DOI: 10.3969/j.issn.1673-5188.2018.02.008
http://kns.cnki.net/kcms/detail/34.1294.TN.20180427.1001.002.html, published online April 27, 2018

SOPA: Source Routing Based Packet-Level Multi-Path Routing in Data Center Networks

LI Dan[1], LIN Du[1], JIANG Changlin[1], and Wang Lingqiang[2]
(1. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China;
2. Pre-Research & Standard Department, Wireline R&D Institute, ZTE Corporation, Beijing 100084, China)

Abstract

Many "rich - connected" topologies with multiple parallel paths between servers have been proposed for data center networks recently to provide high bisection bandwidth, but it remains challenging to fully utilize the high network capacity by appropriate multi-path routing algorithms. As flow-level path splitting may lead to traffic imbalance between paths due to flow size difference, packet-level path splitting attracts more attention lately, which spreads packets from flows into multiple available paths and significantly improves link utilizations. However, it may cause packet reordering, confusing the TCP congestion control algorithm and lowering the throughput of flows. In this paper, we design a novel packet-level multi-path routing scheme called SOPA, which leverages OpenFlow to perform packet-level path splitting in a round-robin fashion, and hence significantly mitigates the packet reordering problem and improves the network throughput. Moreover, SOPA leverages the topological feature of data center networks to encode a very small number of switches along the path into the packet header, resulting in very light overhead. Compared with random packet spraying (RPS), Hedera and equal-cost multi-path routing (ECMP), our simulations demonstrate that SOPA achieves 29.87%, 50.41% and 77.74% higher network throughput respectively under permutation workload, and reduces average data transfer completion time by 53.65%, 343.31% and 348.25% respectively under production workload.

Keywords

data center networks; multi-path routing; path splitting

The work was supported by the National Basic Research Program of China (973 program) under Grant No. 2014CB347800 and No. 2012CB315803, the National High-Tech R&D Program of China (863 program) under Grant No. 2013AA013303, the Natural Science Foundation of China under Grant No.61170291, No.61133006, and No.61161140454, and ZTE Industry-Academia-Research Cooperation Funds.

1 Introduction

Data center networks connect hundred of thousands of servers to support cloud computing, including both front-end online services (e.g., web search and gaming) and back-end distributed computations (e.g., distributed file system [1] and distributed data processing engine [2], [3]). Recognizing that the traditional tree-based topology cannot well embrace the bandwidth-hungry cloud services, in recent years many "rich-connected" data center network topologies have been proposed, such as Fat-Tree [4], VL2 [5], BCube [6] and FiConn [7]. These new topologies provide multiple paths between any pair of servers, and greatly increase the network bisection bandwidth. For instance, in a Fat-Tree network, there are x equal paths between two servers from different pods, where x is the number of core switches in the network; while in a $BCube(n,k)$ network, $k+1$ non-disjoint paths exist between any two servers, not to mention the paths with overlapping links.

Although the advanced data center networks enjoy high network capacity, it remains challenging how to fully utilize the capacity and provide high network throughput to upper-layer applications. Multi-path routing is necessary to exploit the abundant paths between servers. The existing multi-path routing schemes can be divided into two categories, namely, flow-level path splitting and packet-level path splitting. In flow-level path splitting solutions, traffic between two servers is split into different paths at the flow granularity. All the packets belonging to a 5-tuple flow traverse the same path, so as to avoid out-of-order delivery. For examples, equal-cost multi-path routing (ECMP) uses 5-tuple hashing to choose the path for a flow from the multiple candidates, with the possibility of hash collision and unequal utilization of the paths; in order to avoid the hash collision between large flows, Hedera [8] explores a centralized way to schedule the flows by spreading large flows into different paths. However, the flow sizes and packet sizes of different flows are usually diversified, which can also lead to traffic imbalance among different paths.

Packet-level path splitting, on the other hand, splits traffic in the packet granularity, i.e., packets from a flow can be put to different paths. Since packets of the same flow are usually of similar sizes, packet-level path splitting achieves desirable traffic balance among multiple candidate paths. However, a major concern of packet-level path splitting is that it may cause packet reordering for TCP flows. Although recent studies showed that the path equivalence in modern data center networks can help mitigate the packet reordering problem, random next-hop selection in random packet spraying (RPS) [9] still results in considerable packet reordering and unsatisfactory flow throughputs, which is worsen when link fails and network symmetry is broken [9]. DRB [10] employs IP-in-IP encapsulation/decapsulation [11] to select the core level switch and uses re-sequencing buffer at the receiver to absorb reordered packets, which not only introduces much traffic overhead, but also causes considerable re-sequencing delay [10].

In this paper we design SOPA, a new packet-level path splitting scheme to carry on multi-path routing in data center networks. SOPA advances the state of art by two technical innovations.

First, rather than introducing an additional buffer at the receiver, SOPA increases the fast retransmit (FR) threshold (i.e., the number of duplicate ACKs received at the sender that acknowledge the same sequence number) used in TCP to trigger FR, so as to mitigate the impact of packet reordering on reducing flow's throughput. In the current TCP congestion control algorithm, packet reordering is regarded as an indicator of packet loss, hence three duplicate ACKs will cause packet retransmit at the sender without waiting for the timeouts. Although it works well in single-path routing, in multi-path routing paradigm it misleads the congestion control algorithm, since in most cases packet reordering does not come from packet loss. By increasing the FR threshold, say, to 10, SOPA significantly reduces the number of unnecessary packet retransmits, which accordingly improves the effective throughput for a flow.

Second, instead of randomly selecting the next-hop switch or using IP-in-IP encapsulation to pick a core level switch, SOPA employs source routing to explicitly identify the path for each packet. Increasing the FR threshold only cannot help improve the flow's throughput any more when the FR threshold exceeds a certain value, because more timeouts and packet retransmissions will occur if there are not enough ACKs. SOPA lets the source server adopt a round-robin approach to select the path for a packet and uses source routing to encode the path into the packet header. In a Fat-Tree network, SOPA leverages the topological characteristic and only needs at most four additional bytes to identify the path. As a result, packet reordering is significantly mitigated by exactly balanced traffic loads among the equivalent paths with negligible traffic overhead. Source routing is also very easy to implement on commodity switching chips [12] or the emerging SDN/OpenFlow paradigm [13], without updating the switch hardware.

The NS-3 based simulation results show that SOPA can achieve high network throughput under different workloads, no matter what the network size is. Under the synthesized permutation workload, the average throughput achieved by SOPA is 29.87%, 50.41% and 77.74% higher than that of RPS, Hedera and ECMP, respectively, in a Fat-Tree network built with 24-port switches. Under the workload from a production data center, compared with RPS, Hedera, and ECMP, SOPA improves the average throughput by 53.65%, 343.33% and 348.34%, respectively, in the same network. SOPA can also gracefully encompass link failures without significant performance degradation.

The rest of this paper is organized as follows. Section 2 introduces the background knowledge and related works. Section 3 describes the design of SOPA. Section 4 presents the evaluation results. Finally Section 5 concludes the paper.

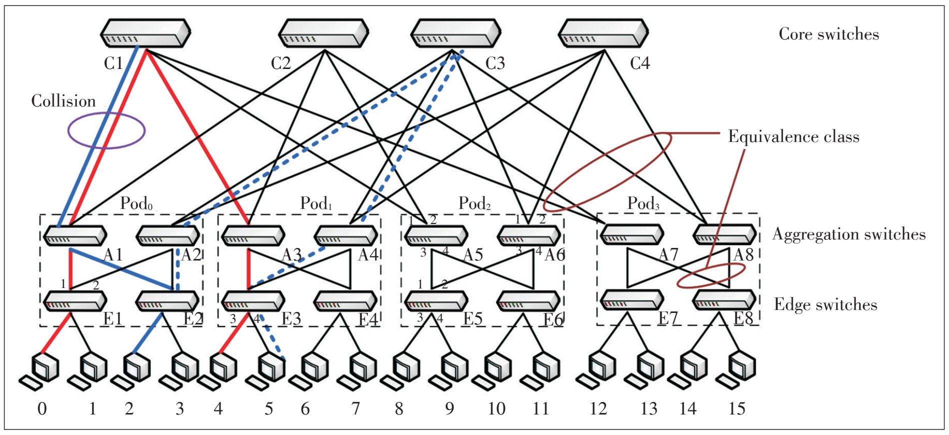
2 Background and Related Work

2.1 Data Center Network and Fat-Tree

A data center network interconnects tens of thousands of, or even hundreds of thousands of servers, and provides routing service to upper-layer applications. Large-scale distributed computations run on these servers and high volumes of traffic are exchanged among the servers. In order to accelerate traffic transfer in bandwidth-hungry applications, "rich-connected" topologies are proposed to increase the network capacity. A typical characteristic of such networks is that usually more than one path exists between any pair of servers.

