output port queues in RPS. For flow 2, packet 1 will leave the leaf switch later than packet 2, 3, 4 and 5, triggering the reduction of TCP congestion window as DupAckThreshold is set to 3. As shown in Fig.1 (c), since all the packets of each flow are hashed to one output port by ECMP, flow 1 and flow 2 may be transmitted through the same path, leaving the other 2 paths idle. Fig.1 (d) shows that, under LetFlow, flow 1 is rerouted only when the congestion state of its path becomes very heavy, resulting in low utilization of the other paths. In ECMP and LetFlow, a significant fraction of links may experience congestion even while there are underutilized links elsewhere.

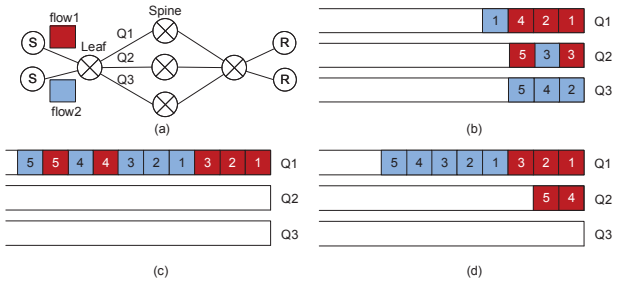


Fig. 1: Illustration example: (a) leaf-spine topology settings with three equal cost paths; (b) RPS; (c) ECMP; (d) LetFlow.

B. Out-of-order problems in RPS

We conduct the NS2 simulation to investigate the impact of out-of-order problem of RPS. The test topology is leaf-spine as shown in Fig.2. Each server sends a flow to a receiver via multiple switches. The buffer size of each switch is set to 256 packets. The bandwidth of each path is 1Gbps and the round trip propagation delay is 100μs.

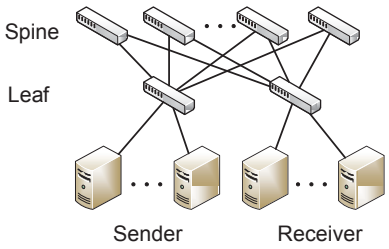


Fig. 2: Leaf-Spine topology

We generate 50 flows (from 50KB to 200KB) in heavy-tailed distribution. Fig.3 shows the proportion of reordered and dropped packets for each flow. The percent of reordered packets is about 30% on average, while the packet loss rate is almost zero. The reason is that, even when the traffic load is not high, RPS randomly spreads the packets of each flow to all paths, unavoidably resulting in out-of-order problem.

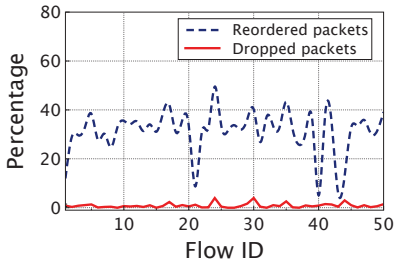
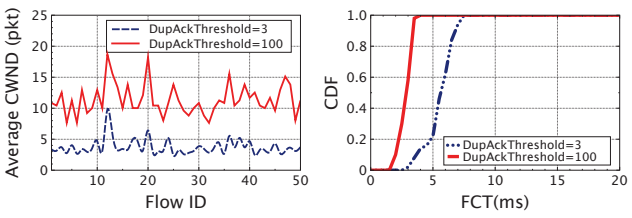


Fig. 3: The percent of reordered and dropped packets

In order to test the impact of out-of-order event on the network performances, we compare the simulation results of RPS with DupAckThreshold of 3 and 100. As shown in Fig.4 (a), the average congestion windows of each flow are larger when DupAckThreshold is 100, because we eliminate the effects of reordered packets on the congestion window. In Fig.4 (b), when the DupAckThreshold is 100, the flow completion time is greatly reduced, showing that the out-of-order problem results in low sending rate and large FCT.

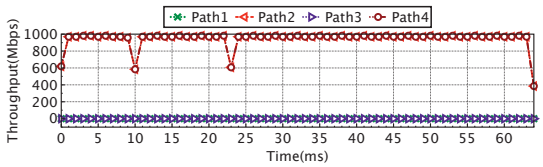


(a) The average congestion window (b) FCT of all flows

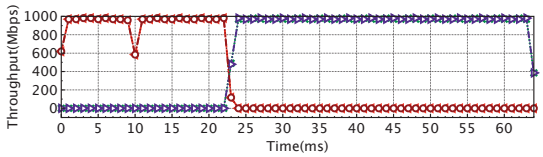
Fig. 4: The effects of packet reordering

C. Low link utilization in ECMP and LetFlow

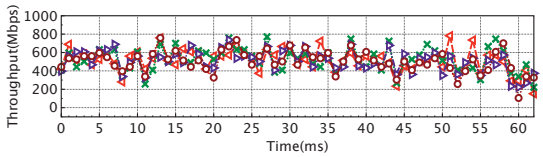
In this test, we analyze the under-utilization problem of ECMP and LetFlow in the leaf-spine topology shown in Fig.2. We generate 2 flows through 4 equal cost paths. The flowlet timeout is set as 500μs as recommended in [7].



(a) ECMP



(b) LetFlow



(c) RPS

Fig. 5: Throughput on each path with different schemes

We measure the link utilizations of 4 paths. As shown in Fig.5 (a), when ECMP is used, 2 flows are respectively hashed to path 2 and path 4, which are almost fully utilized. However, the throughputs of the other two paths are always zero, leading to only 50% utilization ratio of all paths. The result of LetFlow is shown in Fig.5 (b). At beginning, flow 1 and flow 2 are transmitted through path 2 and path 4, respectively. When the time interval between two consecutive packets of a flow arriving at the leaf switch is larger than the threshold of flowlet timeout, flow 1 and flow 2 are rerouted to path 1 and path 3, respectively. Though LetFlow transmits the flows through all paths, it still leads to low link utilization due to its inflexibility. As shown in Fig.5 (c), as a packet-level load balancer, RPS uses all paths at the same time and obtains similar throughputs on all paths. Because of the out-of-order problem, however, RPS is not able to achieve the full utilization on each path.

D. Summary

Our analysis of the existing load balancing schemes leads us to conclude that (i) packet-level load balancer RPS obtains high link utilization, but leads to TCP out-of-order problem; (ii) ECMP and LetFlow are not able to make full use of all paths because of their coarse granularity in rerouting flows. These conclusions motivate us to propose a queueing delay aware packet-level load balancing mechanism to avoid packet reordering and obtain high link utilization. We present the design and implementation of QDAPS in the rest of the paper.

