Zhang *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:37
https://doi.org/10.1186/s13677-021-00252-8

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Fine-grained load balancing with traffic-aware rerouting in datacenter networks

Tao Zhang[1,2], Yasi Lei[1], Qianqiang Zhang[1], Shaojun Zou[1,2*], Juan Huang[1] and Fangmin Li[1,2]

Abstract
Modern datacenters provide a wide variety of application services, which generate a mix of delay-sensitive short flows and throughput-oriented long flows, transmitting in the multi-path datacenter network. Though the existing load balancing designs successfully make full use of available parallel paths and attain high bisection network bandwidth, they reroute flows regardless of their dissimilar performance requirements. The short flows suffer from the problems of large queuing delay and packet reordering, while the long flows fail to obtain high throughput due to low link utilization and packet reordering. To address these inefficiency, we design a fine-grained load balancing scheme, namely TR (Traffic-aware Rerouting), which identifies flow types and executes flexible and traffic-aware rerouting to balance the performances of both short and long flows. Besides, to avoid packet reordering, TR leverages the reverse ACKs to estimate the switch-to-switch delay, thus excluding paths that potentially cause packet reordering. Moreover, TR is only deployed on the switch without any modification on end-hosts. The experimental results of large-scale NS2 simulations show that TR reduces the average and tail flow completion time for short flows by up to 60% and 80%, as well as provides up to 3.02x gain in throughput of long flows compared to the state-of-the-art load balancing schemes.

Keywords: Rerouting, Load balancing, Data center networks

Introduction

Guaranteeing application performance is crucial for providing good user experience in datacenters. Tons of studies have reported that optimizing the transmission performance of datacenter network (DCN) is the key [1–6]. Therefore, to boost the network capacity thus speeding up data transfer, modern DCNs are usually organized in multi-rooted tree topologies with rich parallel paths, such as leaf-spine [7–11], and split the application traffic among multiple available paths. Though large bisection network bandwidth has been achieved by this way, how to improve the transmission performance of application traffic by resorting to efficient load balancing still remains elusive.

Equal Cost MultiPath (ECMP) [12] is the most typical flow-level load balancing scheme in production datacenter, but is far from efficient because of hash collision and the inability to reroute paths [7]. Random Packet Spraying (RPS) [11] and DRB [13] adopts fine-grained rerouting, hence are more flexible and efficient than ECMP. However, they are oblivious to path condition thus suffering from serious packet reordering. LetFlow [8] and Presto [14] make a good balance between packet reordering and link utilization by adopting per-flowlet and per-flowcell switching granularity to reroute flows. Nonetheless, both of them are inherently passive and fail to timely react to the change of path condition. What's more, none of the above schemes is aware of the mixed heterogeneous traffic, thereby leading to the unsatisfied flow-level transmission performance.

*Correspondence: shaojunzou@ccsu.edu.cn
[1]School of Computer Engineering and Applied Mathematics, Changsha University, 410022 Changsha, China
[2]Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, 410022 Changsha, China

Current datacenter supports a large number of soft real-time applications, including advertising, recommender system, retail and web search [6, 15, 16]. They generate huge amounts of data flows with varying sizes and dissimilar performance requirements. Many works have shown that these heterogeneous data flows take the form of a heavy tail distribution, i.e., more than 80% delay-sensitive short flows containing only about 10% data mix with less than 20% throughput-oriented long flows possessing near 90% data [17–19]. Therefore, when short and long flows coexist and compete for the bandwidths of parallel paths, their diverse performance requirements put the existing load balancing schemes in a dilemma.

On the one hand, fine-grained load balancing achieves uniform load distribution, which contributes to providing relatively low queuing delay for short flows. However, they easily result in serious packet reordering, greatly impairing the transmission performances of both short and long flows. On the other hand, coarse-grained load balancing effectively eliminates packet reordering, while easily causes the unbalanced load distribution and low link utilization. Some paths are highly congested, but the remained ones are unused. Once some unlucky short flows enter the congested paths, their flow completion times are inevitably increased, resulting in large tail latency. Besides, casually selecting path regardless of flow types and path conditions when rerouting fails to attain efficient bandwidth allocation.

Therefore, in this paper, we first investigate the key factors that impact the transmission performances of short and long flows in load balancing. Then, we propose a fine-grained load balancing scheme TR, which identifies flow types and avoids packet reordering, as well as carries out flexible and traffic-aware rerouting to balance the performances of both short and long flows.

Our contributions are summarized as follows:

- We conduct extensive simulation-based studies to show that adopting inflexible rerouting cannot provide low latency for short flows and high throughput for long flows simultaneously.
- We propose TR to improve the transmission performances of short and long flows. TR first leverages the reverse ACKs to estimate the switch-to-switch delay, thus excluding paths that potentially cause packet reordering. Then, TR performs flexible and traffic-aware rerouting to balance the performances of both short and long flows.
- We run large-scaled NS2 simulation tests to evaluate the performance of TR. The results show that TR effectively reduces the average and tail flow completion time of short flows by up to 60% and 80%, as well as increases the throughput of long flows by up to 3.02x compared with the state-of-the-art datacenter load balancing schemes.

The rest of the paper is organized as follows. We present the design motivation in Design motivation section. The algorithm and details of TR are elaborated in TR design section. We make performance evaluation in Performance evaluation section, and discuss the related works in Related works section. Finally, we offer concluding remarks in Conclusion section.

Design motivation

In this section, we investigate the key factors impact the transmission performance of short and long flows in load balancing. To illustrate the problem, we conduct extensive NS2-based simulation tests and choose ECMP [12], LetFlow [8], and RPS [11] as the representatives of flow-level, flowlet-level, and packet-level load balancing schemes, respectively. The network topology used in the tests is leaf-spine.

As shown in Fig. 1, two groups of hosts (each has 10 hosts) are connected via two leaf switches, ten spine switches, and many links. The switch buffer size is set as 250 packets [6]. We randomly choose 5 parallel paths as the bad paths, while the remained 5 paths are the good paths. Each link of the good paths is with 1Gbps bandwidth and 25μs propagation delay, and thus the round trip propagation delay (RTT) of good paths between two groups is 200μs. For the bad paths, we gradually increase the propagation delay of their links to enlarge the degree of topology asymmetry. Therefore, the ratio of bad path's RTT to good path's RTT varies from 1.5 to 3.5. In our tests, the hosts in Group 1 send 100 DCTCP flows generated based on Data Mining workload (see Table 1) to the hosts in Group 2 by following a Poisson process. The threshold of flowlet using in LetFlow is set as 500μs [8]. We evaluate the performance of three representative schemes in terms of the average flow completion times (AFCTs) of short flows (<100KB), the total throughput of long flows (≥100KB) [17, 18, 20], average queuing delay, the ratio of retransmission packets caused by packet reordering, etc.

In Fig. 2(a), due to using fine-grained packet scattering, traffic under RPS is evenly distributed to all the available paths, hence the short flows experience the lowest queuing delay. Nonetheless, flows in the asymmetric scenario also have a high risk of experiencing packet reordering, which confuses the control logic of TCP stack deployed on end-hosts, generating many extra retransmission packets. As shown in Fig. 2(b), the short flows under RPS always have much higher retransmission ratio across all the test cases compared to ECMP and LetFlow. Note that no packet dropping happens throughout the whole test, hence all the retransmission packets are attributed to

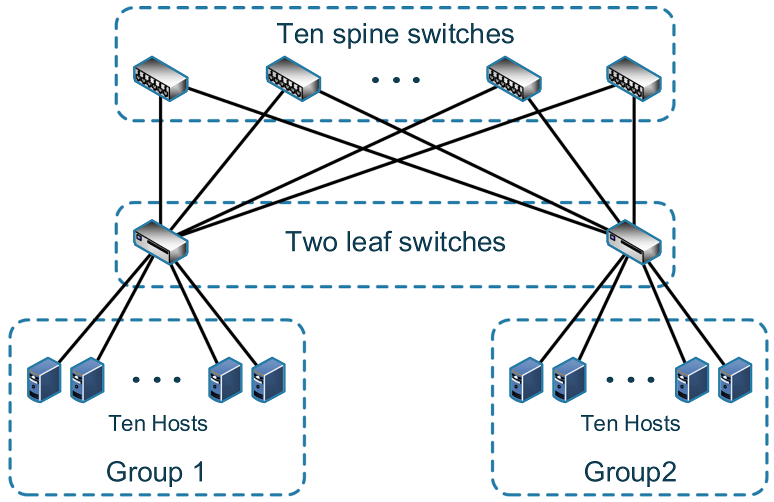


Fig. 1 The leaf-spine topology

packet reordering. Figure 2(c) shows that although possessing much lower queuing delays, the AFCTs of short flows under RPS are significantly higher than those under ECMP and LetFlow, and the performance gaps are widening as enlarging the degree of asymmetry.

On the other hand, it is known that the long flows require high goodput, meaning that the load balancing scheme should help them travel through as many parallel paths as possible because more parallel paths provide larger bisection bandwidth. The fine-grained scheme such as RPS can achieve that since traffic can equiprobably select each path at packet level. However, the result is completely opposite. Figure 3(a) presents the calculated standard deviation after measuring the throughput of long flows transmitted on each path. As expected, RPS achieves much more balanced load distribution compared to the other schemes. LetFlow performs between RPS and ECMP. Nonetheless, the retransmission ratio of long flows under RPS is again pronouncedly higher than those under LetFlow and ECMP, as shown in Fig. 3(b). Consequently, the results shown in Fig. 3(c) imply that, due to taking into account both avoiding the packet reordering and improving the path utilization, LetFlow achieves higher total throughput of long flows compared to the fine-grained RPS and the coarse-grained ECMP. Moreover, with the rising of asymmetric degree, the total throughput of long flows under RPS falls precipitously, and the path utilization is even only about 0.12.

The above observations lead us to conclude that: fine-grained packet scattering can provide uniform load distribution and low queuing delay, but easily leads to serious packet reordering, which greatly impairs the transmission performances of both short and long flows. Coarse-grained path switching at flow- or flowlet-level effectively alleviates packet reordering, but may generate high queuing delay and cannot guarantee high link utilization. Besides, the transmission performances of short and long flows could be further improved if flow types and path condition can be taken into account during traffic rerouting. In the following part, we design a fine-grained load balancing scheme TR to balance the performance requirements of short and long flows under dynamic network conditions.

TR design

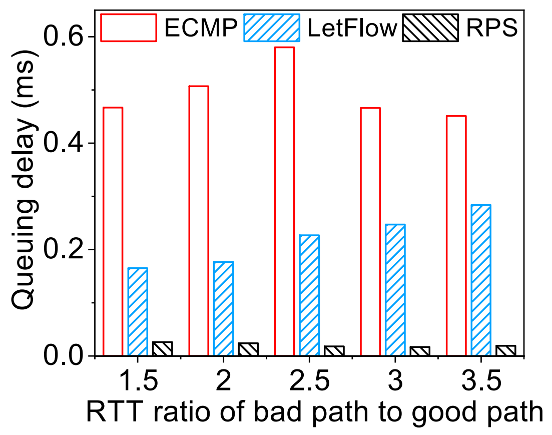
In this section, we first introduce the basic idea of TR. Then, we present its algorithm and elaborate the design details. Finally, we present a model to analyze TR's performance gain.