Fat-Tree [4] is one of the representative "rich-connected" topologies, as shown in **Fig. 1**. The switches are organized into three levels. For a K-array Fat-Tree network (i.e., built with K-port switch), there are K pods ($K = 4$ in the example), each containing two levels of $K/2$ switches, i.e., the edge level and the aggregation level. Each K-port switch at the edge level uses $K/2$ ports to connect the $K/2$ servers, and uses the remaining $K/2$ ports to connect the $K/2$ aggregation-level switches in the same pod. At the core level, there are $(K/2)^2$ K-port switches and each switch has one and only one port connecting to one pod. The total number of servers supported in Fat-Tree is $K^3/4$. For two servers in different pods in Fat-Tree network, there are $(K/2)^2$ paths between them.

In Fat-Tree network, when the packets are forwarded from lower level switches to higher level switches, there are multiple next hops to choose. While there is only one next hop if the



▲Figure 1. A Fat-Tree network with $K = 4$.

packets are forwarded from higher level switches to lower level switches. Let us take the Fig. 1 as an example, and assume a packet needs to be transferred from server 0 to server 4. When the packet arrives at switch E1, it has two next hops, i.e., A1 and A2. Let us suppose that it chooses A1. After arriving at A1, the packet still has two choice, i.e., C1 and C2. After arriving at the core level switch, the packet should be forwarded downwards, and there is only one choice. For instance, if the packet chooses C1 as the next hop at A1, there is only one path to reach server 4 from C1, i.e., C1→A3→E3→4. Similarly, if C2 is chosen as the next hop at A1, the sole path to server 4 is C2→A3→E3→4.

In order to fully utilize the high network capacity of the advanced data center network topologies and provide upper-layer applications with high network throughput, many multipath routing schemes are proposed. Based on the splitting granularity, proposed multi-path routing schemes can be divided into two categories, namely, flow-level path splitting and packet-level path splitting. The former guarantees that packets from the same flow traverse the same path while the latter does not. Besides, multi-path TCP is also designed to utilize the multiple paths in transport level. In what follows we describe the related works respectively.

2.2 Multi-Path Routing with Flow-Level Path Splitting

As a traditional multi-path routing scheme based on flow-level path splitting, ECMP hashes the 5-tuple of every packet to determine the next hop from multiple candidates. VL2 [5] also depends on ECMP to utilize the multiple links in a Fat-Tree like network. However, ECMP fails in balanced utilization of the multiple candidate paths due to the following reasons. First, the random feature of hashing may cause unequal number of flows put in the candidate paths. Second, flows contain different numbers of packets. Hashing collision may forward flows with more packets to the same path, resulting in imbalanced traffic volume. Third, even flows equal in the numbers of packets, the packet size may be different, which also leads to traffic imbalance. Fig. 1 shows an example of hashing collision in ECMP. There are two flows, one from server 0 to server 4, while the other from server 2 to server 5. When switch A1 adopts ECMP to hash the two flows, a collision occurs and both flows choose the link A1→ C1. As a result each flow only grabs half of the link bandwidth. But if we can schedule the flow from server 2 to server 5 to use the path of E2 → A2 → C3 → A4 → E3, no collision exists and each flow can send data with full speed.

In order to overcome the hash collision problem of ECMP, Hedera [8] and Mahout [14] adopt a centralized way to schedule big flows, while using ECMP only for small flows. Hedera depends on edge switches to identify the big flows. Once the bandwidth consumed by the flows exceeds a pre-set threshold (i.e., 10% of the link bandwidth), these flows are identified as big flows. The centralized controller periodically collects information of big flows from edge switches, calculates routing path for each big flow, and installs the routing entries on corresponding switches. Mahout [14] identifies big flows at hosts by detecting the socket buffer taken by the flows, and uses Type of Service (TOS) field in IP header to tag big flows. Each edge switch only needs to install a single routing entry to redirect packets to the centralized controller before routing entries are installed, which greatly reduces the number of routing entries installed on edge switches. Although Hedera and Mahout improve the flow-level path splitting algorithm by spreading big flows into different paths, traffic imbalance among paths still exists when flows are with unequal numbers of packets or unequal packet sizes.

2.3 Multi-Path Routing with Packet-Level Path Splitting

By packet-level path splitting, packets from a flow are distributed to all the candidate paths. Since packets of the same flow are usually of similar sizes, packet-level path splitting can achieve much more balanced utilization of the multiple links. Though it is widely concerned that packet-level splitting may cause packet reordering and confuse TCP congestion control algorithm, the recent work of RPS [9] shows promising results by exploiting the topological feature of data center networks. RPS defines a group of links as an equivalence class, which includes all the outgoing links from the switches at the same hop along all the equal-cost paths [9]. As Fig. 1 shows, links E8→A7 and E8→A8 belong to an equivalence class. RPS tries to keep equal-cost paths between any source-destination pair with similar load by randomly spreading traffic into the links of equivalence class. Considering that the equal-cost paths have the same lengths, if they have similar load as well, the end-to-end latencies along the paths will also be similar. It benefits ordered delivery of packets from different paths, reducing unnecessary FRs at the receiver. But the random packet splitting used in RPS may not result in exactly balanced link utilizations, which will lead to problems as we will show later in this paper.

DRB [10] employs the structure feature of Fat-Tree network and adopts IP-in-IP encapsulation/decapsulation to achieve balanced traffic splitting. In Fat-Tree network, there are many candidate routing paths between each source-destination pair, and each routing path exclusively corresponds to a core switch or an aggregation switch, which is called bouncing switch. For each packet, once the bouncing switch which the packet traverses is picked, the routing path is determined as well. DRB employs the sender to pick a bouncing switch for a packet, and uses IP-in-IP encapsulation to force the packet to take the expected routing path. To mitigate packet reordering, DRB adds a re-sequencing buffer in the receiver below TCP, which stores the reordered packets and postpones delivering the reordered packets to TCP. However, this solution also has the following shortcomings. First, IP-in-IP encapsulation introduces an additional packet overhead of 20 bytes. Second, each connection is equipped with a re-sequencing buffer, and a timer is set up for each reordered packet, which occupies considerable storage and computation resources at the receiver. Third, the re-sequencing buffer causes additional delays to deliver a packet to upper layers, which may affect applications with real-time requirements.

2.4 Multi-Path TCP (MPTCP)

MPTCP [15]–[17] is a transport-layer solution to efficiently utilize the available band-widths in multiple paths. MPTCP splits a flow into many sub-flows, and each sub-flow independently picks a routing path from the available candidates. To achieve fairness and improve throughput, MPTCP couples all the sub-flows together to execute congestion control [17], so as to shift traffic from more congested paths to less loaded ones. In order to eliminate the negative effect of packet reordering, apart from global sequence space, each sub-flow also has its own subsequence space [15]. Each sub-flow uses its own subsequence number to conduct congestion control just as the standard TCP does. A subsequence space to global sequence space mapping scheme is proposed to assemble data at receivers. Compared with multipath routing schemes, MPTCP focuses more on congestion control and fairness issues, paying the overhead for establishing and maintaining the states of sub-flows.

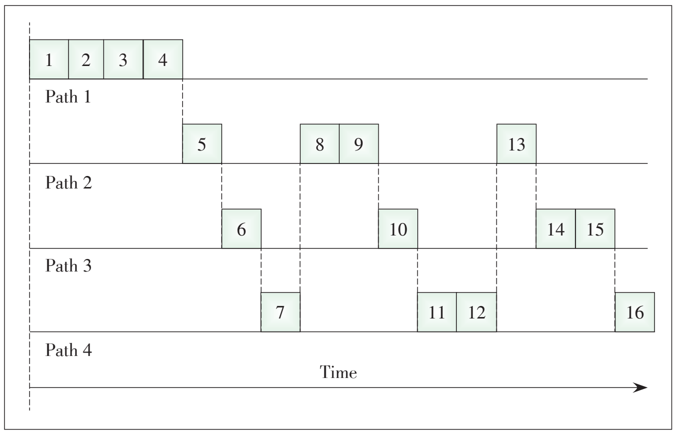
3 SOPA Design

SOPA is a multi-path routing scheme based on packet-level path splitting. In this section we firstly analyze the problems with random packet splitting, then we present the design details of SOPA including two technical components, namely, increasing the FR threshold (to mitigate the impact of packet reordering on lowering a flow's throughput) and source-routing based packet splitting (to mitigate packet reordering).

3.1 Problems with Random Packet Splitting

We start with discussing the problems of random packet splitting, in which every switch randomly splits packets from a flow into equal-cost next hops. From statistical perspective, random packet splitting may lead to similar traffic loads among all the candidate paths during the whole transmission period. However, given a specific short time interval, splitting packets in a random manner cannot guarantee allocating the same amount of traffic to each candidate path. If the load difference among the paths is enough to cause packet reordering and confuse TCP congestion control algorithm to trigger FR, the throughput of the flow will degrade significantly.