III. QDAPS Design

In this section, we describe the overview of QDAPS and elaborate the design and implementation details.

A. QDAPS Architecture

Our goal is to design a packet-level load balancing mechanism QDAPS to resolve packets reordering and achieve high link utilization. Specifically, on the one hand, QDAPS splits individual flows into packets and reroutes each packets on fine-granularity to provide better load balance and high link utilization. On the other hand, QDAPS elaborately selects the output ports for each arriving packets in a single flow in order to mitigate packet reordering. Fig.6 shows the architecture of QDAPS, which is implemented on switch and has three main modules.

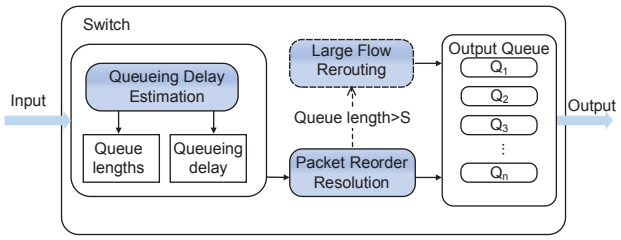


Fig. 6: QDAPS Overview

(1) **Queueing Delay Estimation:** When a new packet arrives, QDAPS estimates the queueing delay of the packet according to the real-time queue length of its assigned output ports. For each flow, QDAPS only records the queueing delay of the last arrival packet.

(2) **Packet Reorder Resolution:** To avoid packet reordering, the earlier arrival packets should be sent by switch ahead of the later arrival ones in the same flow. To achieve this target, for each arriving packet, QDAPS selects a destined output port with larger queueing delay than the last arrival packet in the same flow.

(3) **Large Flow Rerouting:** Since QDAPS lets the later arrival packets wait longer than the earlier arrival ones in a same flow, the packets of large flow may experience large queueing delay. To solve this problem, when the current queue length is larger than a given threshold, QDAPS reroutes the packets to the output port with shortest queue length.

B. Packet Reorder Resolution

When the first packet of a flow arrives at the switch, QDAPS selects the output queue with the shortest queue length for this packet. For later arrival packet, QDAPS selects a output queue to ensure that its queueing delay is larger than the remaining delay of the last arrival packet in the same flow.

As a simple example shown in Fig.7, since the queueing delay of output port Q1 is larger than the remaining queueing delay of packet 3, QDAPS selects Q1 for packet 4 to ensure that packet 4 leaves the switch later than packet 3, avoiding packet reordering.

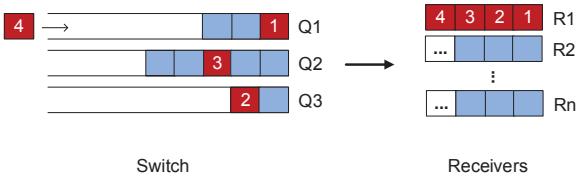


Fig. 7: Packet scheduling method in switch with QDAPS

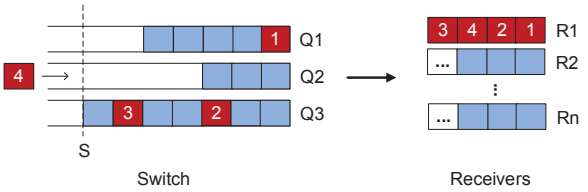


Fig. 8: long queue length problem in QDAPS

Here we present QDAPS's algorithm for scheduling packets on each switch in a leaf-spine topology. We assume that each switch has a forwarding table, which stores the set of candidate next-hops for each destination host. QDAPS is a switch-local scheduling algorithm operating as follows.

When a packet p_i in flow f arrives at time t_i, the queueing delay QD_i is calculated as

$$QD_i = \frac{M \times (l_i + 1)}{C}, \tag{1}$$

where M is the packet size, l_i is the queue length of p_i's destined output port, and C is the link bandwidth corresponding to the output port.

When the following packet p_{i+1} in flow f arrival at time t_{i+1}, the remaining queueing delay RQD_i of p_i is

$$RQD_i = QD_i + t_i - t_{i+1}. \tag{2}$$

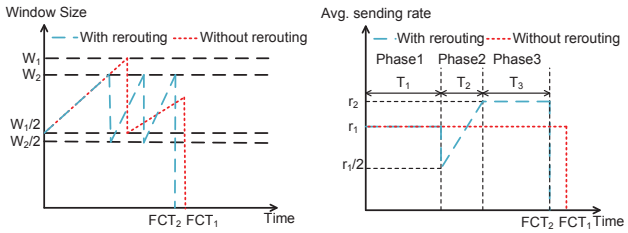
Then, the forwarding engine of switch chooses d output ports with larger queueing delay than RQD_i, finds the one with the current minimum queue length between these d ports, and routes packet p_{i+1} to that port. For each flow, the engine only updates the arriving time t_i and queueing delay QD_i of last arrival packet p_i.

C. Large Flow Rerouting

The data center traffic is heavy-tailed distributed [9], [10], that is, 10% of TCP flows provide around 90% of data traffic, and about 90% of TCP flows transfer only about 10% of data traffic. QDAPS routes the first packet of a flow to the output queue with the shortest queue length. The small flows are usually completed quickly (i.e., even in slow start phase), and will not experience long queueing delay.

However, we find that, the packets of large flows may accumulate on a few queues with large queueing lengths, because QDAPS routes the later arrival packets to output queues with lager queueing delay to resolve the packet reordering. This problem is illustrated in Fig.8. When packet 4 arrives, if QDAPS chooses Q3 for packet 4 to avoid packet reordering, the queueing delay becomes large. Therefore, if the queue length is larger than a given threshold S, we propose to reselect a shortest output queue (i.e., Q2) for packet 4 to reduce the queueing delay.

Fig.9 shows the cost-benefit assessment QDAPS performs when making rerouting decisions. On the one hand, considering a flow that is sending data, if it persists in the same output queue, the large queueing delay leads to the slow increasing speed in congestion window and low link utilization. On the other hand, rerouting to a less congested output path obtains low queueing delay and high link utilization, while may result in packet reordering and rate reduction. In the following, we seek a good tradeoff between queueing delay and packet reordering by rerouting the large flows when its queue length exceeds a threshold S.