Basic insight

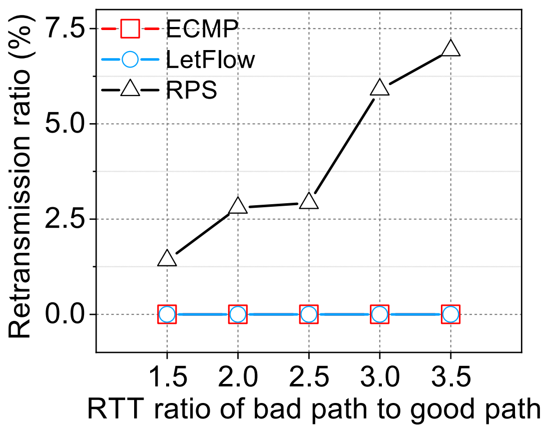
As mentioned earlier, both long and short flows are negatively affected by the out-of-order problem, hence the design of TR includes an out-of-order prediction mechanism. To be specific, for each packet, TR estimates the time at which it reaches the next leaf switch via each alternative output queue, called Time to Next Leaf (TNL), and compares it with the corresponding time of the previous

Table 1 The proportions of flows in different size ranges under realistic workloads

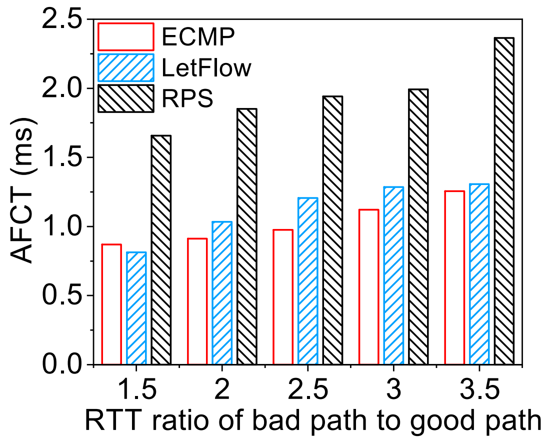
Flow type	Data Mining	Web Search	Cache Follower	Web Server
0-10KB	78%	59%	50%	68%
10KB-100KB	5%	3%	3%	18%
100KB-1MB	8%	18%	18%	14%
>1MB	9%	20%	29%	-



(a) Average queuing delay



(b) Ratio of retransmission packets



(c) Average flow completion time

Fig. 2 The performances of short flows

packet in the same flow. If the computed TNL of an output queue for the current packet is smaller than the stored TNL of its previous packet, the current packet will probably reach the next leaf switch earlier than its previous packet once using this output queue, resulting in packet reordering. Otherwise, TR considers this output queue as an alternative output queue to forward the current packet. By this way, TR ensures the orderly transmission for each flow.

On the other hand, to make TR be traffic-aware, the first thing is to identify if a flow is a short or long flow in advance. However, it is not feasible for many applications and host stack to know how much data involved in a flow before finishing its data transfer. As done by many previous works [18, 20–22], TR identifies flow types by counting how much data has been sent. when the amount of data has been sent in a flow exceeds a threshold of 100KB, it is identified as a long flow. Otherwise, it is a short flow. This threshold value is chosen in accordance with many existing papers [6, 17, 18, 20–27]. One problem of this method is that long flow will be mistakenly considered as short flow when transmitting the first 100KB of data. Fortunately, this kind of negative impact is trivial since the number of long flows are very small under the data center traffic [17, 18, 20], hence such misjudgment does not happen very often. Furthermore, the duration of this process is transient since short flow generally has a very small life time.

With the knowledge of flow types, TR executes flexible and traffic-aware rerouting under orderly transmission. Specifically, each packet of short flows chooses the fastest path to forward, while long flows should use more parallel

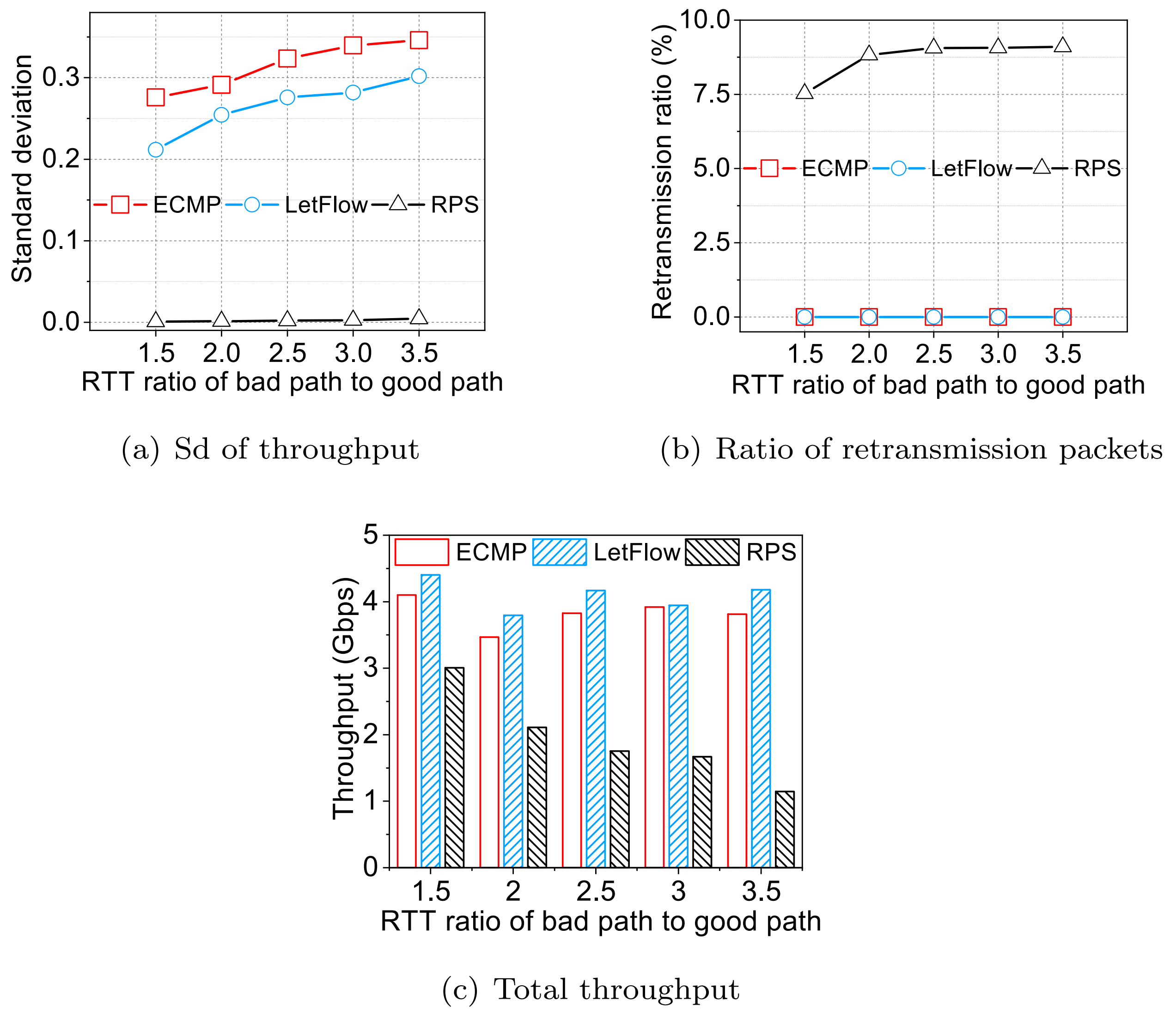


Fig. 3 The performances of long flows

paths to maintain high link utilization and not block the short flows as much as possible.

The algorithm of TR

In this section, we first present the overview of TR, and then describe its pseudocode, in which the used variables are explained in Table 2.

TR overview

TR is deployed on switch. By calculating TNL for the newly arrived packet and comparing with that of the last arrived packet within the same flow, TR picks out those output queues which do not cause packet reordering and uses them to forward packets. Meanwhile, TR adaptively adjusts the granularity of path switching and routing strategy according to the flow types and path condition, so as to ensure the performance of long and short flows simultaneously.

As shown in Fig. 4, when a packet arrives at the leaf switch, TR first updates the size of data sent by current flow to identify its type. Then, TR computes the TNL value of each available output queue for the current packet and compares it with the TNL value of the last arrived packet within the same flow. After finding out those output queues that packets can be delivered in order, TR picks out one that meets the performance requirements of current flow according to its type for forwarding. Specifically, the short flow selects the output queue that can reach the next leaf switch earlier for forwarding, thus achieving low latency transmission. Long flow minimizes its negative impact on the transmission of short flows, and take full advantage of all available paths to increase

Table 2 The descriptions of variables in TR algorithm

Variable	Description
i	The flow number
j	The output queue number
fs (measured in bytes)	The size of data sent in current flow
ps (measured in bytes)	The packet size
p_TNL	The TNL of previous packet
c_TNL	The TNL of current packet
T_l	The size threshold of long flow
S	The set of output queues without introducing packet reordering
a	The output queue with the minimal c_TNL
b	The output queue with the fewest short flows

network utilization, obtaining high throughput. We show the pseudocode of TR in the following part.

Pseudocode of TR

As shown in algorithm 1, when establishing the TCP connection of flow i, the switch assigns it an entry of flow table based on the hash result of its 5-tuple and performs initialization. When a packet P of flow i arrives at the switch, the size of data sent by flow i is updated. Then, TR calculates the TNL for each available output queue of flow i. If an output queue's TNL is larger than the TNL of P's previous packet, it is added into S_i, which is the set of output queues without introducing packet reordering for flow i. After S_i is updated and flow i's type is identified, TR determines the forwarding path for P. If flow i is a short flow, TR selects the output queue a_i from S_i to forward P such that the TNL of a_i is the smallest in S_i. If flow i is a long flow, TR finds out the output queue b_i from S_i to forward P such that there exist the fewest short flows in b_i compared to the remained output queues in S_i. Besides, there may exist multiple output queues with the fewest short flows. In this case, the fastest one among them will be chosen to forward P.

Details

Next, we elaborate the implementation details of TR and answer the following questions: why and how to calculate TNL for each arriving packet, and how to obtain the number of short flows existing in each output queue.

TNL calculation

The two-level structure of leaf-spine topology has been widely used in modern data centers [7–11]. In this kind of network, the number of leaf switches on the path between any pairs of hosts is at most 2 [7–9], which means that if two adjacent packets in a flow can be delivered in order between two leaf switches, they also arrive at the receiver in order. We use an example to illustrate this observation.

In Fig. 5, assuming that H2 sends a flow F to H8. F's data packets may travel through different spine switches (from K to V), and then gather at leaf switch E. There is only one path from E to H8, hence the sequence of F's data packets arrive at E will not be changed when they reach to H8. Similarly, since there is only one path from H2 to M, the sequence of data packets arriving at M certainly follows the sending order of H2. Therefore, as long as F's data packets are delivered in order from M to E, packet reordering does not happen. To avoid packet reordering, when a data packet arrives at the leaf switch near to its sender, TR needs to predict if it reaches to the leaf switch

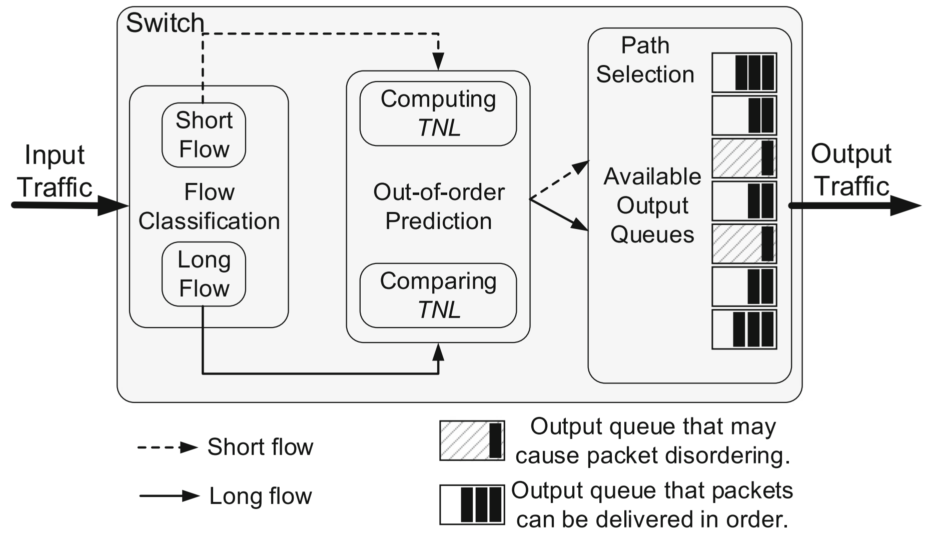


Fig. 4 The overview of TR

Algorithm 1 Pseudocode of TR:

```
1:  Initialization:
2:  $T_l \leftarrow 100KB$;
3:  Once the connection of flow $i$ is established:
4:  $fs_i \leftarrow 0$; $p\_TNL_i \leftarrow 0$; $c\_TNL_{ij} \leftarrow 0$; $S_i \leftarrow \Phi$;
5:  //updating the size of data sent in flow $i$.
6:  On receiving a packet $P$ from flow $i$:
7:  $fs_i \leftarrow fs_i + ps_i$;
8:  //generating the set of output queues without introducing packet reordering.
9:  for each available output queue $j$ of flow $i$ do
10:     computing $c\_TNL_{ij}$;
11:     if $c\_TNL_{ij} \geq p\_TNL_i$ then
12:         adding the $j$th output queue into $S_i$;
13: //carrying out the traffic-aware path selection.
14: if $fs_i \leq T_l$ then
15:     finding out $a_i$ from $S_i$;
16:     forwarding $P$ via $a_i$;
17:     $p\_TNL_i \leftarrow$ the TNL of $a_i$;
18: else
19:     finding out $b_i$ from $S_i$;
20:     if $b_i$ is not unique then
21:         select $b_i$ with the smallest TNL;
22:         forwarding $P$ via $b_i$;
23:         $p\_TNL_i \leftarrow$ the TNL of $b_i$;
```

near to its receiver earlier than its previous packet once choosing a parallel path to forward. Namely, the TNL of current packet is supposed to be not smaller than that of its previous packet in *F*.