Fig. 2 illustrates an example of random packet splitting. We assume the sender's congestion window is big enough to send out 16 packets without waiting for ACKs from the receiver, and there are 4 candidate paths between the two ends. Each box in the figure represents a packet. We can see that each path gets the same share of packets (4 packets on each path) during the



▲Figure 2. An example to illustrate that random packet splitting may cause packet reordering and FR. (Each box denotes a packet, and the number represents the sequence number of the packet. Although random packet splitting allocates 4 packets to each path during the whole period, the instant loads of the paths are different, leading to difference in the queuing delays of the paths. The arrival order of the first 7 packets can be: 1, 5, 6, 7, 8, 2, and 3, which will result in a FR and degrade the throughput of the flow.)

transmission period. But at the beginning of the transmission, the first 4 packets are all allocated to path 1. If unfortunately other flows also allocate packets as shown in Fig. 2, path 1 may have larger queuing delay than other paths in the initial transmission period. The difference in queuing delay can lead to packet reordering at the receiver.

We assume the arriving order of the first 7 packets is: 1, 5, 6, 7, 8, 2, 3. In this case, each reordered packet (packet 5, 6, 7 and 8) will prompt the receiver to send an ACK for the expected packet to the sender, i.e., packet 2. According to the default setting, the three duplicate ACKs will lead the sender into FR phase, cutting down the congestion window into half. So the network throughput drops even though the network is not congested at all. Although the example is just an illustrative one, the simulation below based on NS-3 indeed demonstrates this problem.

In this simulation, we use a Fat-Tree network built by 4-port switches, as shown in Fig. 1. There are 16 servers in the network. A single flow is established between server 0 and server 5, and server 0 sends 100 MB of data to server 5. We set different capacities to links in different levels to intentionally set different oversubscription ratios [4]. All the links connecting servers and edge-level switches as well as the links connecting edge-level and aggregation-level switches have 1 Gbit/s bandwidth, while the capacity of links connecting aggregation-level and core-level switches varies from 1 Gbit/s, 750 Mbit/s, 500 Mbit/s, to 250 Mbit/s. In other words, the oversubscription ratio of the Fat-Tree network varies from 1:1, 4:3, 2:1, to 4:1, respectively.

There are 4 candidate paths between server 0 and server 5 (each corresponding to a core-level switch). As a result, the ideal network throughput for the flow in all the scenarios is 1 Gbit/s.

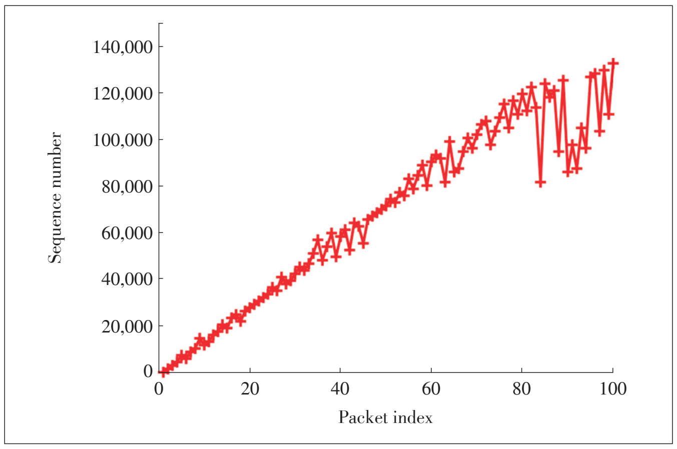
However, when oversubscription ratio varies from 1:1, 4:3, 2:1, to 4:1, the actual throughput of the flow is 986.06 Mbit/s, 966.31 Mbit/s, 578.42 Mbit/s and 296.03 Mbit/s respectively. We can see that the throughput under random packet splitting degrades significantly when the oversubscription ratio grows. When the oversubscription ratio is 4:1, the throughput is only 296.03 Mbit/s, much lower than the ideal value (1 Gbit/s). The reason is that as the oversubscription ratio increases, the bandwidth of links between aggregation-level and core-level switches become smaller. When packets are forwarded upwards, a bottleneck will be built between these two levels of switches, resulting in longer queuing delay. Furthermore, the imbalanced traffic splitting illustrated in Fig. 2 allocates different traffic loads to the candidate paths, and the packets on light-loaded paths will experience shorter delay than that allocated to the more heavily-loaded paths. When the oversubscription ratio is higher, the queuing delay in the bottleneck paths will be longer and the impact of traffic imbalance on the packet reordering will be more significant. Therefore, more FRs are triggered when the oversubscription ratio is higher, resulting in poorer performance.

To validate our analysis, we record the arrival sequence of the first 100 packets under the oversubscription ratio of 4:1, as shown in **Fig. 3**. The x-axis of Fig. 3 denotes the arrival order of packets on server 5, and the y-axis shows the sequence number of each received packet. We observe many reordered packets. These reordered packets send enough duplicate ACKs back to the sender to trigger FR. Trace data shows that, during the whole transmission period (100 MB of data transmission), 347 times of FRs occur at the sender, causing it to resend 2406 packets in total (i.e., 3.35% of all the packets). The FRs reduce the congestion window at the sender and thus degrade the flow's throughput.

3.2 Increasing FR Threshold

Since FR due to packet reordering is the root cause of the undesirable performance of random packet splitting, we want to understand why random packet splitting brings so many FRs. To answer this question, we briefly review the congestion control algorithm of TCP, particularly, the FR algorithm.

In order to ensure reliable data delivery, TCP adopts the acknowledgement scheme, i.e., once receiving a data packet, the receiver sends an ACK message back to the sender. To improve efficiency, modern TCP does not send an ACK for each received packet. Instead, the receiver uses an ACK to acknowledge a batch of sequential packets. However, a reordered packet will prompt an ACK to be sent out immediately. The sender sets a retransmission timer for each unacknowledged packet. If a sent packet or its ACK is dropped, the sender does not know whether the packet has been correctly received or not, and it will resend the packet when the retransmission timer timeouts. This scheme might result in low throughput, because once a packet is lost, TCP has to wait for the expiration of the retransmission timer to resend the packet. During this timeout period, no more new packets can be sent since the congestion window does not slide forward. To tackle this issue, FR is proposed, which is triggered by three duplicate ACKs.



▲Figure 3. Arrival sequence of the first 100 packets with the oversubscription ratio of 4:1. (The random packet splitting causes many reordered packets.)

3.2.1 Algorithm of FR

We use one example to illustrate the working process of FR. In this example, the sender sends out 5 packets in a batch. Unfortunately the first packet is lost, while the 4 subsequent ones successfully arrive at the receiver, each triggering an ACK since it is a reordered packet. Based on today's TCP configuration, three duplicate ACKs trigger the sender to immediately resend the first packet, rather than waiting for its timeout. The FR algorithm is widely used for TCP congestion control in the single-path routing, in which packets belonging to the same flow always traverse the same path if there is no topology change and with high probability a reordered packet indicates there is packet loss due to congestion. However, in the setting of packet-level multi-path routing, packets from the same flow would go through different paths, which may have different loads and queue lengths. As a result, the receiver will get more reordered packets, even if there is no congestion and packet loss in the network.

3.2.2 Benefit and Problem with Increasing FR Threshold

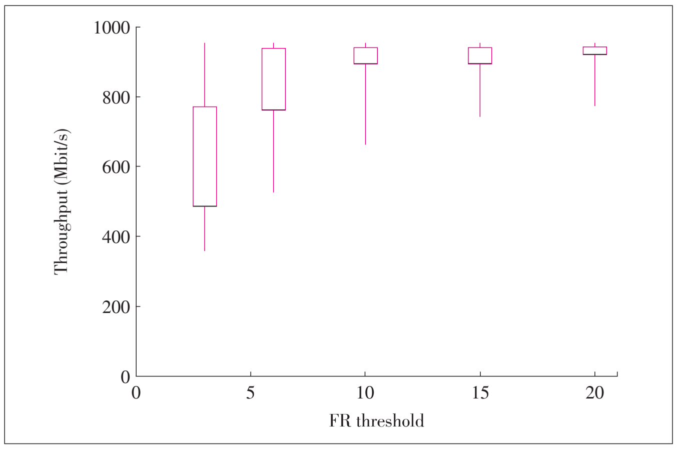
Since packet reordering is most unlikely the sign of congestion in packet-level multi-path routing and unnecessary FRs are the key reasons to lower the throughput, an intuitive idea is to increase the FR threshold to avoid FRs as much as possible. However, does increasing FR threshold really help improve the flow's throughput? And if it does, how large should the FR threshold be? To answer these questions, we need to differentiate two cases, namely, whether there is packet loss or not.