(a) Congestion window size (b) Average sending rate

Fig. 9: Analysis of the cost and benefit

Given the fact that DCTCP [11] has been extensively deployed in many data centers, such as Google [11] and Morgan Stanley [12], we choose DCTCP as the transport protocol to analyze the queue length threshold S in rerouting long flow. We use Y, RTT, C and K to denote the flow size, the propagation delay, the bottleneck link bandwidth and the ECN marking threshold in DCTCP, respectively. Fig.9 compares the congestion windows of a single flow with and without rerouting.

If the flow is not rerouted, its packets are accumulated in an output queue at the switch and the queue length is maintained at around K by DCTCP. Then, the maximum congestion window W_1 is capped by the the total number of packets that can be accommodated in link pipeline and switch buffer, that is, W_1 is calculated as

$$W_1 = C \times RTT + K. \tag{3}$$

When the congestion window grows from $\frac{W_1}{2}$ to W_1, it takes $\frac{W_1}{2}$ RTTs. During each round, the average congestion window and queueing delay are $\frac{3W_1}{4}$ and $\frac{K}{C}$, respectively. Then we get the average sending rate r_1 as

$$r_1 = \frac{\frac{3W_1}{4}}{\frac{K}{C} + RTT}. \tag{4}$$

If the flow does not change its path during the transmission, the flow completion time FCT_1 is calculated as

$$FCT_1 = \frac{Y}{r_1} = \frac{4Y \times (\frac{K}{C} + RTT)}{3W_1}, \tag{5}$$

where Y is the flow size in packets.

If the flow is rerouted to the output port with shortest queue length when the queue length exceeds S, the maximum congestion window W_2 is

$$W_2 = C \times RTT + S. \tag{6}$$

The flow completion time under rerouting includes three phases as shown in Fig.9 (b). In first phase, the flow is sent at the rate r_1. The second phase starts when the average sending rate is dropped from r_1 to $r_1/2$ by rerouting and ends once the average rate increases to r_2, which is the average rate in the destined path. In the last phase, the flow is sent at rate r_2.

Phase1: In this phase, the queue length is less than S. As the congestion window increases from 1 to W_2, it takes W_2 RTTs, and the average window is $W_2/2$. Then the data traffic size S_1 sent in this phase is

$$S_1 = \frac{W_2}{2} \times W_2. \tag{7}$$

The time T_1 in this phase is

$$T_1 = \frac{S_1}{r_1}. \tag{8}$$

Phase2: The average sending rate decreases from r_1 to $r_1/2$ because of rerouting and then increases to r_2, which is the average sending rate after rerouting. Note that if we reroute the flow when the queue length exceeds S, the average queueing delay is $S/2C$ as the queue length increases from 0 to S. Then, r_2 is calculated as

$$r_2 = \frac{\frac{3W_2}{4}}{\frac{S}{2C} + RTT}. \tag{9}$$

In this phase, the congestion window increases from $W_2/2$ to W_2. The data traffic size S_2 sent in this phase is

$$S_2 = (W_2 - \frac{W_2}{2}) \times \frac{W_2 + \frac{W_2}{2}}{2}. \tag{10}$$

The average sending rate in this phase is $(r_1/2 + r_2)/2$. We get the time T_2 in this phase as

$$T_2 = \frac{S_2}{\frac{\frac{r_1}{2} + r_2}{2}}. \tag{11}$$

Phase3: In this phase, the remaining data traffic of the flow is sent at the average rate r_2. We get the time T_3 in this phase as

$$T_3 = \frac{Y - S_1 - S_2}{r_2}. \tag{12}$$

With rerouting, the flow completion time FCT_2 is the sum of T_1, T_2 and T_3. According to Equation (8), (11) and (12), we obtain FCT_2 is

$$FCT_2 = \frac{S_1}{r_1} + \frac{S_2}{\frac{\frac{r_1}{2} + r_2}{2}} + \frac{Y - S_1 - S_2}{r_2}. \tag{13}$$

In order to reduce the flow completion time, the queue length threshold S is set as the minimum value satisfying $FCT_1 \geq FCT_2$.

Finally, we obtain the value of S as

$$S \approx \sqrt{\frac{12YC - 8Yr_1}{6C - 3r_1}} - C \times RTT. \tag{14}$$

Note that flow rerouting is designed for long flows. Typically, there are few large flows in data center applications compared with the large number of small flows. Moreover, the large flow rerouting is triggered only when the queue length exceeds the threshold S. Since the large flow rerouting is not triggered very frequently, its impact on the normal operations of switch is limited.

D. Implementation

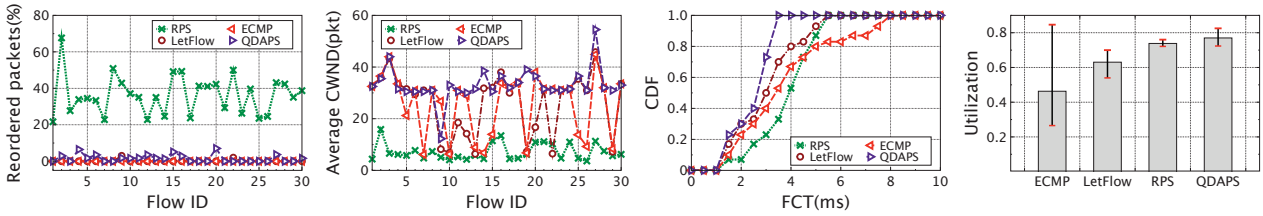
We have implemented QDAPS in BMv2 and deployed it in a small testbed. BMv2 is a P4 [13] software switch in which we test the P4 implementation of QDAPS. Note that our QDAPS prototype is deployed on switch, without any modifications on TCP/IP protocol stack of end-hosts. In the QDAPS's P4 implementation, we mainly take two key points into consideration.

Flow Table: The packets enter the ingress and egress pipeline at the switch, which maintains a flow table to record the flow information including the flow ID, flow size, flow age bit and remaining queueing delay of the last arriving packet in the corresponding flow.