TNL generally includes two parts. The first part is the basic propagation delay between two leaf switches, called *pd*, which can be measured by leaf switch when network is idle. This operation does not incur non-trivial overhead since *pd* is usually unchanged unless link failure happens. In fact, link failure does not happen very often [7–9]. The second part is the total queuing delay of the output queues in the leaf switch near to the sender and the spine switch. The former one can be obtained based on the local information including the queue length of output queue *lql* and its forwarding bandwidth *lqw*. Similarly, for the spine switch, with the queue length *sql* and forwarding bandwidth *sqw* of its output queue, we can also compute its queuing delay. Both *lqw* and *sqw* are static and can be obtained in advance. However, since each packet needs to calculate TNL before selecting path, the leaf switch near the sender is supposed to learn the queue length of spine switch timely, which is not easy.

To address this problem, TR resorts to the reverse ACKs (acknowledgement packets) to carry the queue length of spine switch. In Fig. 5, H8 continues to send ACKs to H2. ACK 3 and ACK 4 are modified when traveling through *K* and *V*. The corresponding modifications is to carry their queue lengths to notify *M*. After understanding the queue lengths of *K* and *V*, *M* restores the modified regions in ACKs 3 and 4, making the modifications be transparent to H2. As shown in Fig. 5, when ACK 1 and ACK 2 arrive at H2, H2 will be unaware that they have ever been modified since *M* have already restored their modified regions when they pass through.

Another problem is which region can be modified in the ACK packet. TR responds that by resorting to the field of Time To Live (TTL), which generally has 8 bits. For the scale of current DCN, the number of hops that a packet experiences between its sender and receiver is commonly smaller than 7 [8, 9, 28], which just needs 3

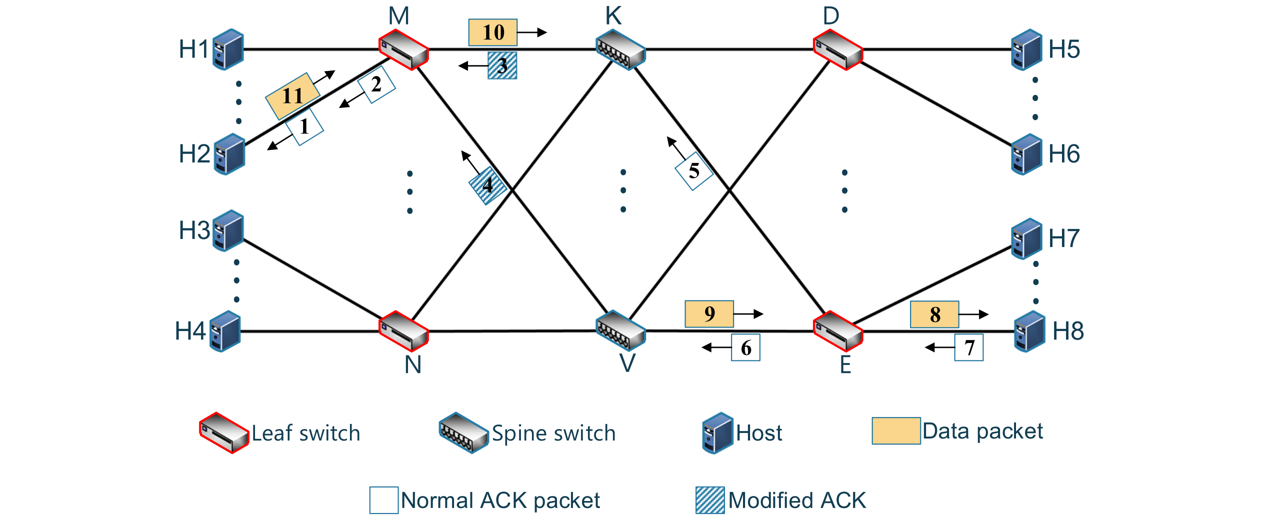


Fig. 5 Out-of-order prediction

bits to store. Hence, we can use the remained 5 bits to carry the information of queue length. The leaf switch and spine switch need to negotiate how to use the 5 bits to represent the specific queue length in advance. There are much coding methods to achieve that, while TR employs a very simple approach to minimize the overhead. Specifically, if the buffer size of spine switch is BS and k bits is used to represent the queue length, the actually increased queue size is $BS/2^k$ when the code of queue length received by leaf switch increases 1. Note that the reverse ACK packet is not the unique option to carry the information of queue length. The reason that TR selects the reverse ACK packet is that it usually has very high priority to be forwarded by switch thus minimizing the effect of reverse delay. One can also use the reverse data packet or other ways to achieve the same functionality.

Finally, according to the above discussion, the TNL of the jth output queue in a leaf switch (TNL_j) can be calculated by

$$TNL_j = ct + pd_j + \frac{lql_j}{lqw_j} + \frac{sql_j}{sqw_j}. \tag{1}$$

wherein ct is the time at which the current packet arrives at the leaf switch.

Counting the number of short flows

To perceive path condition and the number of short flows exist in an output queue, TR constructs a table to store two important information for each available output queue. Each entry of the table includes the obtained queue length of spine switch corresponding to the current output queue and the number of short flows exist in current output queue, called ns. The former one is updated when the leaf switch receives a reverse ACK from the spine switch, while ns is refreshed according to several conditions. Specifically, if a new flow i emerges in the corresponding output queue j, ns_j is increased by 1. If the size of data sent by flow i (fs_i) is greater than the size threshold of long flow (T_l), ns_j is decreased by 1. If fs_i is still less than T_l, but it has been a long time since the last packet arrived at the jth output queue (in TR, the duration threshold is set as one RTT), flow i is considered as inactive and ns_j is also decreased by 1. Once flow i becomes active again, while fs_i is also less than T_l, ns_j is increased by 1. By this way, TR timely understands how many short flows exist in an output queue.

Model analysis

Packet reordering greatly impairs the transmission performances of both long and short flows, especially to those packet-level schemes. TR also works at packet level, but can effectively avoid packet reordering. Therefore, in this part, we discuss the performance gain of TR in terms of avoiding packet reordering by comparing with RPS. We first analyze the impact of packet reordering on flow completion time for RPS and TR, respectively. Then, we verify our analysis by conducting the NS2 simulation tests. The parameters in our model are listed in Table 3.

Suppose that N TCP flows (each of which has S data) are synchronously transmitted in an asymmetric scenario. Assuming that there are n parallel paths in total, and they are divided into two categories, i.e., good paths and bad paths. The number of good paths is n_g, while there are n_b bad paths. The packets transmitted on the good paths always arrive at the receiver earlier than those transmitted on the bad paths. Meanwhile, the paths belonging to the same kind have the similar delay that does not cause packet reordering.

Consider a flow f belongs to one of those TCP flows. During f's each round of transmission, if the average size of congestion window is w and the end-to-end delay is r, we can calculate the flow completion time of f under RPS, F^f_{RPS}, by

$$F^f_{RPS} = \frac{S}{\frac{3\times w^2}{8}} \times \frac{w \times r}{2} = \frac{4 \times S \times r}{3 \times w}. \tag{2}$$

Table 3 Parameters used in model analysis

Parameters	Description
n	The total number of parallel paths
n_g	The number of good paths
n_b	The number of bad paths
d_g	The propagation delay of good path
d_b	The propagation delay of bad path
S (measured in packets)	The flow size
H (measured in packets)	The threshold of TCP fast retransmission
w (measured in packets)	The average size of congestion window in each transmission round under RPS
$\hat{w}$ (measured in packets)	The average size of congestion window in each transmission round under TR
r	The end-to-end delay under RPS
$\hat{r}$	The end-to-end delay under TR
p	The end-to-end propagation delay under RPS
$\hat{p}$	The end-to-end propagation delay under TR
q	The end-to-end queuing delay under RPS
$\hat{q}$	The end-to-end queuing delay under TR

The reason is that in each round of transmission, f's sender spends $\frac{w\times r}{2}$ RTTs to increase its congestion window from $\frac{w}{2}$ to w, and the total number of transmitted packets is $\frac{3\times w^2}{8}$ during this process. Then, we discuss how to get w and r.

Generally, if the packet reordering does not happen, the sender increases its congestion window until the switch buffer is full. Since there are n parallel paths, the number of queuing buffers at the switch is also n. Hence, when packet reordering does not exist, the maximum congestion window of f is $\frac{n\times B}{N}$, wherein B is the buffer size of each output port at the switch. However, once packet reordering occurs, the sender will decrease its congestion window to $\frac{n\times B}{2\times N}$. Give the probability P_{fr} of fast retransmission due to packet reordering, w can be computed by

$$w = \frac{n\times B}{N}\times(1-P_{fr}) + \frac{n\times B}{2\times N}\times P_{fr}. \quad (3)$$

Next, we discuss how to get P_{fr}. For any packet i ($1\le i\le S-H$) in f, we assume that the packets (whose number is $i-1$ in total) before i have been received in order, but i is transmitted on one of the bad paths. If the following k ($H\le k\le S-i$) packets are transmitted on the good paths, the fast retransmission will be triggered. Thus, under this circumstance, the number of possibilities of fast retransmission is $\sum_{k-H}^{S-i} C_{S-i}^{k}\times n_g^{i-1+k}\times n_b^{S-i-k+1}$, wherein n_g and n_b are the number of good and bad paths, respectively. Since i can be any packet selected from the first $S-H$ packets in all the packets of f, there also exist $S-H$ possibilities corresponding to the above case. In addition, in the typical multi-path transmission like RPS, there are n^S possible path assignments in total. Therefore, we can calculate P_{fr} by

$$P_{fr} = \frac{\sum_{i=1}^{S-H}\sum_{k-H}^{S-i} C_{S-i}^{k}\times n_g^{i-1+k}\times n_b^{S-i-k+1}}{n^S}. \quad (4)$$

Then, with Eq. (4), Eq. (3) can be rewritten by

$$w = \frac{n\times B}{N}\times\left(1-\frac{\sum_{i=1}^{S-H}\sum_{k-H}^{S-i} C_{S-i}^{k}\times n_g^{i-1+k}\times n_b^{S-i-k+1}}{n^S}\right) + \frac{n\times B}{2\times N}\times\frac{\sum_{i=1}^{S-H}\sum_{k-H}^{S-i} C_{S-i}^{k}\times n_g^{i-1+k}\times n_b^{S-i-k+1}}{n^S}. \quad (5)$$