If there is no packet loss, all the reordered packets will finally arrive the receiver, and increasing the FR threshold can efficiently avoid unnecessary packet retransmissions caused by packet reordering and accordingly greatly improve the flow's throughput.

However, if packet loss indeed occurs, the situation is a little bit tricky. If there are enough subsequent packets after the lost packet arriving at the receiver, enough duplicate ACKs can be received by the sender to trigger FR, even if we increase the FR threshold. The only difference is that the time to resend the lost packet(s) is postponed. Considering the Round-Trip Time (RTT) in data center network is very small (e.g., several hundred microseconds), we believe that the performance enhancement by increasing the FR threshold outweighs the introduced delay. However, if there are not enough subsequent packets after the lost packet to trigger FR (e.g., when the congestion is window is small), the lost packet will be transmitted after the retransmission timer timeouts. In this case, increasing the FR threshold will result in more packet retransmissions by timeouts, which inversely degrades the flow's throughput since the TCP's retransmission timeout value (RTO) is much larger than the RTT in data center network.

We use two examples simulated in NS-3 to study the effect of increasing the FR threshold in random packet splitting.

Example 1: We use a 24-array Fat-Tree network and a permutation workload [9], in which each server is a sender or receiver for just one flow. In this workload, there are at most 144 parallel paths for every flow. There is no packet loss in this case, since Fat-Tree network is non-blocking. The FR threshold is set as 3 (default value in TCP), 6, 10, 15, and 20, respectively. **Fig. 4** shows the flows' throughputs against the FR threshold. For each candlestick in the figure, the top and bottom of the straight line represent the maximum and minimum value of the flows' throughputs, respectively. The top and bottom of the rectangle denote the 5th and 99th percentile of average throughput, respectively. The short black line is the average throughput of all the flows. We can see that the throughput indeed improves as the FR threshold increases. By checking the traces, we find that when the FR threshold is 3, 6, 10, 15 and 20, FR takes place for 28708, 6753, 516, 62 and 3 times, respectively.



▲Figure 4. Effect of increasing FR threshold. (As the threshold increases, the throughput improves as well. However, when the FR threshold is larger than 10, the improvement of performance is quite marginal.)

However, when the FR threshold is larger than 10, the improvement of average throughput is quite marginal: the average throughput is 896.24 Mbit/s, 919.82 Mbit/s, and 921.16 Mbit/s when the FR threshold is 10, 15, and 20, respectively. Besides, the minimum flow throughput is also less than expected, because FRs are still triggered even the threshold is larger than 10. When the FR threshold is 10, 15 and 20, there are 516, 62 and 3 flows experiencing FR, and the average number of FRs for each flow is 2.02, 1.26 and 1 time(s), respectively. If we continue increasing the FR threshold, indeed we can eliminate FR, but the following example will show that we cannot increase the FR threshold to an unlimited value.

Example 2: We use the same network topology as Example 1 and assume a client fetches data from 3 servers to mimic an operation in distributed file system and the sum of data size is 64 MB. During the file transmission period, a short flow starts to send small data (64 KB) to the same client. In our simulation, when the FR thresholds are 3, 6, 10, 15, and 20, the throughputs of the short flow are 53.16 Mbit/s, 61.14 Mbit/s, 57.84 Mbit/s, 2.41 Mbit/s and 2.42 Mbit/s, respectively.

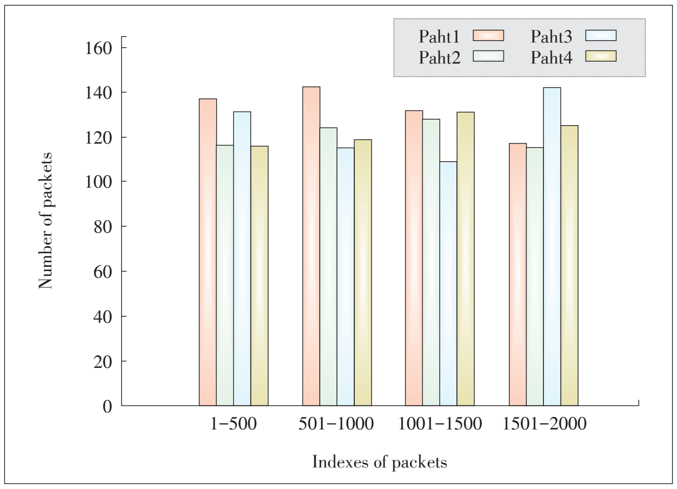
The two examples above demonstrate that increasing the FR threshold has both benefits and problems for the throughput improvement. On one hand, higher FR threshold helps avoid many unnecessary FRs caused by packet reordering. On the other hand, when packet loss indeed occurs, higher FR threshold also causes more timeouts of retransmission timers and thus smaller congestion window. We thus argue that simply increasing the FR threshold alone cannot solve the problem thoroughly. In SOPA we set the FR threshold as 10 (motivated by Examples 1 and 2), and seek for more balanced packet splitting solutions among the equal-cost paths.

3.3 Source-Routing Based Packet Splitting

3.3.1 Design Rationale

We first run a simple simulation to demonstrate that random packet splitting can lead to aggravated packet reordering. We setup a single flow in a 4-array Fat-Tree network, which sends 100 MB data. There are 4 parallel paths between the sender and the receiver. During the entire transmission period, the percentiles of packets allocated to the four paths are 24.89%, 25.10%, 25.01% and 25.00% respectively. However, within every 500 consecutive packets, the number of packets allocated to each path deviates significantly from each other. For the first 2000 packets, **Fig. 5** shows the number of packets allocated to each path for every 500 consecutive packets.

We learn from the figure that the maximum deviation from the average value can be up to 13.6% (this observation well matches the example shown in Fig. 2), even though the overall ratios are roughly close to each other. Due to the imbalanced traffic allocation, some paths may experience congestion and



▲Figure 5. Packets allocation in random packet splitting. (This figure shows how the first 2000 packets are allocated to 4 equal-cost paths. Each group of square columns represents the allocation of 500 packets. Even though almost the same traffic is allocated to each path during the whole transmission period, the instant allocations to the paths are different. The maximum deviation from the average allocation is 13.6%.)

even packet loss, while others do not. Consequently, we need a solution which can avoid packet reordering instead of tolerating packet reordering only.

Rather than randomly selecting the next hop for a packet, SOPA explicitly identifies the routing paths for packets of a flow in a round-robin fashion, which results in exactly balanced utilization of the equal-cost paths regardless of the workload. There are two possible solutions to explicit path identification, namely, switch based and server based.

In switch-based solution, we can let each switch explicitly forward the packets from a flow to the interfaces in a round-robin way. However, it requires the switch to maintain the state for each flow, i.e., records the forwarding interface of the last packet from the flow. Since the number of concurrent flows in data center network is huge [5] and the technical trend for today's data center network is to use low-end commodity switches [4]–[6], it is quite challenging, if not impossible, to implement this idea in a practical data center switch.

In server-based solution, the senders are responsible for explicitly identifying the routing path of each packet and inserting the routing information into the packet, i.e., using source routing. Switches only need to support source routing, without the requirement to maintain per-flow state. It has been shown feasible to realize source routing by programming today's commodity switching chips [12]. Moreover, source routing is even easier to be implemented in the emerging SDN/OpenFlow paradigm, by appropriately configuring the flow tables in switches [13]. In one word, source-routing based packet splitting puts the complexity into servers which have sufficient calculation power and memory [18], while does not need to update the switches' chips.

SOPA further reduces the packet overhead caused by source routing in a Fat-Tree network by exploiting the topological feature. In Fat-Tree network, we define upward forwarding as forwarding packets from a lower-level switch to a higher level switch (i.e., from edge switch to aggregation switch, or from aggregation switch to core switch); and define downward forwarding as forwarding packets from a higher-level switch to a lower-level switch (i.e., from core switch to aggregation switch, or from aggregation switch to edge switch). Although there are 6 hops at most for a source-destination pair (if source and destination are located in different pods), the routing path is determined by the two upward forwarding hops. Therefore, at most two intermediate switches are inserted into the IP headers to support source-routing in SOPA. For each hop in upward forwarding, we use 1 byte to store the forwarding interface. It can theoretically support a 512-array Fat-Tree network (noting that only half of a switch's interfaces are upward ones), which includes 33,554,432 servers in total. Since we only need to store up to two hops' information, the overhead introduced by source routing in SOPA is at most 4 bytes (one byte is option-type octet, two bytes are for two hops' information, and the fourth byte is a padding byte to ensure that the IP header ends on a 32 bit boundary). For a packet size of 1500 bytes, the traffic overhead is only 0.26%.

3.3.2 Technical Components

SOPA includes three major technical components to realize the source-routing based packet splitting, which are described as follows.