Specifically, when a packet arrives at the switch and enters the ingress pipeline, QDAPS acquires the 5-tuple of the packet header and adopts CRC16 algorithm to calculate the hash value used as the flow ID. The flow size is measured as the amount of data that a flow has already sent through the switch. The remaining queueing delay of the arrival packet is computed according to the queue length of the packet's destined output port. To avoid unnecessary storage wastage in flow table due to idle connections, QDAPS samples the flows periodically and removes the idle connections from the flow table. Upon the arrival of a packet, the corresponding flow's age bit is set to 1. At a fixed time interval, $2RTT$, QDAPS removes the flow with age bit as 0 and clears the age bits of the other flows. QDAPS uses 8 bytes to record the information of each flow. Since the number of active flows is less than 10,000 on a leaf switch [11], the deployment overhead is negligible.

Leaf-to-leaf Delay Estimation: As a localized load balancing scheme, QDAPS makes forwarding decisions according to the queue lengths of the output ports in the leaf switches. However, the local congestion state on the leaf switch is not always consistent with the global or leaf-to-leaf information about congestion. In our implementation, we design an enhanced version called QDAPS*, which gathers congestion feedback from remote leafs to make forwarding decisions.

We modify the egress pipeline at the leaf switches to record the sending time. When a packet is sent by the port p_s of source leaf switch, the sending time is filled in the timestamp field of packet header. When receiving the packet at a port p_d, the destination leaf switch fetches the sending time in the packet header and calculates the leaf-to-leaf delay by subtracting the sending time from the arriving time. To quickly feed the path delay to the source leaf switch without any traffic



(a) The percent of reordered packets (b) The average congestion window (c) The average FCT of flows (d) The utilization ratio of network

Fig. 10: The basic simulation performance with different load balancing mechanisms

overhead, the destination leaf switch piggybacks the leaf-to-leaf delay on the timestamp field of any data or ACK packet sent though port p_d to port p_s of source leaf switch. Finally, the source leaf switch obtains the delay from p_s to p_d by reading the timestamp field of packets arriving port p_s, and makes the forwarding decisions based on the sum of leaf-to-leaf delay and local queueing delay. The test results in section IV.D show that QDAPS* is resilient enough to mitigate the packet reordering under the impact of global congestion.

It should be noted that, to reduce the feedback delay, QDAPS* uses the data or ACK packets at the head of the queue to piggyback the leaf-to-leaf delay. Moreover, QDAPS* employs all available packets in reverse direction to piggyback the delay information to ensure high reliability. Therefore, QDAPS* utilizes the data or ACK packets to obtain the real-time leaf-to-leaf delay without any extra traffic overhead.

IV. Simulation Evaluation

We conduct NS2 simulations to evaluate the performance of QDAPS in the large-scale scenarios. In the tests, DCTCP is adopted as the transport protocol. The buffer size of switches and buffer occupancy threshold K are 256 and 65 packets, respectively. We compare QDAPS with the following three state-of-the-art load balancing schemes.

(1) **ECMP:** As a standard multipath load balancing mechanism, ECMP is widely deployed on commodity switches in data center networks. It hashes each flow to an available shortest path to the destination based on the five-tuple of TCP packet header.

(2) **LetFlow:** LetFlow is a flowlet-level load balancing algorithm which has been implemented in silicon for data center switches. A flowlet is formed when the gap between the arrival time of two consecutive packets of a flow is larger than the threshold, which is 500μs to avoid packet reordering.

(3) **RPS:** RPS is a simple packet-level approach which randomly spreads packets to all available equal cost paths between any pairs of source and destination hosts.

A. Basic Performance

In this test, we compare the percentage of packet reordering, average congestion window of TCP flows and network utilization in different load balancing approaches. The test topology is a leaf-spine network with 4 equal cost paths shown in Fig.2. We generate 30 flows from 50KB to 200KB in heavy-tailed distribution and the start time of these flows follows the Poisson distribution. The round trip propagation delay is 100μs and the link bandwidth is 1Gbps.

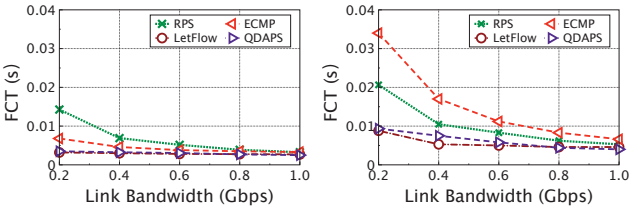
As shown in Fig.10 (a), the percentage of reordered packets in RPS is as large as about 40%. The reason is that RPS randomly selects the output ports for each packet, resulting in lots of reordered packets. ECMP avoids packet reordering because it spreads all the packets of each flow through a single path to the destination. In LetFlow, the percentage of reordered packets is very small as the flowlet timeout is large enough to trigger less flow rerouting. QDAPS experiences few packet reordering as it schedules packets according to queueing delay.

As shown in Fig.10 (b), the average congestion window in RPS is much less than the other load balancing mechanisms due to its heavy packet reordering. Since ECMP adopts a static hash method and does not sense the congestion state, the hash collision leads to the packet loss and reduction of congestion window. QDAPS obtains larger congestion window than the other schemes, because it is able to perceive the queueing length of output ports to forward packets, achieving the in-order delivery and packet-level flexibility in balancing traffic.

Fig.10 (c) shows the CDF of flow completion time. QDAPS effectively reduces the FCT with the following two reasons. On the one hand, QDAPS forwards the packet based on the remaining queueing delay of the last arriving packet at the switch to avoid packet reordering. On the other hand, as shown in Fig.10 (d), since QDAPS is a packet-level transmission scheme, it is more flexible to switch packets than flow-based scheme and achieves the higher link utilization. We show the maximum, minimum and average link utilization ratios in Fig.10 (d). The experiments are repeated for 10 times. Since ECMP is a static load balancing mechanism, the average link utilization of all paths in ECMP is less than 50%, that is, some paths are congested while some ones are unutilized. Since LetFlow scheme uses fixed gap between two consecutive packets to trigger rerouting flows, the link utilization is reduced due to its inflexibility. Though RPS scatters packets to all paths, the packet reordering degrades the sending rate. Therefore, QDAPS obtains the largest link utilization by making full use of all parallel paths and avoiding out-of-order packets.