In addition, the end-to-end delay in DCN is mainly composed of the end-to-end propagation delay and the end-to-end queuing delay. Given the average congestion window w, the probability that all the packets in w does not select the bad path can be calculated by $\left(\frac{n_g}{n}\right)^w$. Accordingly, the probability that at least one packet in w selects the bad path is $1-\left(\frac{n_g}{n}\right)^w$. Thus, the average end-to-end propagation delay can be computed by $\left(1-\left(\frac{n_g}{n}\right)^w\right)\times d_b + \left(\frac{n_g}{n}\right)^w\times d_g$, wherein d_g and d_b are the propagation delays of good and bad paths, respectively. Besides, since there are N flows that share n queuing buffers, and each of them possesses the average congestion window of w, the average queuing delay can be expressed as $\frac{N\times w\times t}{n}$, wherein t is the transmission delay of each packet. Then, r can be calculated by

$$r = \left(1-\left(\frac{n_g}{n}\right)^w\right)\times d_b + \left(\frac{n_g}{n}\right)^w\times d_g + \frac{N\times w\times t}{n}. \quad (6)$$

Finally, with Eqs. (2), (5), and (6), we obtain the flow completion time of f under RPS as

$$F_{RPS}^{f} = \frac{4\times S\times\left(\left(1-\left(\frac{n_g}{n}\right)^w\right)\times d_b + \left(\frac{n_g}{n}\right)^w\times d_g + \frac{N\times w\times t}{n}\right)}{3\times w}. \quad (7)$$

Since TR can effectively avoid packet reordering, flow f does not experience packet retransmission unless packets are dropping by switch, which means that the network is congested. Therefore, during $f's$ transmission under TR, the maximum congestion window is $\frac{n\times B}{N}$, and the average congestion window $\hat{w}$ is $\frac{3\times n\times B}{4\times N}$. As to the end-to-end delay $\hat{r}$ under TR, there are two possibilities, i.e., the first packet of f selects the good path or the bad path. In this model scenario, the worst case is the latter case, which means that all the packets of f will select the bad paths, resulting in the maximum end-to-end delay. For simplicity, we calculate $f's$ flow completion time F_{TR}^{f} under the worst case of TR. Then, based on Eq. (2), we get

$$\begin{aligned} F_{TR}^{f} &= \frac{4\times S\times\left(d_b + \frac{\frac{N\times 3\times n\times B\times t}{4\times N}}{n}\right)}{3\times\frac{3\times n\times B}{4\times N}} \\ &= \frac{S\times N\times(16\times d_b + 12\times B\times t)}{9\times n\times B}. \end{aligned} \quad (8)$$

Finally, we conduct both model and NS2 tests based on the scenario in Design motivation section to verify our analysis. In Fig. 6(a), the number of flows is set as 20, and the data size of flow f is gradually increased from 100KB to 350KB. The results show that, under both RPS and TR, f takes more time to finish its transmission as its data size is enlarged. In Fig. 6(b), the data size of flow f is fixed to 300KB, and we introduce more flows to make tests. Flow f still spends more and more time accomplishing its transmission since its available bandwidth becomes scarcer as increasing the number of flows. Nonetheless, with the help of out-of-order prediction, TR always outperforms RPS due to effectively avoiding packet retransmission. Overall, the results of simulation are basically close to the corresponding theoretical values, verifying the above analysis.

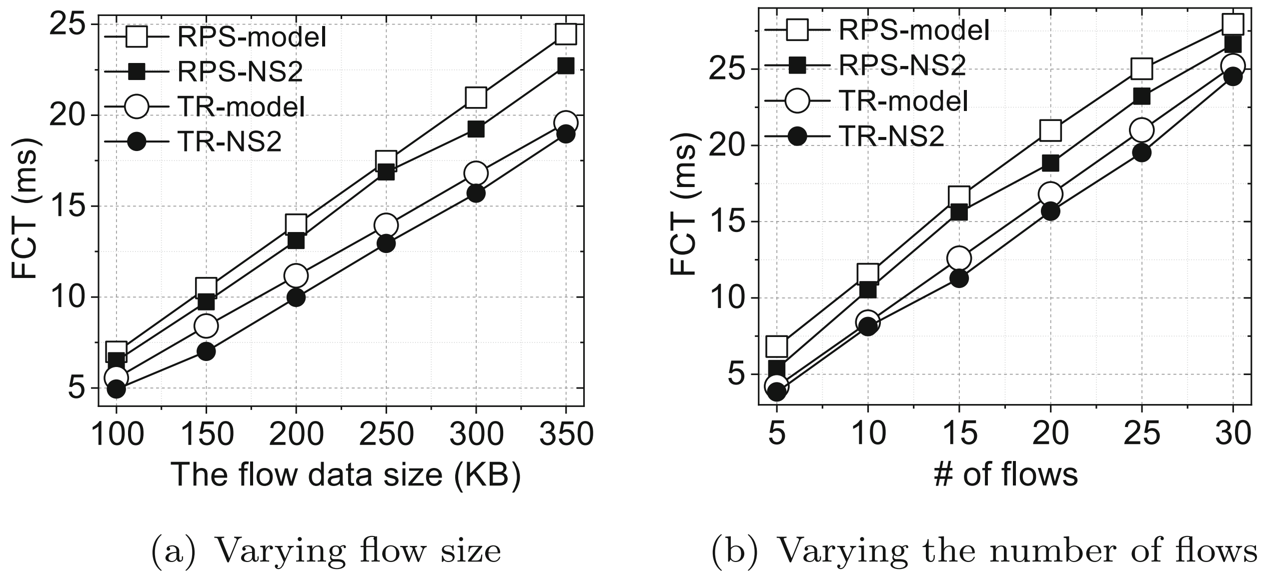


Fig. 6 Model verification

Implementation

The existing common Commercial Off The Shelf (COTS) switch has limited scalability and can not support relatively complex schemes (e.g., PIAS [20]), due to its poor reconfigurability and limited on-chip memory [20, 29, 30]. Similarly, TR is also hard to deploy on the existing common COTS switch directly. Fortunately, the programmable switches that can realize complex processing logic are becoming more commonplace [30]. TR can be deployed on these programmable switches, such as the Tofino switch, which is a kind of end-user Ethernet switch with powerful programmability, and built using a P4-programmable Protocol Independent Switch Architecture (PISA) [31]. With the programmable switch, the new forwarding logic does not have to be baked into the silicon, but resides in the P4 program that provides the logic for handling all supported protocols [31]. When deploying TR, the network operator or switch manufacturer can easily add the TR logic to the P4 program.

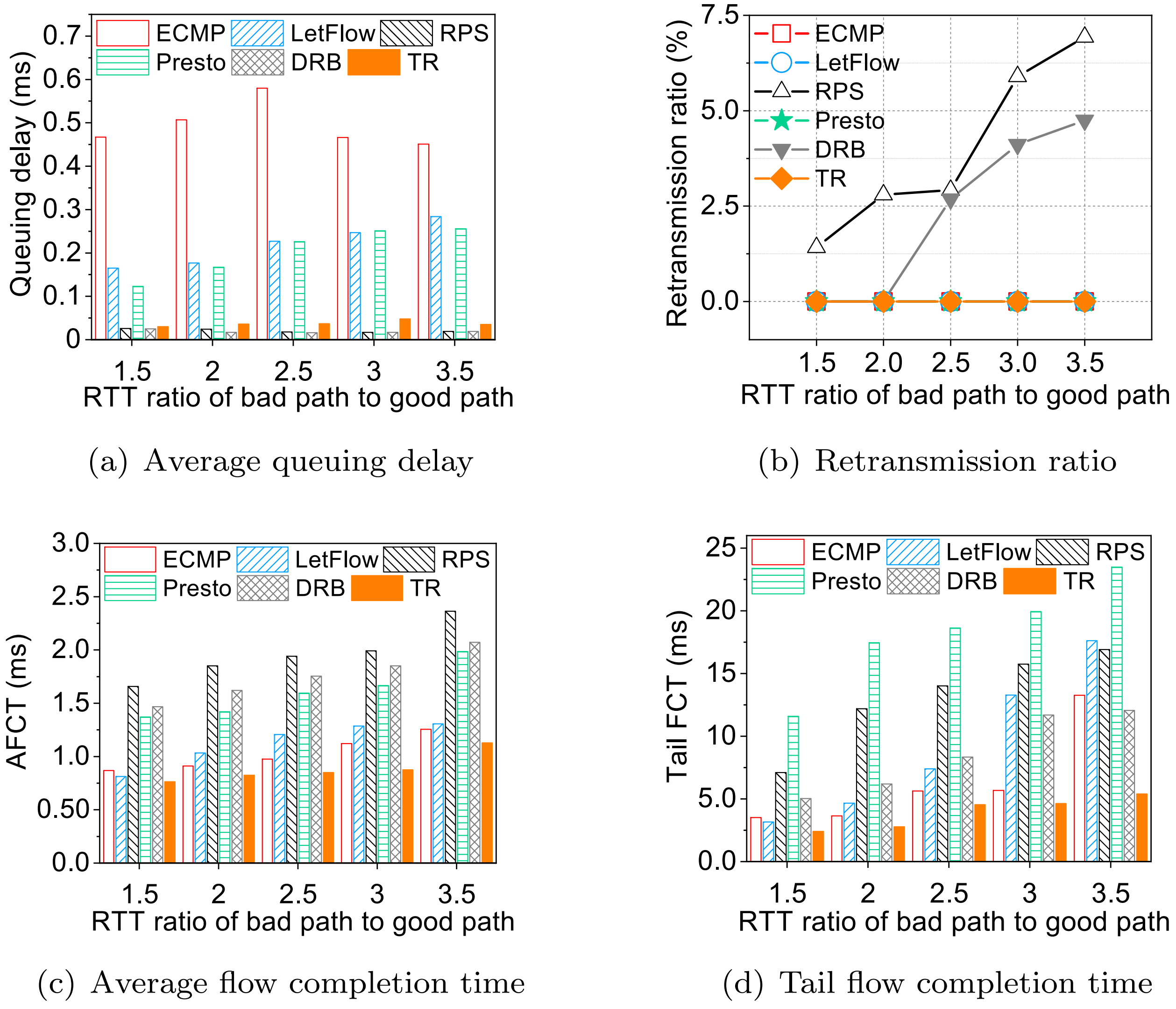
TR needs to track the per-flow state, including counting the bytes of data sent and comparing the flow sizes. Fortunately, these two requirements have already been met by the current programmable switches [29–33]. Meanwhile, there also exist many sketch-based methods to decrease the overhead for network measurement tasks [31, 34], and TR can combine them to further reduce the implementation overhead during deployment.

Performance evaluation

In this section, we conduct numerous NS2 simulation tests and introduce another two state-of-the-art data center load balancing schemes, i.e., DRB[13] and Presto [14], to evaluate the performance of TR. We also install four typical datacenter workloads including Data Mining, Web Search, Cache Follower, and Web Server to make a comprehensive evaluation. The threshold of flowlet emerging is set to 500μs in LetFlow[8], and the data size of flowcell is 64KB for Presto[14]. DCTCP is used as the congestion control scheme at TCP senders, and the initial TCP window size is set to 10 packets.

Micro-Benchmark

Firstly, we redo the micro-benchmark in Design motivation section to observe whether TR performs as expected. Figure 7(a) compares the average queuing delay of short flows under different schemes. Since the coarse-grained ECMP and LetFlow lead to the unevenly load on each path, their short flows experience larger queuing delay. Although Presto is medium-grained (flow-cell based), its queuing delay of short flows is still non-trivial. By contrast, the finer-grained schemes, including RPS, DRP, and TR, have much lower queuing delay across all cases. In Fig. 7(b), however, the rising of asymmetric degree causes RPS and DRB to experience increasingly serious packet reordering, generating much higher retransmission ratio compared to the other schemes. TR also works at packet level, but its retransmission ratio is always 0. The reason is that TR can filter out those output queues potentially cause packet reordering before forwarding each packet. Consequently, RPS performs much worse than the other schemes in terms of short flows' AFCT. DRB is also troubled by packet reordering, and its performance is not good either. On the contrary, TR can keep queuing delay at low level, and effectively control packet reordering, as well as



(a) Average queuing delay (b) Retransmission ratio (c) Average flow completion time (d) Tail flow completion time

Fig. 7 The performances of short flows

allow long flows to avoid short flows for further speeding up the short flow's data transfer, thus always obtaining the shortest average and tail flow completion times compared to the other schemes across all cases, as shown in Fig. 7(c) and (d).