Calculating Candidate Paths: This component runs at servers. Flows in a Fat-Tree network can be divided into three categories, namely, inter-pod flows (i.e., the source and destination are in different pods), inter-rack flows (i.e., the source and destination are in the same pod but different racks), and intra-rack flows (i.e., the source and destination are under the same edge switch). The path lengths (in terms of number of hops) for the three categories of flows are 6, 4 and 2, respectively. For the intra-rack flows, there is only one path between the senders and receivers, and we do not need to calculate the candidate paths at all. When calculating the routing paths for inter-pod flows and inter-rack flows, we only need to focus on upward forwarding hops. It is worth noting that the candidate paths should exclude the ones with faulty links. We assume there is a failure notification scheme that the source can be aware of the updated topology, which is easy to realize if there is a central controller in the data center.

Selecting Flow Path: This component runs at servers too. Before sending out a packet, the sender needs to select a routing path from the candidate ones for the packet. The sender chooses the paths for outgoing packets in a round-robin fashion to make sure that each path grabs the same share of traffic from the flow. Once the path is chosen, the sender inserts the path information into the optional field of IP header. Given the powerful processing capability and large memory in today's data center servers, the processing overhead is affordable.

Packet Forwarding: This component runs at switches. The switches' function in SOPA is simply forwarding packets according to the source routing information (upward forwarding) or the destination address (downward forwarding). In an SDN/OpenFlow network [13], forwarding keys are extracted from the packet header and matched against the flow table. In a regular network implementing source routing, in upward forwarding the next hop is popped from the source routing field and used as a key to lookup the forwarding table; while in downward forwarding the table lookup is operated on the destination address.

3.3.3 Benefit and Problem with Source Routing

Fig. 6 shows the performance between source-routing based packet splitting and random packet splitting. We run the same simulation as in Section 3.2 and also set the FR threshold as 3, 6, 10, 15 and 20, respectively. From the simulation results we find that source routing outperforms random splitting under all settings. Both the two solutions improve the flows' throughputs as the FR threshold increases. However, when the FR threshold reaches 10, source routing achieves close-to-optimal throughput for all the flows, by the balanced traffic splitting among the equal-cost paths.

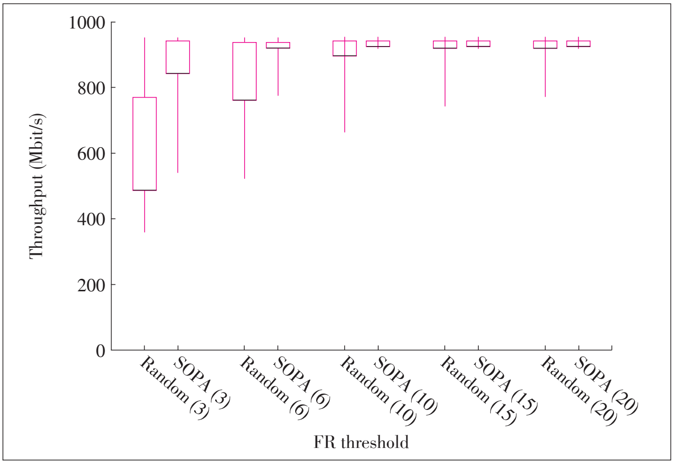
We can also observe from Fig. 6 that using source routing alone cannot completely avoid unnecessary FR. When source routing is employed and the FR threshold is set to 3, the receivers see 2.83% reordered packets, and 2728 times of FRs are triggered (in contrast there are 10.96% reordered packets and 28,708 times of FRs in random packet splitting). The reason is that even we use source routing to ensure balanced traffic splitting, the end-to-end delays of the candidate paths are not the same (the simulations in Section 4.2 show this phenomenon). So the arrival order of packets is not strictly consistent with the sending order. Furthermore, in production data center networks there may be other reasons causing packet reordering, such as the buggy driver of network interface cards (NICs).

As a result, in SOPA we employ both increasing the FR threshold (to 10) and source-routing based packet splitting to improve the flows' throughputs in packet-level multi-path routing. Note that SOPA does not need to update the switch hardware. The modifications on server side is also light-weighted. We only need to rewrite the FR threshold number in TCP congestion control algorithm and maintain the routing path of the last packet for every outgoing flow.

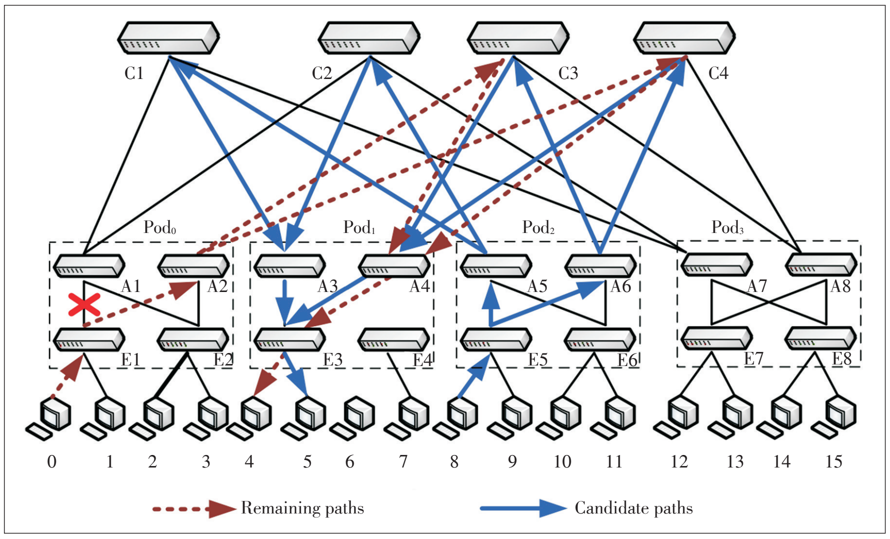
3.4 Failure Handling

The discussion above assumes there is no failure and the network topology is symmetric. However, failures are common in data center networks instead of exceptions [4], [19]. The failures will break the symmetry of the network, and imbalanced traffic load will be allocated to candidate paths, resulting in more aggravated packet reordering. We use an example shown in **Fig. 7** to explain the negative effect brought by failure upon packet-level multi-path routing.

There are two flows, one from server 0 to 4 while the other from server 8 to 5. If there is no failure, both flows have four candidate paths. Now we assume the link between E1 and A1 is broken. Because some paths of flow 0→4 contain the faulty link, the flow can only take two paths, i.e., E1→A2→C3→



▲**Figure 6. Performance comparison between random packet splitting and SOPA. (The number in the parentheses denotes the FR threshold. Both random packet splitting and SOPA improve the performance as the threshold increases, and SOPA outperforms random packet splitting in all settings.)**

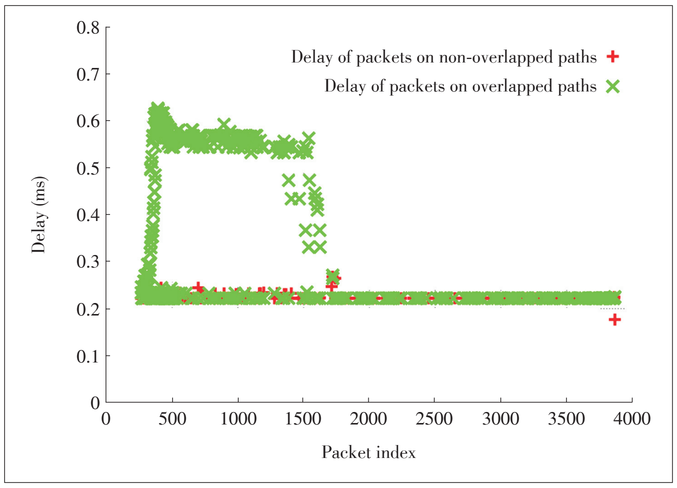


▲**Figure 7. An example to showcase the negative effect brought by failure upon packet-level multi-path routing. (There are two flows, flow 0→4 and flow 8→5. If the link between E1 to A1 fails, the flow (0→4) can only take the two remaining paths, while the other flow (8→5) can still use the four candidate paths, which may cause load imbalance across multiple paths of flow 8→5, degrading its performance.)**

A4→E3 and E1→A2→C4→A4→E3 (displayed as dotted arrow lines in the Fig. 7 and called remaining paths). On the contrary, flow 8→5 can still use the four candidate paths (displayed as solid arrow lines in Fig. 7). Both flows evenly allocate traffic to its available paths, and load imbalance occurs across the four candidate paths of flow 8→5. Different queue lengths will be built among the four paths, leading to aggravated packet reordering at the receiver (server 5).