B. Asymmetric Scenario

In real data center networks, the traffic dynamics, device heterogeneity and switch malfunctions easily lead to asymmetric scenario, which greatly hurts the performances of load balancing. Here, we test if QDAPS can appropriately split traffic



(a) The average FCT (b) The 99th percentile FCT

Fig. 11: The FCT performance in asymmetric topology

among multiple paths in reaction to path conditions under an asymmetric topology. We adopt the leaf-spine topology with 8 paths between two leaf switches and vary the bandwidth of one randomly selected leaf-to-spine link from 200Mbps to 1Gbps. The bandwidth of other links is 1Gbps. We generate 50 flows (from 50KB to 200KB) in heavy-tailed distribution. We measure the average and 99th percentile FCT of all flows.

Fig.11 (a) shows that the average FCT of for RPS rises quickly when the bandwidth difference is large. For example, the average FCT with the degraded link as 200Mbps is $\sim$5$\times$ of that in 1Gbps. The reason is that RPS does not sense the link congestion and randomly spreads packets to all available paths. Under the asymmetric topology, RPS experiences heavy packet reordering and rate reduction. ECMP is also a congestion-oblivious load balancing scheme. When the flows are hashed to a congested path, ECMP does not reroute these unlucky flows. Since QDAPS and LetFlow sense the congestion state and adaptively route the flows to uncongested paths, they achieve low average FCT.

Fig.11 (b) shows the 99th percentile FCT, presenting the tail delay of all flows. When the bandwidth of degraded link is 200Mbps, QDAPS reduces the 99th percentile FCT by $\sim$80% and $\sim$70% over ECMP and RPS, respectively. In brief, QDAPS perceives the queue differences of all output ports before making the routing decisions and thus achieves good performance in the asymmetric networks.

C. Large-Scale Application Performances

We further compare the performance of QDAPS with the state-of-the-art load balancing mechanisms through a large-scale test. In the test, we adopt the workload of web search [11] and data mining [1], which are the typical applications in real data center networks. The distribution of the flow size in different workload scenarios are shown in Table I. The average flow sizes are 1.6MB and 7.4MB in the web search and data mining applications, respectively.

TABLE I: Flow size distribution of realistic workload.

	0-10KB	10KB-100KB	100KB-1MB	>1MB
Data Mining	77%	5%	8%	10%
Web Search	55%	7%	18%	20%

We use a 8×8 leaf-spine topology with 8 equal cost paths between any pair of hosts in the simulation. Each leaf switch connects to 32 hosts and the oversubscription ratio is 4:1. The bandwidth of links is 1Gbps and the round trip propagation delay is 100μs. We evaluate the performance of QDAPS with the load varying from 0.1 to 0.8.

The flow completion time is the primary metric in our tests because the delay performance is very important for online applications which focus on the user experience. For short flows ($\leq$100KB), we compare the average and 99th percentile FCT. We also compare the average FCT of long flows ($\geq$1MB) and all flows to provide throughout understanding. We show the simulation results of web search workload and data mining workload in Fig.12 and Fig.13, respectively.

As shown in Fig.12 (a) and Fig.13 (a), under light load, RPS and ECMP perform well and the FCT of all flows is close to that of LetFlow and QDAPS. But under heavy load, QDAPS performs much better because it spreads packets according to the queueing delay to achieve less packet reordering. Specifically, in web search workload, QDAPS reduces the FCT of all flows by $\sim$50%, $\sim$30% and $\sim$40% over ECMP, LetFlow and RPS at 0.8 network load, respectively. In data mining workload, QDAPS reduces the FCT of all flows by $\sim$50%, $\sim$25% and $\sim$45% over ECMP, LetFlow and RPS at 0.8 network load, respectively.

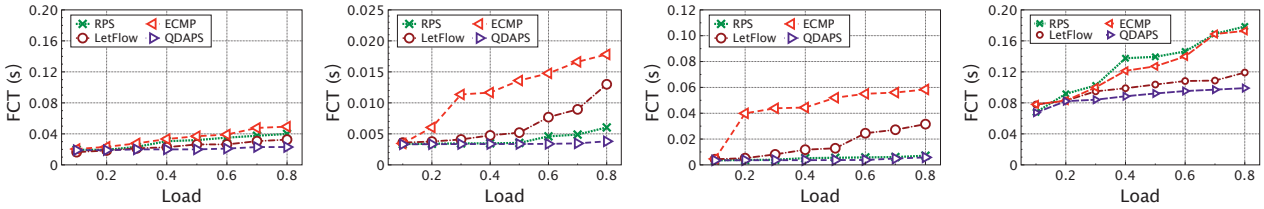
As shown in Fig.12 (b) and Fig.13 (b), for short flows, RPS and QDAPS perform better than ECMP and LetFlow. Since the short flows experience less packet reordering, RPS obtains low delay for short flows. Due to many hash collisions, FCT of short flows in ECMP is very large. Fig.12 (c) and Fig.13 (c) show the 99th percentile FCT of short flows to present the tail FCT. QDAPS performs well especially under heavy loads. For example, in web search workload, QDAPS reduces the 99th percentile FCT by $\sim$90%, $\sim$80% and $\sim$15% over ECMP, LetFlow and RPS at 0.8 network load, respectively.

Fig.12 (d) and Fig.13 (d) show the FCT of long flows. RPS spreads packets to all available paths, results in many reordered packets and large FCT. ECMP and LetFlow easily degrade the link utilization due to their inflexibility. Under the light and heavy loads, QDAPS always obtains better performance compared with the other schemes.

D. QDAPS* Performance

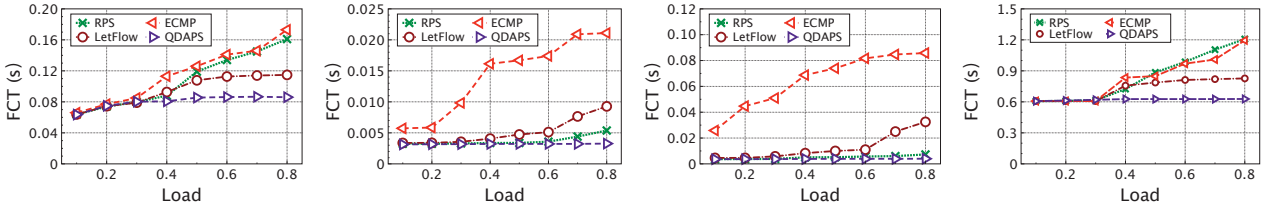
In this test, we evaluate the performance of QDAPS* under global congestion. We use a leaf-spine topology with 4 equal cost paths and 30 flows. The propagation delay of a congested path is varying from 100μs to 300μs and that of the other paths is 100μs. We generate the mixture of long and short flows (from 50KB to 500KB) in uniform distribution and the flow arrival time follows the Poisson distribution. The bandwidth of all links is set to 1Gbps.