On the other hand, the long flows' performance is closely related to packet retransmission. As shown in Fig. 8, RPS and DRB can distribute load in the most balanced way, but possess high retransmission ratio, which in turn leads to the lowing of total throughput of long flows as increasing the asymmetric degree. ECMP and LetFlow can completely avoid packet reordering, while fail to fully utilize all parallel paths. As a whole, though only TR and Presto can cover both sides, TR still outperforms Presto. The reason is that TR carries out traffic-aware rerouting and is aware of path station, especially effectively avoids packet reordering. Therefore, it is not affected by the rising of asymmetric degree, and always maintains the highest total throughput for long flows compared to the other schemes.

Threshold discussion

TR employs a threshold to identify flow types, and then carries out different operations for the short flows and long flows, respectively. However, if the value of threshold is improper, TR's performance will be probably impaired. Therefore, in this part, we evaluate the performance of TR under five different thresholds (i.e., 10KB, 50KB, 100KB, 500KB, and 1MB). To make a comprehensive comparison, we install four typical realistic workloads, including a Data Mining workload, a Web Search workload, a Cache Follower workload and a Web Server workload [19]. The

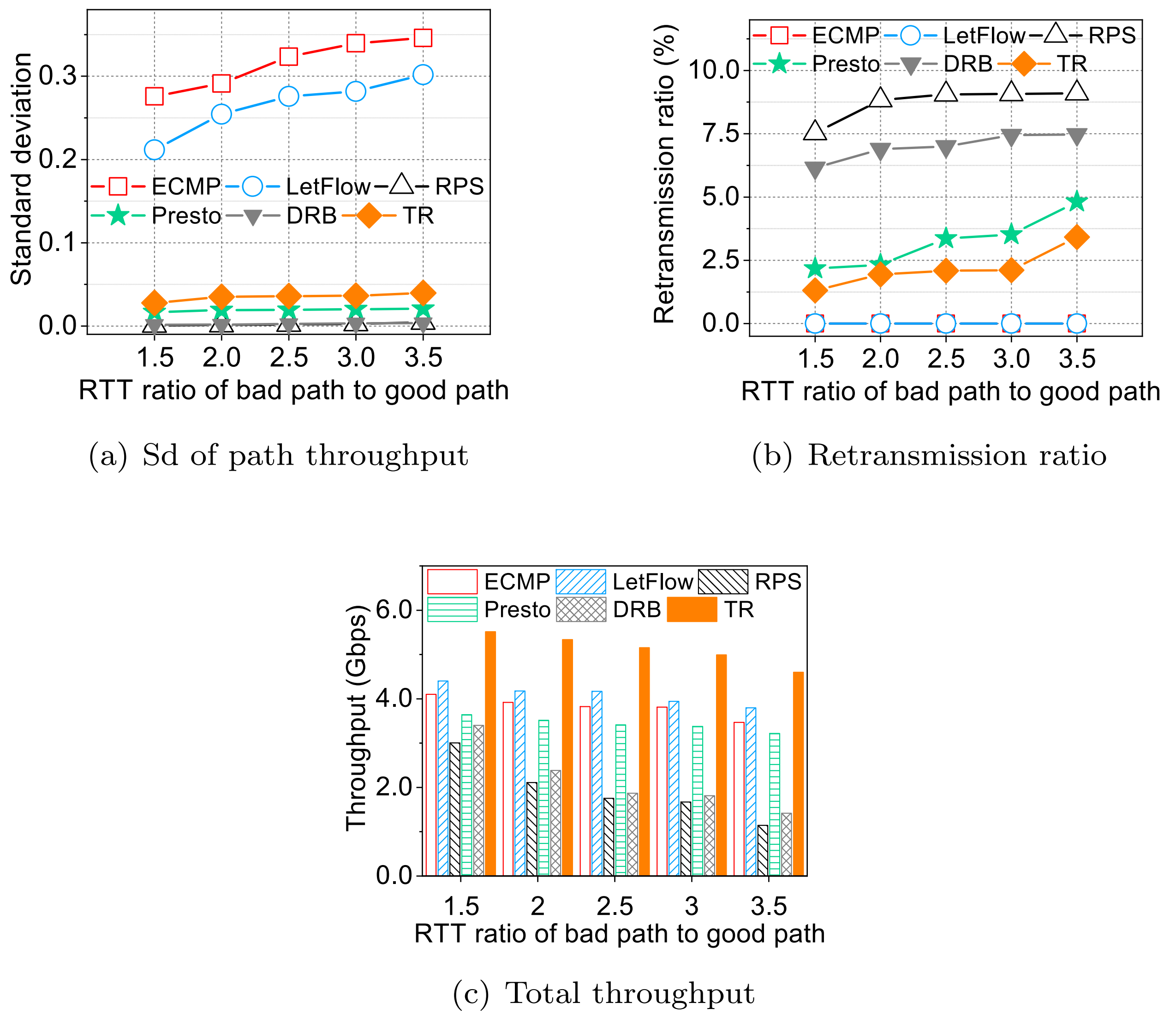


Fig. 8 The performances of long flows

specific proportions of flows belonging to different size ranges in each workload are shown in Table 1.

Web Server has the most short flows (<100KB) and the fewest long flows (>100KB) compared to the other workloads. Meanwhile, the data sizes of long flows in the Web Server workload are all smaller than 1MB. Cache Follower is just the opposite for which has the most long flows and the fewest short flows. In the Data Mining workload, the proportion of extremely short flows (<10KB) is the largest. Except the Web Server workload, the proportions of short flows whose data size is between 10KB and 100KB in the other workloads are very low (5% at most). Overall, across four workloads, their proportions of short flows are always higher than those of long flows, following the heavy-tail distribution of data center traffic [16–18, 20].

We continue to use the asymmetric scenario described in the section of "Design motivation", and change the link bandwidth to 10Gbps. We increase the link propagation delay of bad path so that the ratio of bad path's propagation RTT is 3.5x times the good path's propagation RTT. The hosts in Group 1 send 1000 DCTCP flows to the hosts in Group 2 by following a Poisson process. These flows are generated based on four realistic data center workloads mentioned before. Besides, according to the method described in Ref. [35], we maintain the network load at 0.3 since most DCNs operate at this load [17, 20].

Figure 9 presents the normalized average FCTs and tail FCTs of all flows through using the results of threshold of 100KB as the baseline. In all cases, we observe that both the average FCTs and tail FCTs are increased when the threshold becomes very large or small. The

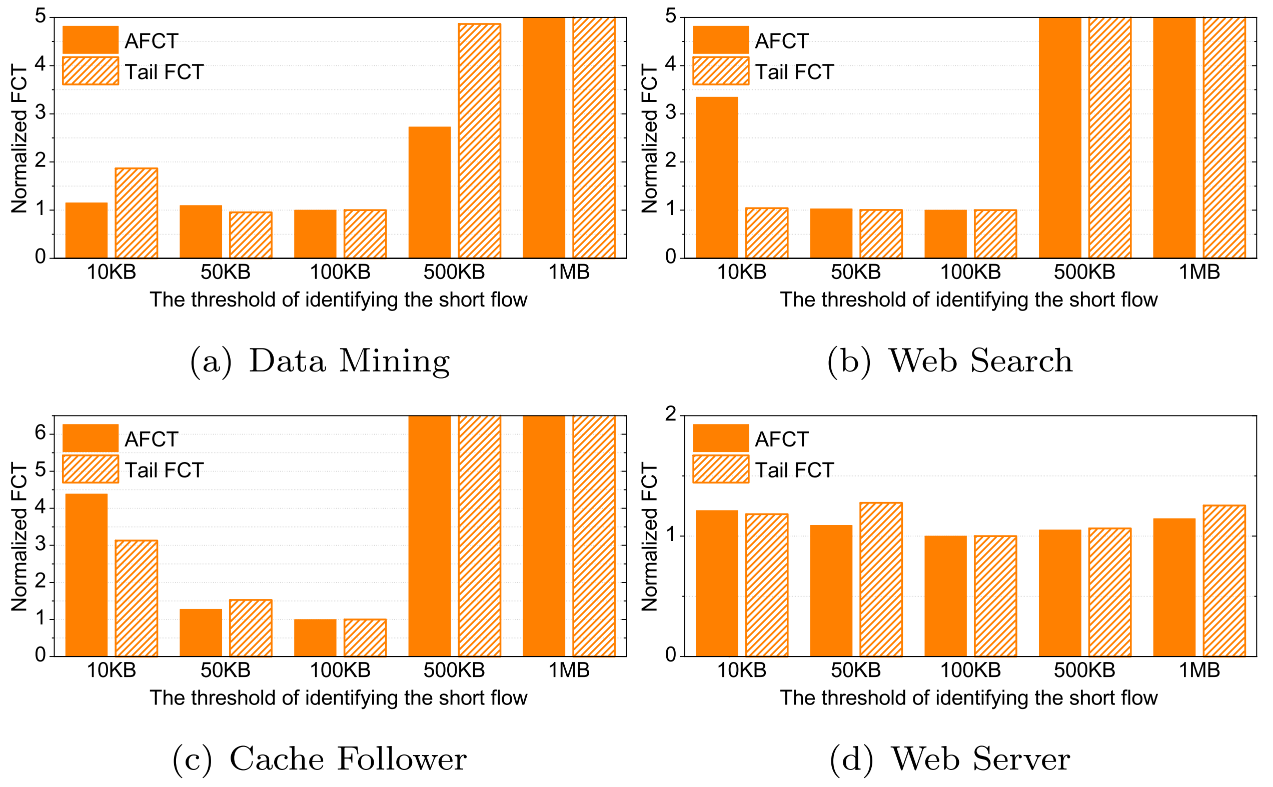


Fig. 9 The impact of thresholds

main reasons are as follows. If the threshold is too large, some long flows are mistakenly identified as short ones, resulting in the sharp increase of queueing delay of short flows. Thus, the performance of short flows is seriously degraded. Conversely, under a too small threshold, some short flows are mistakenly distinguished as long ones. The short flows handled as long ones probably fail to flexibly utilize multiple paths to finish quickly.

Large-scale evaluation

In this section, we construct a large-scale leaf-spine network in which 200 hosts are connected via 10 leaf switches, 10 spine switches, and many 40Gbps links. The switch buffer size is set to 375KB [19]. To generate asymmetry, we randomly choose half of the parallel paths and consider them as the good paths, and the remained paths are converted into the bad paths by increasing their links' propagation delay. Consequently, the round-trip propagation delay of bad path is 1.5x times the round-trip propagation delay of good path.

During the whole test, 100 hosts are randomly selected to send 50000 DCTCP flows to the remained hosts. All flows start by following a Poisson process. We vary the load intensity from 0.2 to 0.8 in each test by following the method described in Ref. [35], and continue to install the representative data center workloads mentioned before. Their respective flow size distributions are shown in Fig. 10.

To make a comprehensive performance comparison, we divide all flows into three classes according to different flow size ranges, including (0,100KB], (100KB,1MB], and (1MB,∞) [19, 20]. We compute and compare the average FCT for all flows, the average FCT of flows in each class, and the tail FCT of flows belonging to (0,100KB]. All results are normalized to those achieved by LetFlow (the normalized FCTs for LetFlow are always 1).

For the short flow whose data sizes are in (0,100KB], we find that TR performs better than LetFlow, and greatly outperforms ECMP, Presto, DRB, and RPS. For example, when compared to LetFlow, TR improves the average FCTs for short flows across four workloads by around 20 percent, as shown in Figs. 11(a), 12(a), 13(a), and 14(a). Moreover, the tail FCTs of short flows under four workloads are also improved by up to 50 percent, as shown in Figs. 11(b), 12(b), 13(b), and 14(b). This is because that TR is aware of the different flow requirements and real-time path conditions.