Fig. 8 validates our analysis, which shows the end to end delay of all packets from flow 8→5. The figure displays the delays for two groups of packets. One group of packets takes the paths which overlap with the remaining paths of flow 0→4, and these paths are called overlapped paths. The other group of packets takes the paths that do not overlap with the remaining paths of flow 0→4, and these paths are called non-overlapped paths. For the first 1700 packets, the packets allocated to the overlapped paths experience much longer delay than those allocated to the non-overlapped paths. Subsequent packets experience almost the same delay. It is because during the initial transmission period, two flows run simultaneously, and the traffic load of the overlapped paths is higher than that of the non-overlapped paths. The different traffic loads across the candidate paths of flow 8→5 result in much more reordered packets at the receiver (i.e., server 5), leading to degraded performance. So flow 0→4 finishes earlier than flow 8→5. In the late stages of data transmission, there is only flow 8→5, so the packets from all paths get similar delays.

From the example, we see that link failure can break the balanced traffic allocation, resulting in more aggravated packet reordering. One possible solution is to employ random early detection (RED) for flows that experience link failure, as in [9]. The goal is to reduce the transmission rate of the flow, decrease the difference in queue lengths of the candidate paths, and mitigate the packet reordering introduced by failure. We argue that this solution does solve the problem efficiently. RED only starts to tag packets when the queue length exceeds a threshold. Before reducing the transmission rate (which is caused by tagged ACK), the flows still send data as if no failure occurs. As a result, different queue lengths can still be built among the candidate paths of the flow whose paths are not affected by the failure, just as Fig. 8 shows.



▲Figure 8. End to end delay of the packets from flow 8→5. (The failure causes flow 0→4 only can take two remaining paths, which are overlapped with two candidate paths of flow 8→5. The figure shows the packets on the overlapped paths experience much longer delay than the packets allocated to the non-overlapped paths.)

Another possible solution is to introduce a re-sequencing buffer at the receiver to absorb the reordered packets, as in [10]. Reordered packets are restored in the re-sequencing buffer and postponed to deliver to TCP. A timer is set for each reordered packet. If an expected packet arrives before timeout, the timer is canceled and this packet along with in-sequence packets are delivered to TCP. Otherwise, the timer expires and the reordered packet is handed to TCP to send back ACKs. Given no packet loss, the solution works fine. However, if some packets are dropped, the buffer has to wait for the expiration of the timer to deliver the packets to TCP, and an additional delay is introduced.

Packet reordering is the main negative effect brought by failure. As long as unnecessary FRs are avoided, the performance can still be guaranteed. SOPA increases the FR threshold, which can effectively avoid unnecessary FRs due to reordered packets and greatly mitigate the negative effect brought by failure. Even in case of packet loss, reordered packets can also produce duplicate ACKs in time to trigger FR. We will evaluate our design under scenarios with failure in Section 4.5.

4 Evaluation

4.1 Simulation Setup

In this section, we evaluate the performance of SOPA using NS-3 [20], which is a packet-level simulator. We use Fat-Tree as the data center network topology. Given the switches composing the Fat-Tree network have K ports, the total number of servers in the network is $K^3/4$. We set K=24 by default unless specified otherwise. The bandwidth of each link is set to 1 Gbit/s, and the latency of each link is 25 ns. The packet size is set as 1500 bytes (including IP header and TCP header). For SOPA, the FR threshold is 10, and for the other schemes, we use TCP's default configuration, i.e., FR threshold of 3. From previous analysis, we know that the throughput of random packet splitting degrades when the oversubscription ratio increases, so we set 1:1 oversubscription ratio for all the simulations, which actually favors random traffic splitting.

In the following simulations, we use RPS as the implementation of random traffic splitting. We firstly use a small-scale simulation to demonstrate that the random packet splitting creates imbalanced load distribution and degraded performance. Then we compare SOPA with ECMP, Hedera and RPS by using large-scale simulations. We implement Hedera following

[8], and use Global First Fit algorithm to calculate the routing paths for big flows. We use two workloads, i.e., permutation workload and production workload, to study various traffic scenarios (details about these two workloads will be introduced later). At last, we also evaluate the performance of SOPA with link failures.

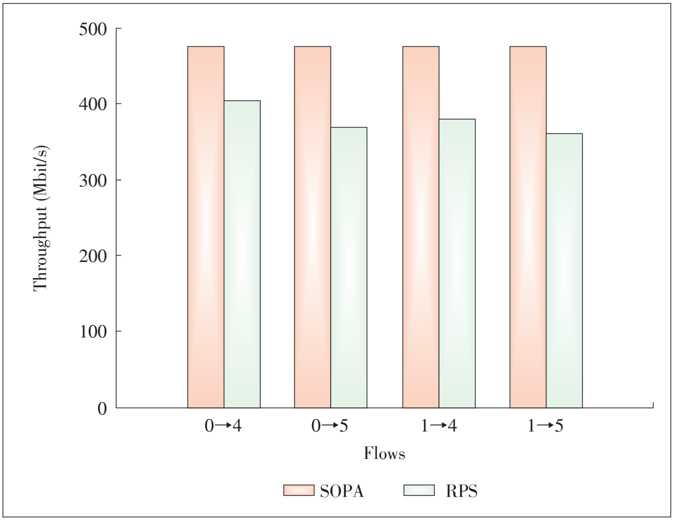
4.2 Negative Effect of Imbalanced Load Distribution

In this section, we use a small-scale simulation to demonstrate the negative effect of imbalance load distribution. A 4-array Fat-Tree network, just as Fig. 1 shows, is used in this simulation. We set up four flows in the simulation, i.e., flow 0→4, 0→5, 1→4, and 1→5, respectively. Each flow sends 10 MB data to its destination. We run both SOPA and RPS in the simulation, respectively.

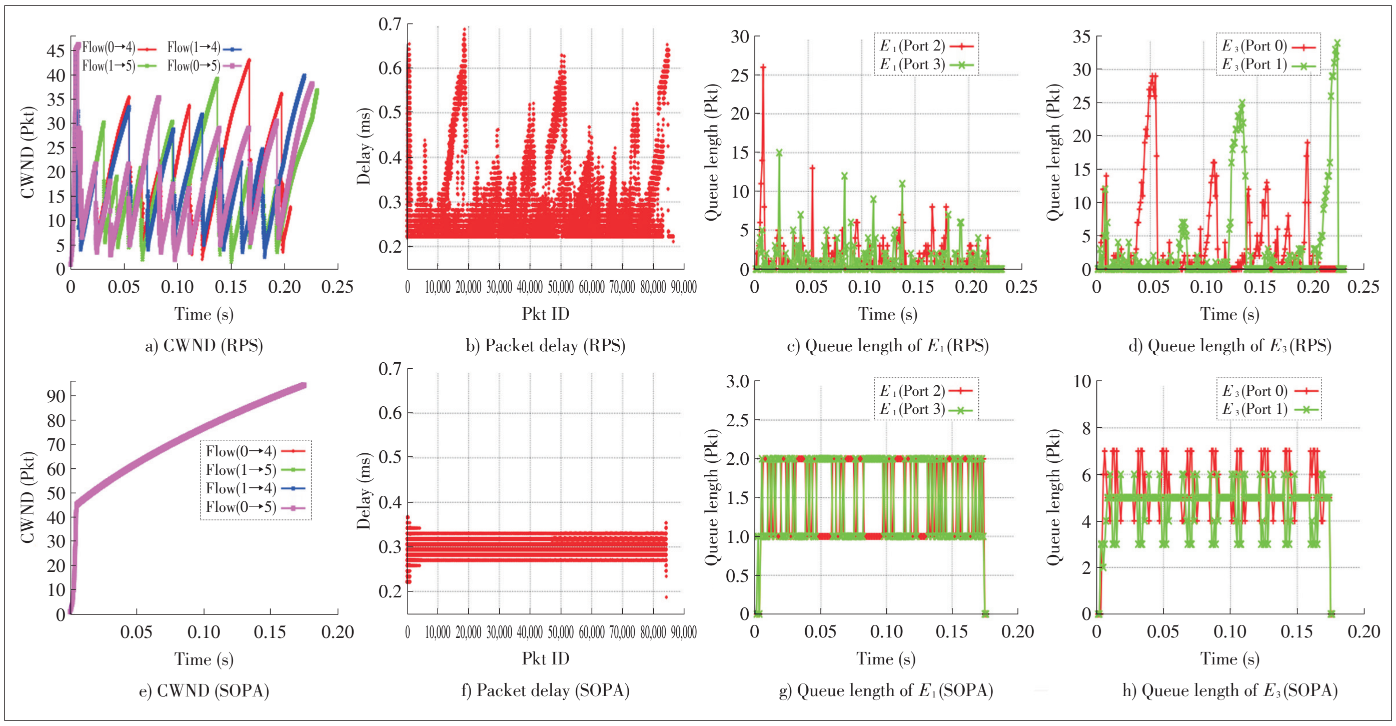
Fig. 9 shows the flows' throughput. When SOPA is used, each flow grabs the fair share of bandwidth, and the throughput of each flow is about 475 Mbit/s. However, when using RPS, the throughput of each flow varies much. The maximum throughput is 404.27 Mbit/s (flow 0→4), while the minimum throughput is 360.55 Mbit/s (flow 1→5). The average throughput of these four flows is only 378.30 Mbit/s. We thoroughly analyzes the reason for the difference in the performance of SOPA and RPS as follows.