Fig.14 (a) shows the average FCT of all the flows. When the propagation delay of congested path increases, all schemes experience larger average FCT. Since RPS does not sense the congestion on the degraded path, it causes more packet reordering and much larger FCT than the other schemes. As a congestion-unaware load balancing mechanism, ECMP



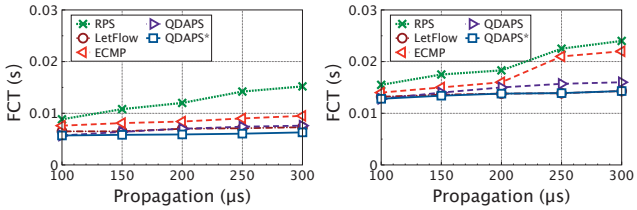
(a) Overall avg FCT (b) Short flows ($\leq$ 100KB) avg (c) Short flow 99th percentile (d) Long flows ($\geq$ 1MB) avg

Fig. 12: FCT statistics for web search workload



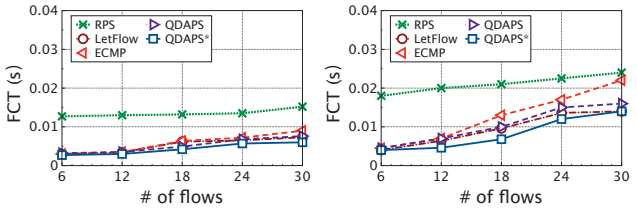
(a) Overall avg FCT (b) Short flows ($\leq$ 100KB) avg (c) Short flow 99th percentile (d) Long flows ($\geq$ 1MB) avg

Fig. 13: FCT statistics for data mining workload



(a) The average FCT (b) The 99th percentile FCT

Fig. 14: QDAPS* performance under different delay



(a) The average FCT (b) The 99th percentile FCT

Fig. 15: QDAPS* performance under different number of flows

will not reroute flow even though serious congestion is experienced. Though LetFlow is aware of global congestion, it randomly reroutes flow once senses the path congestion and therefore is not able to achieve the optimal performance. QDAPS* reduces the average FCT by up to ∼35%, ∼15%, and ∼60% over ECMP, LetFlow and RPS, respectively. Fig.14 (b) shows the 99th percentile FCT of all flows. RPS performance is the worst in the test. Compared with QDAPS, QDAPS* detects the leaf-to-leaf delay and makes more accurate decisions to resolve packet reordering.

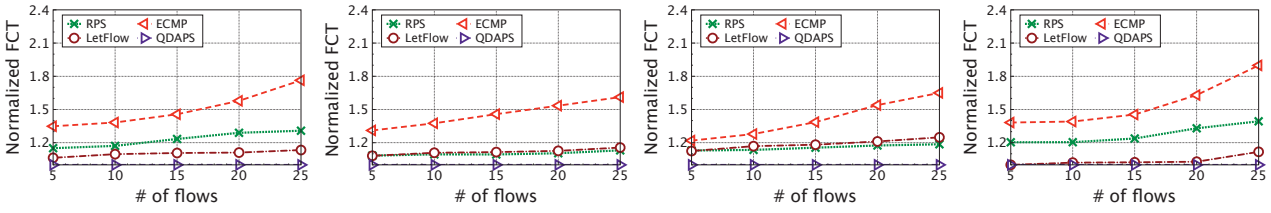
Next, we test the delay performance under different network loads. The propagation delay of a congested path is 300μs, while that of the other paths are 100μs. We change the number of flows and show the delay results of different load balancing mechanisms in Fig.15. When the number of flows increases, the delay performances are degraded in all schemes. RPS's delay is always very large due to the packet reordering problem. QDAPS* achieves the best delay performance in both average and 99th percentile FCT.

V. Testbed Evaluation

In this section, we implement QDAPS with P4, which is a high-level language for programming protocol-independent packet processors. We test the performance of QDAPS through a realistic testbed in Mininet [14], [17], [18]. Mininet is a network emulation system, which creates virtual hosts, switches, links and controller on the standard Linux kernel. Most of the behaviors in Mininet are similar to the real network elements [19]. However, the test scale in Mininet is smaller than real data center networks due to the limitation of CPU.

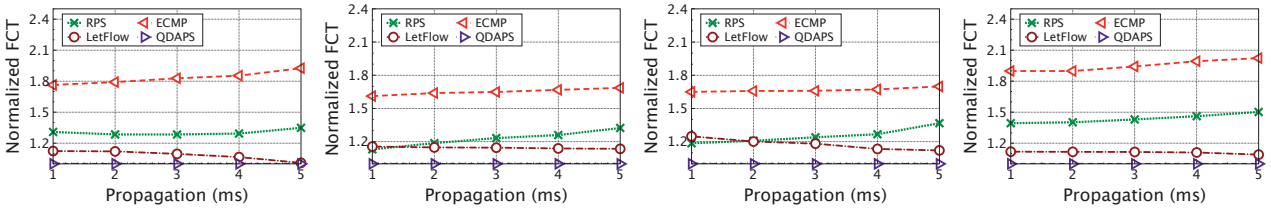
In the test, we implement the packet processing pipeline of QDAPS with $P4_{16}$ v1.0. We use Mininet 2.3.0d1 to create a leaf-spine topology as shown in Fig.2. There are 4 equal cost paths between the leaf and spine switch. BMv2 is installed as the software programmable switch. The link bandwidth is set as 20Mbps as recommended in [14]. We set the round trip propagation delay to 1ms and the buffer size at switches to 256 packets [17]. We compare the performance of QDAPS with ECMP [5], LetFlow [7] and RPS [3]. The flowlet timeout in LetFlow is set to 15ms. The average flow size is 64KB [16] and all flows are less than 1MB.