In addition, for the flows whose data sizes are between 100KB and 1MB, TR still manages to outperform LetFlow by about 20 percent in average for both Data Mining workload and Web Search workload. Although the advantage of TR slightly diminishes when it comes to deal with the

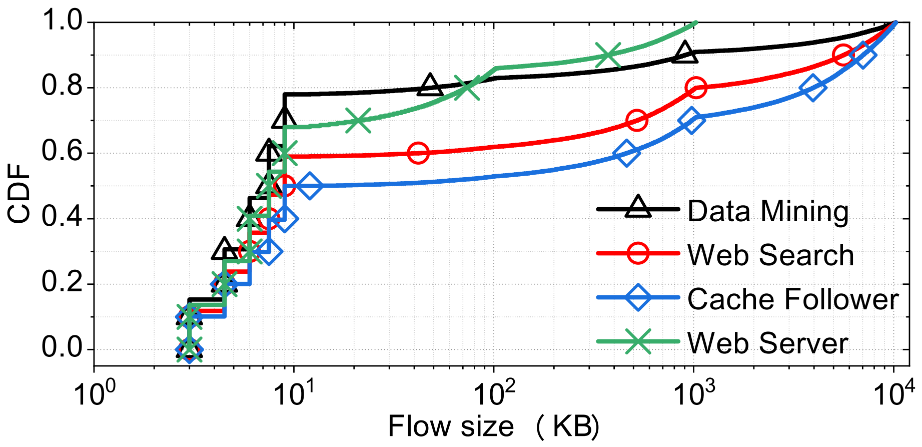


Fig. 10 The CDFs (Cumulative Distribution Functions) of flow size under realistic workloads

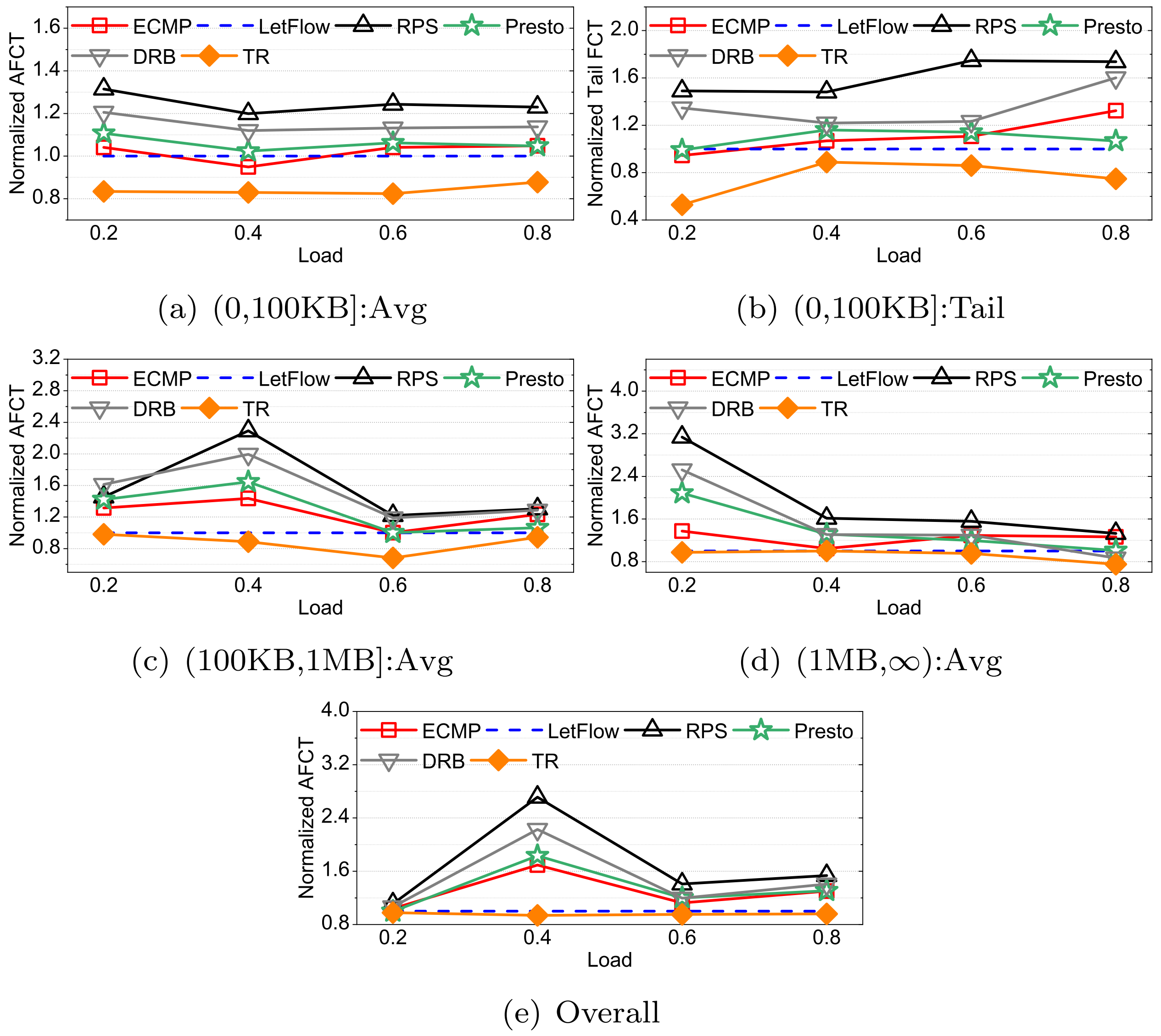
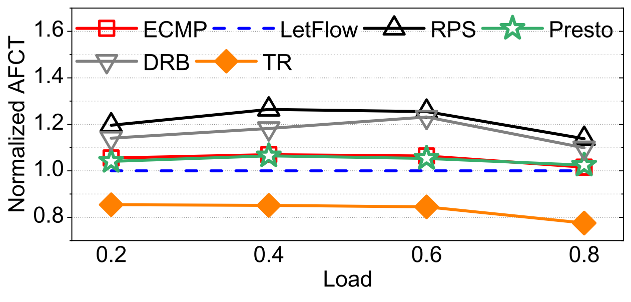
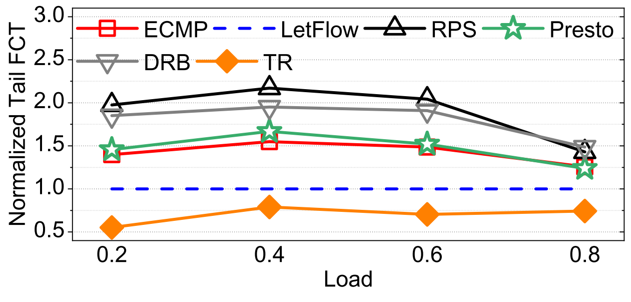


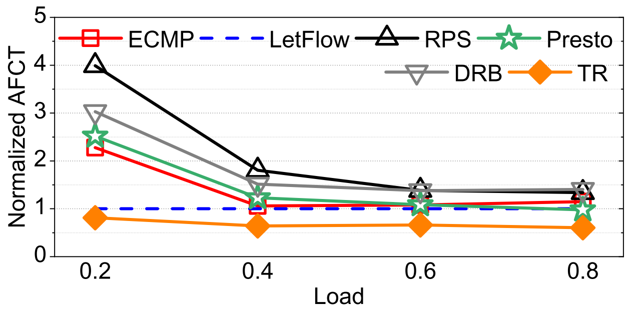
Fig. 11 Data mining workload



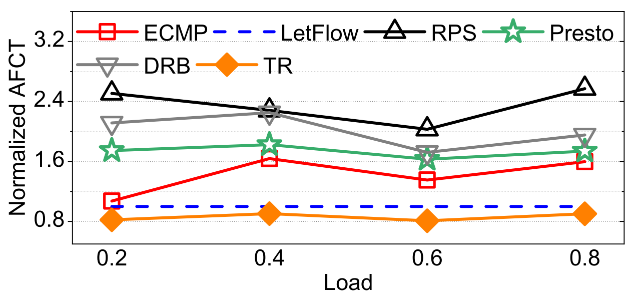
(a) (0,100KB]:Avg



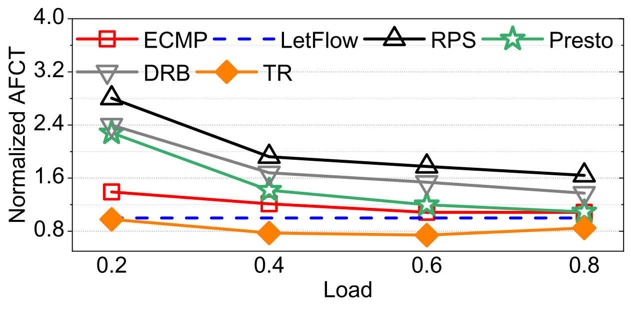
(b) (0,100KB]:Tail



(c) (100KB,1MB]:Avg



(d) (1MB,∞):Avg



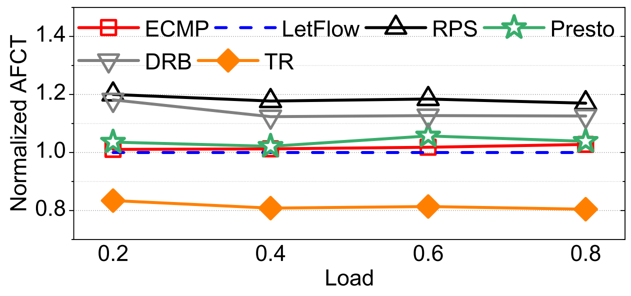
(e) Overall

Fig. 12 Web search workload

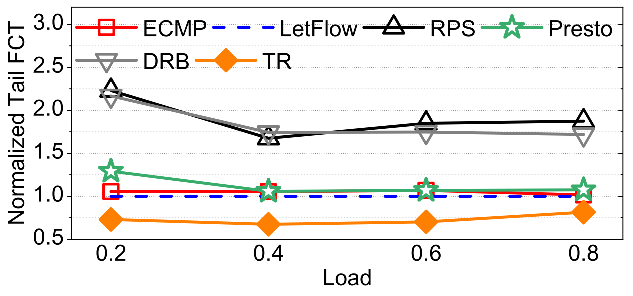
Cache Follower workload and the Web Server workload, the improvement is still up to 20 percent, as shown in Figs. 11(c), 12(c), 13(c), and 14(c).

As to those flows larger than 1MB, TR slightly outperforms LetFlow, and the improvement is about 10 percent, as shown in Figs. 11(d), 12(d), and 13(d). Although the delay-sensitive short flows dominate the data traffic in DCN and TR gives them more priorities, the performance of the other flows can still be guaranteed as much as possible since TR effectively alleviates packet reordering and helps these flows flexibly utilize multiple paths to finish quickly. Note that there does not exist the flows larger than 1MB in the Web Server workload (see Table 1), hence the corresponding results are not presented.