4.2.1 Evolvement of CWND

First, we check the evolvement of the congestion window (CWND), as shown in **Figs. 10a** and **10e**. Fig. 10a demonstrates that the size of congestion window of each flow varies much when RPS is used, because each flow experiences many times of FR. For the 4 flows, the FR takes place for 23, 24, 23 and 31 times, respectively. Each FR halves the size of congestion window, and the throughput drops down accordingly. On the contrary, Fig. 10e showcases that when SOPA is employed, the congestion window of each flow increases monotonically during the whole transmission period (from slow start phase to congestion avoidance phase). Therefore, SOPA achieves good



▲**Figure 9. Throughput of 4 flows. (SOPA allocates traffic evenly, and each flow grabs fair share of bandwidth, and the throughput of each flow is about 475 Mbit/s. However, RPS fails to achieve balanced traffic allocation, the average throughput of these four flows is only 378.30 Mbit/s.)**



▲**Figure 10. Comparisons between SOPA and random packet spraying.**

performance, while RPS does not.

4.2.2 End-to-end Packet Delay

To find out why RPS causes so many reordered packets and then many unnecessary FRs, we measure the end-to-end packet delay. **Figs. 10b** and **10f** show the packet delay when RPS and SOPA are employed, respectively. Fig. 13b shows that the packets' delays vary heavily in RPS. The maximum packet delay is 687.4 us, while the minimum packet delay is 211 us. However, the end to end delays of all packets fluctuate within a small range when SOPA is adopted, which is shown in Fig. 10f. The maximum packet delay is 366.27 us, while the minimum packet delay is 187 us. And SOPA only introduces 0.4% reordered packets during the simulation, while 14.92% packets experience reordering when RPS is adopted. This aggravated packet reordering can be attributed to imbalanced traffic splitting of RPS, which builds up different queue lengths on switches and packets from different paths experience different delays. We measure the queue lengths to validate our analysis.

4.2.3 Queue Length

In the simulation all the traffic of the four flows goes through the edge switch E1 and E3 in Fig. 1. The two switches are the highest-loaded switches, so we focus on the queue lengths of the two switches. Since all traffic is forwarded upwards at switch E1 and switch E3 forwards all traffic downwards, we measure the queue lengths of all the upward forwarding ports on E1 (i.e., port 2 and port 3) and the queue lengths of all downward forwarding ports on E3 (i.e., port 0 and port 1).

Figs. 10c and **10d** plot the queue lengths on switch E1 and E3 for RPS, respectively. The figures reveal that RPS builds very different queue lengths on different forwarding ports due to imbalanced traffic splitting. We take switch E1 as an example, at 0.008 s, the queue length of the port 2 is 16 packets, while the queue of the port 3 is empty. Switch E3 also has more diverse queue lengths on port 0 and port 1. For example, at 0.2250 s, the queue of port 0 is empty, while the queue length of port 1 is 34 packets. Note that the average queue length of E3 is bigger than that of E1. Since RPS produces imbalanced traffic splitting at each hop, and E1 is the first hop and E3 is the last hop of these flows. As the last hop, the larger queue length of E3 embodies the accumulated imbalanced traffic splitting at previous hops.

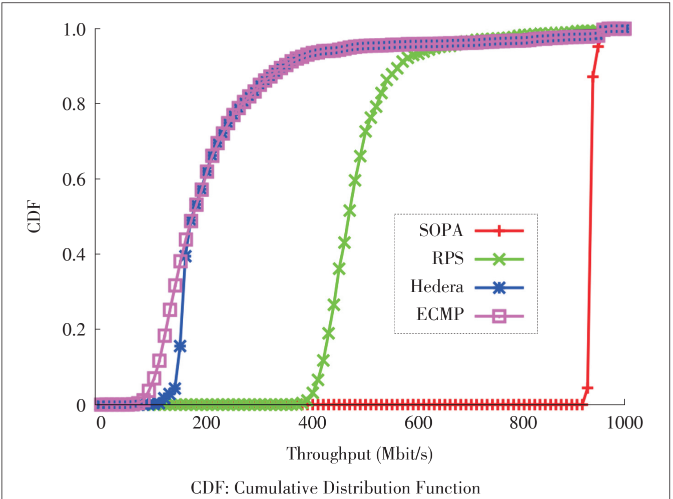
For SOPA, the queue lengths on switch E1 and E3 are shown in Figs. 10g and 10h, respectively. Compared with Figs. 10c and 10d, it is obvious that the queue lengths of different ports on each switch are almost the same, due to the more balanced traffic splitting achieved by SOPA. As a consequence, SOPA does not introduce aggravated packet reordering, and no FR is triggered. However, RPS creates different queue lengths on switches' different forwarding ports, packets on different routing paths may experience different delays (as Fig. 10b shows). Therefore, the receivers see larger number of reordered packets, and many unnecessary FRs are triggered.

For SOPA, the queue lengths on switch E1 and E3 are shown in Fig. 10g and Fig. 10h, respectively. Compared with Fig. 10c and Fig. 10d, it is obvious that the queue lengths of different ports on each switch are almost the same, due to the more balanced traffic splitting achieved by SOPA. As a consequence, SOPA does not introduce aggravated packet reordering, and no FR is triggered. However, RPS creates different queue lengths on switches' different forwarding ports, packets on different routing paths may experience different delays (as Fig. 10b shows). Therefore, the receivers see larger number of reordered packets, and many unnecessary FRs are triggered.

4.3 Permutation Workload

We then study the performance of various multi-path routing schemes, namely, SOPA, RPS, Hedra and ECMP, under synthesized permutation workload. The senders and receivers are picked randomly. We use a Fat-Tree network with K =24, in which there are 3456 servers in total. Each sender sends 10 MB data to its receiver. All flows start simultaneously.

Fig. 11 shows the Cumulative Distribution Function (CDF) of all the flows' throughput. We can see that SOPA significantly outperforms the other three multi-path routing schemes. The average throughput of SOPA is 925.13 Mbit/s, and all the flows' throughputs are above 910 Mbit/s. Compared with SOPA, the average throughput of the flows drops by 47.47% in RPS. The fundamental reason is as explained above: RPS cannot evenly split the traffic across candidate paths, and unequal queue lengths will be built. So the receivers will receive more reordered packets, and unnecessary FRs will be triggered at the senders. As flow-based path splitting solutions, Hedera and ECMP expose even lower performance than RPS. Compared with SOPA, the average throughput drops by 75.21% and



CDF: Cumulative Distribution Function

▲Figure 11. CDF of the flows' throughputs for the five multi-path routing schemes under permutation workload. (Both SOPA and DRB outperform the other three routing schemes, and SOPA also achieves more balanced traffic splitting than DRB.)

76.48%, respectively. The basic reason is that the flow-based multi-path routing cannot fully utilize the rich link resource in the Fat-Tree network. ECMP achieves the lowest throughput because the hashing results of some flows may collide and be scheduled to the same path, which can be avoided by centralized negotiation in Hedera.

4.4 Production Workload

We next study the performance under a more realistic workloads from a production data center [5]: 95% flows are less than 1 MB, and only less than 2% flows exceeds 10 MB. Different from the permutation workload, in this simulation the data is transmitted by multiple flows instead of a single one. The flows are issued sequentially, and each flow randomly picks a destination server. All the servers start data transmission at the same time, and we measure the average throughput of the data transmission under this workload.

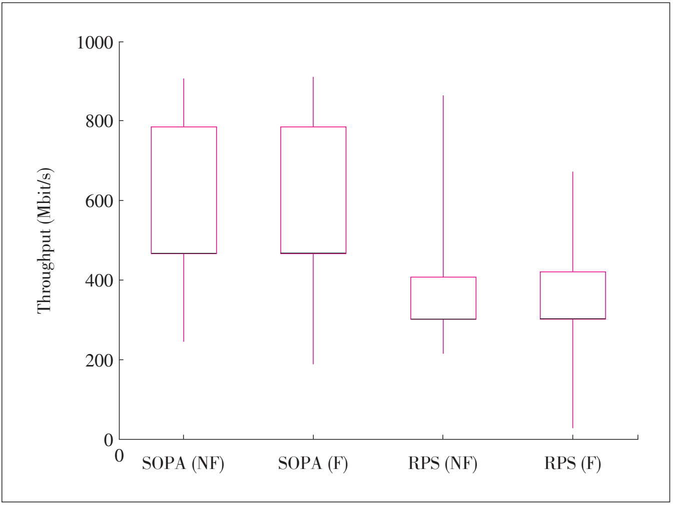
In our simulation, the average throughput for SOPA, RPS, Hedera, and ECMP are 465.5 Mbit/s, 302.96 Mbit/s, 105.005 Mbit/s, and 103.849 Mbit/s, respectively. ECMP gets the lowest throughput, since it neither evenly spreads the flows into multiple paths nor considers traffic sizes in splitting. Hedera performs a little better than ECMP, but the gap is marginal (less than 2 Mbit/s). It is because Hedera targets at scheduling large flows but in this workload 95% flows are small ones (with data size less than 1 MB). Both RPS and SOPA can achieve much higher throughput, due to the fine-grained link utilization by packet-level path splitting. Compared with RPS, SOPA can even improve the throughput by 53.65%, since it explicitly spreads the traffic into the multiple paths in a more balanced way. The result is consistent with that in previous simulations.