We firstly test the delay performance under different number of flows that follow the distribution like Web Server [15]. As shown in Fig.16, we normalize the FCT of ECMP, RPS and LetFlow to that of QDAPS. In Fig.16 (a), the normalized



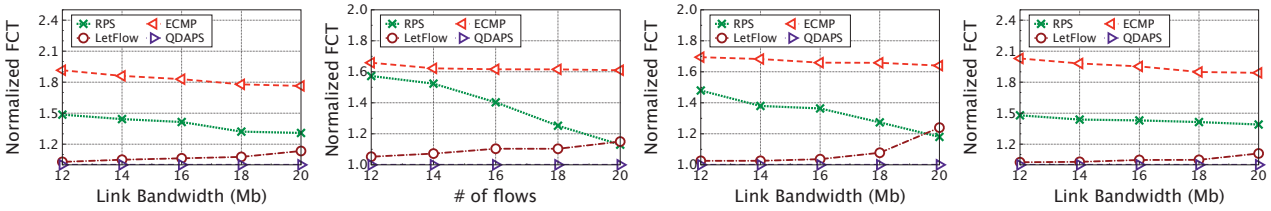
(a) Overall avg FCT (b) Short flows ($\leq$ 100KB) avg (c) Short flow 99th percentile (d) Long flows ($\geq$ 100KB) avg

Fig. 16: Performance with different number of flows in symmetric scenario



(a) Overall avg FCT (b) Short flows ($\leq$ 100KB) avg (c) Short flow 99th percentile (d) Long flows ($\geq$ 100KB) avg

Fig. 17: Performance with varying propagation delay in asymmetric scenario



(a) Overall avg FCT (b) Short flows ($\leq$ 100KB) avg (c) Short flow 99th percentile (d) Long flows ($\geq$ 100KB) avg

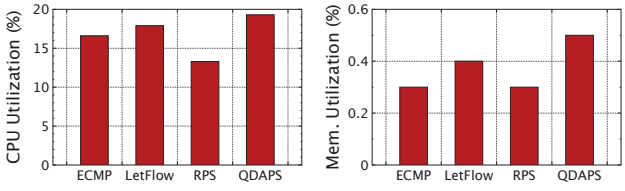
Fig. 18: Performance with varying bandwidth in asymmetric scenario

overall average FCT of ECMP, RPS and LetFlow is large than 1, meaning that QDAPS achieves the shortest average FCT of all flows. Specifically, QDAPS reduces the average FCT of all flows by ∼25%-50%, ∼10%-20% and ∼15%-25% over ECMP, LetFlow and RPS, respectively. Due to heavy hash collisions, ECMP experiences much larger FCT with the increasing number of flows.

Fig.16 (b) and Fig.16 (c) show the performances of short flows. QDAPS reduces the average and tail FCT thanks for its flexibility in load balancing mechanism and delay-awareness to avoid packet reordering. As shown in Fig.16 (d), QDAPS reduces the FCT of long flows by mitigating the packet reordering and reducing queueing delay. QDAPS reduces the average FCT of long flows by ∼35%-45%, ∼2%-10% and ∼20%-40% over ECMP, LetFlow and RPS, respectively.

Next, we further explore the performance of QDAPS in the asymmetric scenarios. We set the number of flows as 25. We vary the propagation of one randomly selected path from 1ms to 5ms to produce delay asymmetry. We also change the bandwidth of one of the output ports from 12Mb to 20Mb to make bandwidth asymmetry.

The results shown in Fig.17 demonstrate that QDAPS achieves better performance than ECMP and RPS schemes when the propagation delay is different between the parallel paths. Fig.18 presents the similar trend of the FCT under bandwidth asymmetry. For RPS, when the delay or bandwidth asymmetrical extent becomes higher, more packets are out-of-order, leading to more spurious retransmission and even timeout. For ECMP, once a flow is hashed to the path with large delay or low bandwidth, the flow will be transferred on the bad path all the time instead of being rerouted to other good paths. Since conducting load balancing according to the network congestion, QDAPS and LetFlow perform well under the delay and bandwidth asymmetry.



(a) CPU utilization (b) Memory utilization

Fig. 19: Overhead of the leaf switch with different schemes

QDAPS is implemented on switch with computing overhead. We measure the CPU and memory utilization ratios to evaluate the overhead. The experimental scenario involves 25 flows. Fig.19(a) shows the CPU utilization of the leaf switch with different schemes. Since RPS simply sprays all the packets to all the equal cost paths, its CPU utilization is the lowest. QDAPS does not incur excessive CPU overhead to switch compared with other schemes, because the computing overhead only generates a tiny fraction of CPU load. From Fig.19(b), we observe that QDAPS brings about 0.5% memory utilization of the leaf switch. Compared with the gain in the delay performance, the system overhead is acceptable.

VI. Related Works

Nowadays, the load balancing mechanisms in data center network are roughly divided into the following four types: flow-level, flowlet-level, flowcell-level and packet-level mechanisms. We demonstrate existing approaches and discuss their pros and cons in the section.

ECMP [5] is a standard flow-level load balancing mechanism used in data center networks. The switches use the five-tuple in packet header for hash calculations. Based on the hash result, an outgoing port is selected for a flow. This static method works well for short flows. However, when some long flows are hashed to the same path, the hash collision leads to the persist congestion on a few paths and low link utilization on the other paths. Moreover, the performance of ECMP is degraded greatly in asymmetric topology as it does not sense the congetion. According to the path states, WCMP [20] adds weights to ECMP to obtain better load balancing. When sensing the path congestion through the congestion signals like ECN, FlowBender [21] randomly reroutes flows to another path. However, the random approach is very hard to achieve the optimal performance.

MPTCP [22] divides a TCP flow into multiple subflows to make use of the multiple transmission paths. Each subflow maintains its congestion window to achieve load balancing on multiple paths and improve network utilization. However, MPTCP needs to modify the host stack, which is not suitable for deployment in a multi-tenant environment. Moreover, MPTCP does not perform well in incast scenarios as many subflows are transfered simultaneously .

Other proposals like Hedera [23] and MicroTE [24] are centralized load balancing mechanisms. Hedera adaptively schedules long flows to uncongested paths with global knowledge of traffic to avoid collisions. But the network utilization is still low since Hedera adopts flow-level scheduling method which is not flexible. Similar to Hedera, MicroTE schedules flows by leveraging the traffic predictability to achieve load balancing through OpenFlow switches.

CONGA [6], LetFlow [7], HULA [25] and CLOVE [26] are flowlet-level load balancing approaches, which reroute the flowlet to get high link utilization and low packet reordering. CONGA and HULA use the path utilization information to reroute the flowlets. On the switch, LetFlow randomly spreads each flowlet to one available shortest path. However, the threshold set for a flowlet gap directly affects the performances of these approaches in highly dynamic network. On the one hand, when the threshold is too large, these approaches are not flexible enough to utilize the idle paths, resulting in suboptimal performance. On the other hand, when the threshold is too small, the flowlet gaps and path reselection operations are likely to occur, easily resulting in the out-of-order problem.