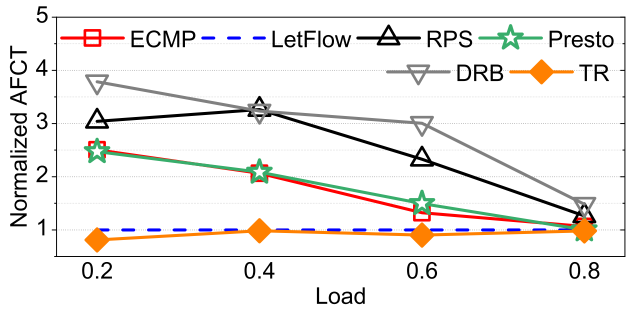
Finally, we observe the overall performance of all flows. In Figs. 11(e), 12(e), 13(e), and 14(d), we find that the performances of fine-grained schemes, such RPS and DRB, are always worse than the other schemes. This is mainly caused by plenty of retransmission packets since in the asymmetric network, flows are very prone to experience packet reordering. By contrast, DRB performs a little better than RPS, which is also because that DRB is slightly better at controlling packet reordering compared to RPS. For TR, its mechanism of out-of-order prediction effectively controls the packet reordering, greatly reducing the packet retransmissions. Meanwhile, it can timely perceive the change of path condition, and legitimately assign parallel paths to short and long flows according to their



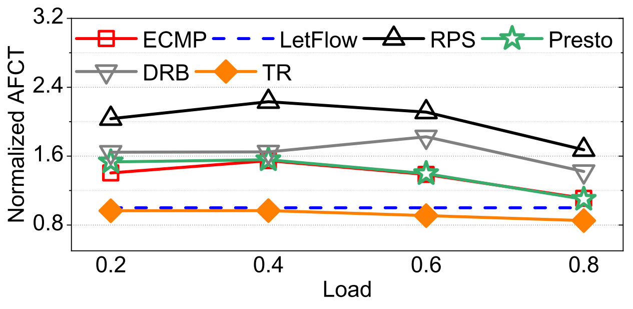
(a) (0,100KB]:Avg



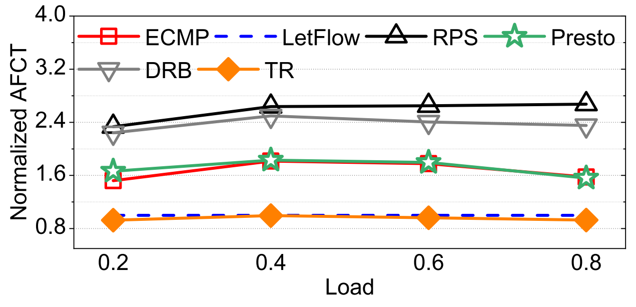
(b) (0,100KB]:Tail



(c) (100KB,1MB]:Avg



(d) (1MB,∞):Avg



(e) Overall

Fig. 13 Cache follower workload

different performance requirements. As a consequence, its overall performance is better than those of the other schemes under different load intensities.

Implementation overhead

In this part, we run some tests based on Mininet [36] to show the implementation overhead of TR. Mininet is a network emulation system with high fidelity on Linux kernel [36], and its behavior is similar to the real hardware elements [37]. With the limitation of single-machine CPU, Mininet only supports tens of Mbps link bandwidth, and has smaller test scale compared to the real DCNs [18]. However, in view that its codes and test scripts can be deployed on a real network scenario, Mininet is widely used as a flexible testbed for networking experiments [37].

In the test, we implement the packet processing pipeline of TR with P4 program ($P4_{16}$ 1.0) [38], and use Mininet 2.3.0 to create a leaf-spine network in which two leaf switches and ten spine switches are connected via many links. The bandwidth of each link is set to 20Mbps as recommended in Ref. [18]. We install BMv2 to generate the software programmable switches, and the switch buffer size is set as 256 packets [18]. Each leaf switch connects to 10 servers, and there are ten equal-cost paths in total between two leaf switches. To create the topology asymmetry, five equal-cost paths are randomly chosen as the good paths, while the remained paths are considered as the bad paths. The round trip propagation delay of good path is 1ms, while the corresponding delay of bad path is set to 4ms. We generate 100 DCTCP flows with 90% short

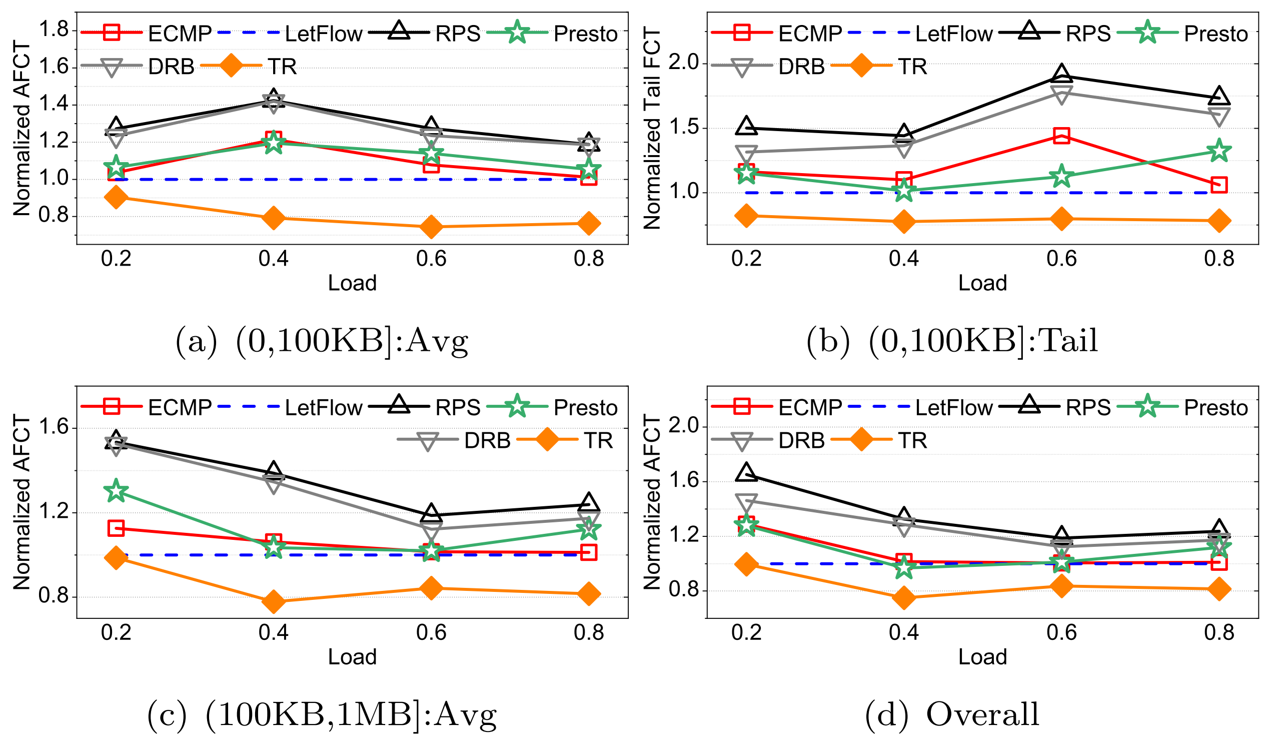


Fig. 14 Web server workload

flows and 10% long flows by following a Poisson process [17]. The network load is also varied from 0.2 to 0.8 in each test according to the method described in Ref. [35].

To evaluate the system overhead of TR, we measure the maximum, minimum and average CPU and memory utilizations at the leaf switch. In Fig. 15(a), since RPS and DRB simply spray all the packets to all the equal cost paths, their CPU utilizations are very low. For ECMP, LetFlow, and Presto, due to their simple operations at switches, their CPU utilizations are also relatively low. Also, TR does not incur excessive CPU overhead to switch compared with the other schemes. The reason is that the computing overhead of TNL can be greatly decreased by some pretreatment operations. For example, the calculation of queuing delay when computing TNL can be simplified through table look-up, which only generates a tiny fraction of CPU load. On the other hand, Fig. 15(b) shows that even at 80% load, TR's memory utilization is only around 5%. Therefore, compared with the performance gain of TR, its system overhead, on the whole, is acceptable.

Related works

In recent years, researchers have designed plenty of transport protocols to improve the transmission performance of datacenter networks [1–6, 18, 21, 39–45]. Nonetheless, with the sharp increase of network capacity, various load balancing mechanisms have also been proposed to facilitate parallel data transmission across multiple paths, thus further obtaining performance enhancements.

ECMP [12] leverages several fields in packet headers to calculate a hash value, which is mapped to one of the equal-cost paths. However, the key problem of ECMP is that the flow-to-path assignment is static and easily causes congestion due to the hash collisions of large flows. Since the congested flows can not be rerouted to the paths with low utilization, it inevitably degrades the network performance. To address this problem, various fine-grained mechanisms are designed to split traffic across multiple paths. These fine-grained solutions can be roughly classified into two categories: per-packet and per-flowlet/flowcell mechanisms.

RPS [11] is an intuitive and simple per-packet scheme, which randomly allocates one available path for each packet. DRILL [46] implements a random packet allocation mechanism based on switch local information. Specifically, when a switch receives one packet, it randomly picks two available ports and compare their queue length with a recorded port. Then the switch chooses the port with the lowest buffer to forward the packet. These per-packet solutions can achieve high network utilization and near-optimal tail latency in symmetric topologies. However, in production data center, there are a multitude of uncertainties such as highly dynamic traffic and

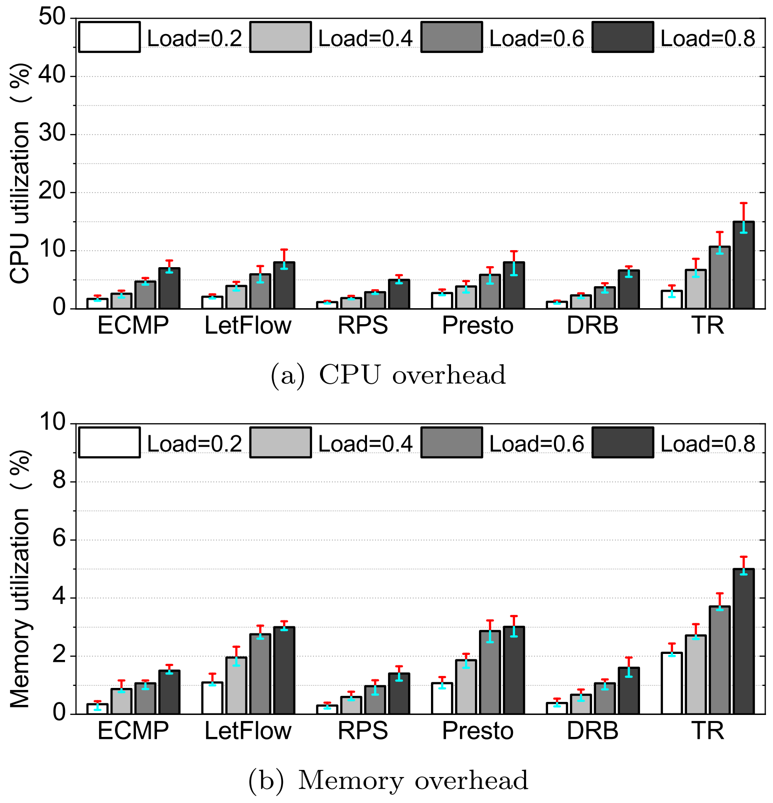


Fig. 15 Implementation overhead

link/switch failures [7, 47], which inevitably makes network topology become asymmetric and results in serious packet reordering.

MMPTCP [21] initially scatters packets to exploit all available paths and increases fast retransmission threshold to handle packet reordering. Then MMPTCP switches to the MPTCP [43] when the amount of data transmitted by sender is larger than a given threshold. JUGGLER [48] leverages the re-sequence buffer at the receiver to absorb out-of-order packets, which are delivered to upper layer when the buffer is full or a timer expires. However, setting a suitable threshold value is not trivial, especially in data center networks with highly dynamic traffic. APS [49] adopt isolate mechanism to achieve adaptive packet spraying. QDAPS [50] is a delay-aware mechanism. When a packet arrives at the switch, it is assigned to the output port, which has larger queueing delay than the recorded port that forwards the last arrival packet in the same flow. Unfortunately, if one packet is assigned to a congested path, then subsequent packets in the same flow will be assigned to one more congested path, potentially increasing flow completion time.

Furthermore, researchers design lots of per-flowlet solutions such as AMR [51] and LetFlow [8] to achieve the tradeoff between packet reordering and network performance. The basic idea of per-flowlet solutions is that for each flow, when the time interval between two adjacent packets is larger than a predetermined threshold, it suggests that the transmission path becomes congested. Then, the switch either shifts traffic to less-congested paths or picks a path at random for subsequent packets of the flow. For example, both MLAB [52] and CONGA [9] are congestion-aware approaches that leverage path congestion to achieve better load balancing.