4.5 Link Failures

We then evaluate the performance of SOPA when failure occurs. Since failure brings more negative effect for packet-level traffic splitting (introducing more aggravated packet reordering), we only compare SOPA with RPS in this group of simulations. We use production workload to conduct simulation in the same topology as that in the previous simulation. We let the leftmost aggregation switch in the first Pod break down. **Fig. 12** shows the result. In order to showcase the effect of failure, the performance without failure is also plotted.

The x-axis of Fig. 12 denotes both multi-path routing schemes under different settings, wherein "NF" in parenthesis means no failure, and "F" in parenthesis denotes failures. The y-axis shows the throughput of flows. Similarly, for each candlestick in the figure, the top and bottom of the straight line represent the maximum and minimum values of the flow's throughput, respectively. The top and bottom of the rectangle denote the 5th and 99th percentile of average throughput, respectively. The short black line is the average throughput of all flows.

The performance of SOPA is almost not affected by the link



▲**Figure 12. The performance comparison between SOPA and RPS under production workload when failures occur. (In order to show the effect of failure, the performance without failure is also plotted. "NF" means no failure, while "F" denotes that failure has occurred.)**

failure at all, and only the minimum throughput decreases from 244 Mbit/s to 188.17 Mbit/s. This mild performance degradation is attributed to the high FR threshold of SOPA, which can absorb the more reordered packets introduced by the failure. However, failure brings more negative effects to RPS, and both the maximum and minimum throughput of RPS are dropped. When there is no failure, the maximum and minimum throughput are 865.41 Mbit/s and 214.54 Mbit/s, respectively. But in the failure case, their values drop to 672.25 Mbit/s and 26.17 Mbit/s, respectively. This performance degradation is primarily caused by timeouts of the retransmission timers. Trace data shows that when failure occurs, RPS experiences 67 times of timeout and 12000 packets have been dropped (as a contract, there is no packet loss when no failure). The packet losses are caused by traffic congestion, since RPS cannot achieve balanced traffic splitting and the failure aggravates this situation. However, benefiting from balanced traffic splitting, SOPA does not cause packet loss, and there is not a single timeout in the simulation, with or without failures.

5 Conclusion

Many "rich-connected" topologies have been proposed for data centers in recently years, such as Fat-Tree, to provide full bisection bandwidth. To achieve high aggregate bandwidth, the flows need to dynamically choose a path or simultaneously transmit data on multiple paths. Existing flow-level multipath routing solutions do not consider the data size, and may lead to traffic imbalance. While the packet-level multipath routing scheme may create large queue length differential between candidate paths, aggravating packet reordering at receivers and thus triggering FR at the senders. In this paper we design SOPA to efficiently utilize the high network capacity. SOPA adopts source routing to explicitly split data to candidate routing paths in a round robin fashion, which can significantly mitigate packet reordering and thus improve the network throughput. By leveraging the topological feature of data center networks, SOPA encodes a very small number of switches into the packet header, introducing a very light overhead. SOPA also immediately throttles the transmission rate of the affected flow as soon as the failures are detected to promptly mitigate the negative affect of failures. NS-3 based simulations show SOPA can efficiently increase the network throughput, and outperform other schemes under different settings, irrespective of the network size.

Reference

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. 19th ACM Symposium on Operating Systems Principles*, New York, USA, 2003, pp. 29–43.
[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. 6th Symposium on Operating Systems Design and Implementation*, Berkeley, USA, 2004, pp. 137–149.
[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *in Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, New York, USA, 2007, pp. 59–72. doi: 10.1145/1272998.1273005.
[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM 2008 Conference on Data Communication*, Seattle, USA, 2008, pp. 63–74. doi: 10.1145/1402958. 1402967.
[5] A. Greenberg, J. R. Hamilton, N. Jain, et al., "VL2: a scalable and flexible data center network," in *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, 2009, pp. 51–62. doi: 10.1145/1592568. 1592576.
[6] C. Guo, G. Lu, D. Li, et al., "BCube: a high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM*, Barcelona, Spain, 2009, pp. 63–74.
[7] D. Li, C. Guo, H. Wu, et al., "Scalable and cost-effective interconnection of data-center servers using dual server ports," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 102–114, Feb. 2011. doi: 10.1109/TNET.2010.2053718.
[8] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, USA, 2010, pp. 1–15.
[9] A. Dixit, P. Prakash, Y. Hu, and R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, Turin, Italy, 2013, pp. 2130–2138. doi: 10.1109/INFCOM.2013.6567015.
[10] J. Cao, R. Xia, P. Yang, et al., "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. Ninth ACM Conference on Emerging Networking Experiments and Technologies*, Santa Barbara, USA, 2013, pp. 49–60. doi: 10.1145/2535372.2535375.
[11] IETF. (2013, Mar. 2). *IP encapsulation within IP* [Online]. Available: https://datatracker.ietf.org/doc/rfc2003
[12] C. Guo, G. Lu, H. J. Wang, et al., "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th International Conference on Emerging Networking Experiments and Technologies*, Philadelphia, USA, 2010. doi: 10.1145/1921168.1921188.
[13] ONF. (2017, Apr. 1). *Open networking foundation* [Online]. Available: https://www.opennetworking.org
[14] A. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, Shanghai, China, 2011, pp. 1629–1637. doi: 10.1109/INFCOM.2011.5934956.
[15] C. Raiciu, S. Barre, C. Pluntke, et al., "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, Toronto, Canada, 2011, pp. 266–277. doi: 10.1145/2018436.2018467.
[16] C. Raiciu, C. Paasch, S. Barr, et al., "How hard can it be? Designing and implementing a deployable multipath TCP," in *USENIX Symposium of Networked Systems Design and Implementation*, San Jose, USA, 2012, pp. 29–29.
[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Conference on Networked Systems Design and Implementation*, Boston, USA, 2011, pp. 99–112.
[18] M. Alizadeh, A. Greenberg, D. A. Maltz, et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, New York, USA, 2010, pp. 63–74. doi: 10.1145/1851182.1851192.
[19] R. Niranjan Mysore, A. Pamboris, N. Farrington, et al., "PortLand: a scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM*, Barcelona, Spain, 2009, pp. 39–50. doi: 10.1145/1594977.1592575.
[20] *NS-3* [Online]. Available: http://www.nsnam.org

Manuscript received: 2017-03-22

Biographies

LI Dan (tolidan@tsinghua.edu.cn) received the M.E. degree and Ph.D. from Tsinghua University, China in 2005 and 2007 respectively, both in computer science. Before that, he spent four undergraduate years in Beijing Normal University, China and got a B.S. degree in 2003, also in computer science. He joined Microsoft Research Asia in Jan. 2008, where he worked as an associate researcher in Wireless and Networking Group until Feb. 2010. He joined the faculty of Tsinghua University in Mar. 2010, where he is now an associate professor at Computer Science Department. His research interests include Internet architecture and protocol design, data center network, and software defined networking.

LIN Du (lindu1992@foxmail.com) received the B.S. degree from Tsinghua University, China in 2015. Now, he is a master candidate at the Department of Computer Science and Technology, Tsinghua University. His research interests include Internet architecture, data center network, and high-performance network system.

JIANG Changlin (jiangchanglin@csnet1.cs.tsinghua.edu.cn) received the B.S. and M.S. degrees from the Institute of Communication Engineering, PLA University of Science and Technology, China in 2001 and 2004 respectively. Now, he is a Ph.D. candidate at the Department of Computer Science and Technology, Tsinghua University, China. His research interests include Internet architecture, data center network, and network routing.

WANG Lingqiang (wang.lingqiang@zte.com.cn) received the B.S. degree from Department of Industrial Automation, Zhengzhou University, China in 1999. He is a system architect of ZTE Corporation. He focuses on technical planning and pre-research work in IP direction. His research interests include smart pipes, next generation broadband technology, and programmable networks.