Presto [27] is a flowcell-level load balancing proposal. Presto is implemented in the network edge, such as Open vSwitch, to split a flow into many units with fixed size (i.e., 64KB) and spread the units to all paths. Presto deals with packet reordering problem with enhanced Generic Receive Offload (GRO) at the receiver side. Presto is a congestion-oblivious scheme which is not able to sense and avoid the congested paths.

SRR [28] is a multipath transmission scheme based on quantum of service, which is proportional to the path bandwidth. The receiver uses Causal Fair Queueing (CFQ) to reorder the packets from different paths. However, it does not work well in asymmetric scenario as it can not sense the network congestion.

RPS [3] adopts a simple packet-level multipath transmission scheme to achieve good load balancing. However, RPS causes the out-of-order problem due to the negative interaction of different flows and the random manner in path selection. To mitigate the packet reordering of RPS [3], DRB [29] sprays packets to multiple paths in a round robin way. Detail [30] proposes an adaptive packet-level mechanism which can deal with the asymmetry network, but needs the complex modifications of the network stack. SAPS [31] is a SDN-based scheme which divides the topology into several symmetric virtual ones to improve the performance of RPS in the asymmetric topologies.

Hermes [8] is resilient to the network uncertainties such as traffic dynamics and switch failures. It is an edge-based load balancing solution which makes timely yet cautious rerouting decisions. To mitigate the impact of congestion mismatch, only when there is benefit, Hermes changes the transmission path for packets instead of vigorous rerouting to reduce packet reordering and FCT. DRILL [32] uses a packet-based load balancing mechanism which is based on the information of local switch. Its main objective is to solve the problem of micro bursts load balancing with a method similar to power of two choices paradigm [33]. When DRILL selects the path for a packet arriving at the switch, the destined path will be selected from the best path in last round and the two paths randomly selected in this round.

Our QDAPS is a packet-level load balancing scheme to achieve low packet reordering and high link utilization. QDAPS makes rerouting decision for each packet according to the queueing delay. For each TCP flow, QDAPS selects the output port with the shortest queue for the firstly arriving packet on the switches. The following packets belonging to the same flow are scheduled to the output queue according to the remaining queueing delay of the last arrival packet to avoid packet reordering. QDAPS is implemented at the switches and does not require any modifications in the existing TCP/IP protocol stack.

VII. Conclusion

In this paper, we propose QDAPS, a queueing delay aware load balancing mechanism which significantly reduces the flow completion time. QDAPS selects a suitable output queue to ensure that the packets arrive at the receiver in order and avoid the low link utilization with a flexible manner. We also design a flow rerouting method to reduce the queueing delay of long flows. As a supplementary for QDAPS, we design QDAPS* which takes both global leaf-to-leaf delay and local queueing delay into consideration.

We implement QDAPS at the software switch BMv2 in P4 language. We evaluate QDAPS through the large-scale NS2 simulations and a Mininet testbed. The test results show that QDAPS effectively reduces the flow completion time by $\sim$30%-50% compared with the state-of-the-art load balancing mechanisms.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61572530, 61629302 and 61420106009), CERNET Innovation Project (Grant No.NGII20160113) and Fundamental Research Funds for Central Universities of Central South University (2018zzts066).

References

[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In Proc. ACM SIGCOMM, 2009.

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

[3] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In Proc. IEEE INFOCOM, 2013.

[4] Per packet load balancing. http://www.cisco.com/c/en/us/td/docs/ios/12_0s/ feature/guide/pplb.pdf?dtid=osscdc000283.

[5] CE Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. In RFC 2992.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In Proc. ACM SIGCOMM, 2014.

[7] V. Erico, P. Rong, A. Mohammad, T. Parvin, and E. Tom. Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In Proc. USENIX NSDI, 2017.

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

[9] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In Proc. IMC, 2010.

[10] L. Chen, K. Chen, W. Bai, et al. Scheduling mix-flows in commodity datacenters with karuna. In Proc. ACM SIGCOMM, 2016.

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

[12] G. Judd. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. in Proc. USENIX NSDI, 2015.

[13] P. Bosshart, D. Daly, G. Gibb, et al. P4:Programming Protocol-Independent Packet Processors. ACM SIGCOMM Computer Communication Review, 44(3):87-95, 2014.

[14] H. Xu, B. Li. RepFlow: Minimizing flow completion times with replicated flows in data centers. In Proc. IEEE INFOCOM, 2014.

[15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social Network's (Datacenter) Network. In Proc. ACM SIGCOMM, 2015.

[16] I. Cho, K. Jang, and D. Han. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In Proc. ACM SIGCOMM, 2017

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

[18] N Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In Proc. ACM CoNEXT, 2012.

[19] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P.B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In Proc. USENIX NSDI, 2013.

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

[21] 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.

[22] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In Proc. ACM SIGCOMM, 2011.

[23] 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.

[24] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine grained traffic engineering for data centers. In Proc. ACM CoNEXT, 2011.

[25] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In Proc. ACM Symposium on SDN Research, 2016.

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

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

[28] H. Adiseshu, G. Varghese, and G. Parulkar. An architecture for packet-striping protocols[J]. ACM Transactions on Computer Systems (TOCS), 1999, 17(4): 249-287.

[29] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet Load-balanced, Low-latency Routing for Clos-based Data Center Networks. In Proc. ACM CoNext, 2013

[30] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. Detail: Reduing the flow completion time tail in datacenter networks. In Proc. ACM SIGCOMM, 2012.

[31] S. M. Irteza, H. M. Bashir, T. Anwar, I. A. Qazi, F. R. Dogar. Load Balancing Over Symmetric Virtual Topologies. In Proc. IEEE INFOCOM, 2017.

[32] S. Ghorbani, Z. Yang, P. Godfrey, Y. Ganjali, and A. Firoozshahian. DRILL: Micro Load Balancing for Low-lantency Data Center Networks. In Proc. ACM SIGCOMM, 2017.

[33] Michanel Mizenmacher. The Power of Two Choices in Randomized Load Balancing. IEEE Tansactions on Parallel and Distributed Systems 12, 10(2001).