Presto [14] splits data of each flow into equal-size units called flowcell, whose default size is set to 64KB. Since Presto is insensitive to path conditions, it adopts round-robin way to blindly assign one path for each flowcell. Moreover, since Presto has to record all flows' states, it may cause overhead issue. Luopan [53] is a congestion-aware method. It periodically samples some paths and picks the least congested path for new flowcells. Given the that most flows in data center are smaller than flowcell, they fail to take advantage of multiple paths, which may cause low link utilization.

Conclusion

This paper presents the design and analysis of a fine-grained load balancing scheme TR to simultaneously improve the transmission performances of both short and long flows in data center networks. To avoid packet reordering, TR leverages the reverse ACKs to quickly feedback the queue length of spine switch, so that the source leaf switch can predict the delay from itself to destination leaf switch and excludes paths that potentially cause packet reordering. Moreover, TR uses flexible switching granularity and rerouting decision to make a good trade-off among queuing delay, packet reordering and link utilization. Although working at packet level, TR can be applied to both symmetric and asymmetric scenarios, while is only deployed on the switch without any modification on end-hosts. We evaluate TR through large-scale NS2 simulation tests under a wide range of data center workloads. The experimental results show that TR reduces the average and tail flow completion time for short flows by up to 60% and 80%, as well as provides 3.02x gain in throughput of long flows over the state-of-the-art load balancing schemes, respectively.

Acknowledgements
We would like to thank Dr. Chang Ruan for his valuable help in evaluation.

Authors' contributions
All authors have participated in conception and design, or analysis and interpretation of this paper.

Funding
This work is supported in part by the National Natural Science Foundation of China under Grant 61872403 and Grant 61772088; in part by the Hunan Province Key Laboratory of Industrial Internet Technology and Security under Grant 2019TP1011; in part by the Natural Science Foundation of Hunan Province under Grant 2020JJ6064.

Availability of data and materials
Not applicable

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 28 March 2021 Accepted: 22 June 2021
Published online: 06 July 2021

References

1. Kumar G, Dukkipati N, Jang K, Wassel H. M. G, Wu X, Montazeri B, Wang Y, Springborn K, Alfeld C, Ryan M, Wetherall D, Vahdat A (2020) Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. Proc ACM SIGCOMM:514–528
2. Saeed A, Gupta V, Goyal P, Sharif M, Pan R, Ammar M, Zegura E, Jang K, Alizadeh M, Kabbani A, Vahdat A (2020) Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates. Proc ACM SIGCOMM:735–749
3. Hu S, Bai W, Zeng G, Wang Z, Qiao B, Chen K, Tan K, Wang Y (2020) Aeolus: A Building Block for Proactive Transport in Datacenters. Proc ACM SIGCOMM:1–13
4. Zhang T, Huang J, Chen K, Wang J, Chen J, Pan Y, Min G (2020) Rethinking Fast and Friendly Transport in Data Center Networks. IEEE/ACM Trans Networking 28(5):2364–2377
5. Zeng G, Bai W, Chen G, Chen K, Han D, Zhu Y, Cui L (2019) Congestion control for cross-datacenter networks. Proc IEEE ICNP:1–12
6. Alizadeh M, Greenberg A, Maltz D. A, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M (2010) Data Center TCP (DCTCP). Proc ACM SIGCOMM:63–74
7. Zhang H, Zhang J, Bai W, Kai C, Chowdhury M (2017) Resilient datacenter load balancing in the wild. Proc ACM SIGCOMM:253–266
8. Vanini E, Pan R, Alizadeh M, Taheri P, Edsall T (2017) Let it flow: resilient asymmetric load balancing with flowlet switching. Proc USENIX NSDI:407–420
9. Alizadeh M, Edsall T, Dharmapurikar S, Vaidyanathan R, Chu K, Fingerhut A, Lam V. T, Matus F, Pan R, Yadav N, Varghese G (2014) CONGA: Distributed congestion-aware load balancing for datacenters. Proc ACM SIGCOMM:503–514
10. Michelogiannakis G, Ibrahim K. Z, Shalf J, Wilke J. J, Knight S, Kenny J. P (2017) APHiD: Hierarchical Task Placement to Enable a Tapered Fat Tree Topology for Lower Power and Cost in HPC Networks. IEEE/ACM CCGRID:228–237
11. Dixit A, Prakash P, Hu Y. C, Kompella R. R (2013) On the impact of packet spraying in data center networks. Proc IEEE INFOCOM:2130–2138
12. Hopps C. E (2000) Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992
13. Cao J, Xia R, Yang P, Guo C, Lu G, Yuan L, Zheng Y, Wu H, Xiong Y, Maltz D (2013) Per-packet Load-balanced," Low-latency Routing for Clos-based Data Center Networks. Proc ACM CoNEXT:49–60
14. He K, Rozner E, Agarwal K, Felter W, Carter J, Akellay A (2015) Presto: edge-based load balancing for fast datacenter networks. Proc ACM SIGCOMM:465–478
15. Zhou L, Chou C, Bhuyan L. N, Ramakrishnan K. K, Wong D (2018) Joint Server and Network Energy Saving in Data Centers for Latency-Sensitive Applications. IEEE IPDPS:700–709
16. Ye J, Ma L, Pang C, Xiao Q, Jiang W (2020) Inferring Coflow Size based on Broad Learning System in Data Center Network. Proc IEEE ICCT:903–907
17. Benson T, Akella A, Maltz D (2010) Network traffic characteristics of data centers in the wild. Proc ACM IMC:267–280
18. Xu H, Li B (2014) RepFlow: Minimizing Flow Completion Times with Replicated Flows in Data Centers. Proc IEEE INFOCOM:1581–1589
19. Cho I, Jang K, Han D (2017) Credit-scheduled delay-bounded congestion control for datacenters. Proc ACM SIGCOMM:239–252
20. Bai W, Chen L, Chen K, Han D, Tian C, Wang H (2015) Practical Information-Agnostic Flow Scheduling in Data Center Networks. Proc USENIX NSDI:455–468
21. Kheirkhah M, Wakeman I, Parisis G (2016) MMPTCP: A multipath transport protocol for data centers. Proc IEEE INFOCOM:1–9
22. Hu J, Huang J, Lv J, Zhou Y, Wang J, He T (2018) CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center. Proc IEEE INFOCOM:2294–2302
23. Hong C. Y, Caesar M, Godfrey P. B (2012) Finishing flows quickly with preemptive scheduling. Proc ACM SIGCOMM:127–138
24. Zats D, Das T, Mohan P, Borthakur D, Katz R (2012) DeTail: Reducing the flow completion time tail in datacenter networks. Proc ACM SIGCOMM:139–150
25. Alizadeh M, Yang S, Sharif M (2013) pFabric: Minimal near-optimal datacenter transport. Proc ACM SIGCOMM:435–446
26. Munir A, Qazi I. A, Uzmi Z. A, Mushtaq A, Ismail S. N, Iqbal M. S, Khan B (2013) On the impact of packet spraying in data center networks. Proc IEEE INFOCOM:2157–2165
27. Chen L, Chen K, Bai W, Alizadeh M (2016) Scheduling mix-flows in commodity datacenters with karuna. Proc ACM SIGCOMM:174–187
28. Wang T, Wang L, Hamdi M (2018) A Cost-effective Low-latency Overlaid Torus-based Data Center Network Architecture. Comput Commun 129:89–100
29. Sivaraman A, Kim C, Krishnamoorthy R, Dixit A (2015) DC.p4: Programming the Forwarding Plane of a Data-Center Switch. Proc ACM SOSR:1–8
30. Jose L, Yan L, Varghese G, McKeown N (2015) Compiling Packet Programs to Reconfigurable Switches. Proc USENIX NSDI:103–115
31. Sharma N. Kr, Liu M, Atreya K, Krishnamurthy A (2018) Approximating Fair Queueing on Reconfigurable Switches. Proc USENIX NSDI:1–16
32. Qu T, Joshi R, Chan M. C, Leong B, Guo D, Liu Z (2019) SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks. Proc IEEE ICNP:1–12
33. Zhao Z, Shi X, Yin X, Wang Z, Li Q (2019) HashFlow for Better Flow Record Collection. Proc IEEE ICDCS:1–12
34. Liu Z, Basat R. B, Einziger G, Kassner Y, Braverman V, Friedman R, Sekar V (2019) Nitrosketch: robust and general sketch-based monitoring in software switches. Proc ACM SIGCOMM:334–350

35. Gao C, Lee VCS, Li K (2021) D-SRTF: Distributed Shortest Remaining Time First Scheduling for Data Center Networks. IEEE Trans Cloud Comput 9(2):562–575
36. Handigol N, Heller B, Jeyakumar V, Lantz B, McKeown N (2012) Reproducible network experiments using container-based emulation. Proc CoNEXT:253–264
37. khurshid A, Zou X, Zhou W, Caesar M, Godfrey P. B (2012) Veriflow: Verifying network-wide invariants in real time. ACM SIGCOMM Comput Commun Rev 42(4):467–472
38. Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexflor J, Schlesinger C, Talayco D, Vahdat A, Varghese G, Walker D (2014) P4: Programming Protocol-Independent Packet Processors. ACM SIGCOMM Comput Commun Rev 44(3):87–95
39. He K, Rozner E, Agarwal K, Gu Y, Felter W, Carter J, Akella A (2016) AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. Proc ACM SIGCOMM:244–257
40. Cronkite-Ratcliff B, Bergman A, Vargaftik S, Ravi M, McKeown N, Abraham I, Keslassy I (2016) Virtualized Congestion Control. Proc ACM SIGCOMM:230–243
41. Lee C, Park C, Jang K, Moon S, Han D (2017) DX: Latency-Based Congestion Control for Datacenters. IEEE/ACM Trans Networking 25(1):335–348
42. Mittal R, Lam V. T, Dukkipati N, Blem E, Wassel H, Ghobadi M, Vahdat A, Wang Y, Wetherall D, Zats D (2015) TIMELY: RTT-based Congestion Control for the Datacenter. Proc ACM SIGCOMM:537–550
43. Raiciu C, Barre S, Pluntke C, Greenhalgh A (2011) Improving datacenter performance and robustness with multipath TCP. Proc ACM SIGCOMM:266–277
44. Zou S, Huang J, Wang J, He T (2021) Flow-Aware Adaptive Pacing to Mitigate TCP Incast in Data Center Networks. IEEE/ACM Trans Networking 29(1):134–147
45. Wang T, Hamdi M (2018) eMPTCP: Towards High Performance Multipath Data Transmission By Leveraging SDN. Proc IEEE GLOBECOM:1–5
46. Ghorbani S, Yang Z, Godfrey P, Ganjali Y, Firoozshahian A (2017) DRILL: micro load balancing for low-latency data center networks. Proc ACM SIGCOMM:225–238
47. Zhou Z, Abawajy J, Chowdhury M, Hu Z, Li K, Cheng H, Alelaiwi A. A, Li F (2018) Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. Futur Gener Comput Syst 86:836–850
48. Geng Y, Jeyakumar V, Kabbani A, Alizadeh M (2016) JUGGLER: a practical reordering resilient network stack for datacenters. Proc ACM EuroSys:1–16
49. Liu J, Huang J, Lv W, Wang J (2020) APS: Adaptive Packet Spraying to Isolate Mix-flows in Data Center Network. IEEE Trans Cloud Comput:1–14. https://doi.org/10.1109/TCC.2020.2985037
50. Huang J, Lv W, Li W, Wang J, He T (2021) Mitigating Packet Reordering for Random Packet Spraying in Data Center Networks. IEEE/ACM Trans Netw. https://doi.org/10.1109/TNET.2021.3056601
51. Pizzutti M, Schaeffer-Filho A. E (2019) Adaptive multipath routing based on hybrid data and control plane operation. Proc IEEE INFOCOM:730–738
52. Fan F, Hu B, Yeung K. L (2019) Routing in black box: modularized load balancing for multipath data center networks. Proc IEEE INFOCOM:1639–1647
53. Wang P, Trimponias G, Xu H, Geng Y (2019) Luopan: Sampling-based load balancing in data center networks. IEEE Trans Parallel Distrib Syst 30(1):2230–2241

Publisher's Note