

# Fine-grained Load Balancing with Traffic-aware Rerouting in DataCenter Networks

Tao Zhang (✉ [tzhang@ccsu.edu.cn](mailto:tzhang@ccsu.edu.cn))

Changsha University

Yasi Lei

Changsha University

Qianqiang Zhang

Changsha University

Shaojun Zou

Changsha University

Juan Huang

Changsha University

Fangmin Li

Changsha University

---

## Research Article

**Keywords:** Rerouting, Load balancing, Data center networks

**Posted Date:** April 13th, 2021

**DOI:** <https://doi.org/10.21203/rs.3.rs-375253/v1>

**License:**   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

---

**Version of Record:** A version of this preprint was published at Journal of Cloud Computing on July 6th, 2021. See the published version at <https://doi.org/10.1186/s13677-021-00252-8>.

## RESEARCH

# Fine-grained Load Balancing with Traffic-aware Rerouting in Datacenter Networks

Tao Zhang<sup>1,2</sup>, Yasi Lei<sup>1</sup>, Qianqiang Zhang<sup>1</sup>, Shaojun Zou<sup>1,2\*</sup>, Juan Huang<sup>1</sup> and Fangmin Li<sup>1,2</sup>

\*Correspondence:

shaojunzou@ccsu.edu.cn

<sup>1</sup>School of Computer Engineering and Applied Mathematics, Changsha University, 410022 Changsha, China

Full list of author information is available at the end of the article

## 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, 2, 3, 4, 5, 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, 8, 9, 10, 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.

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, 18, 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.

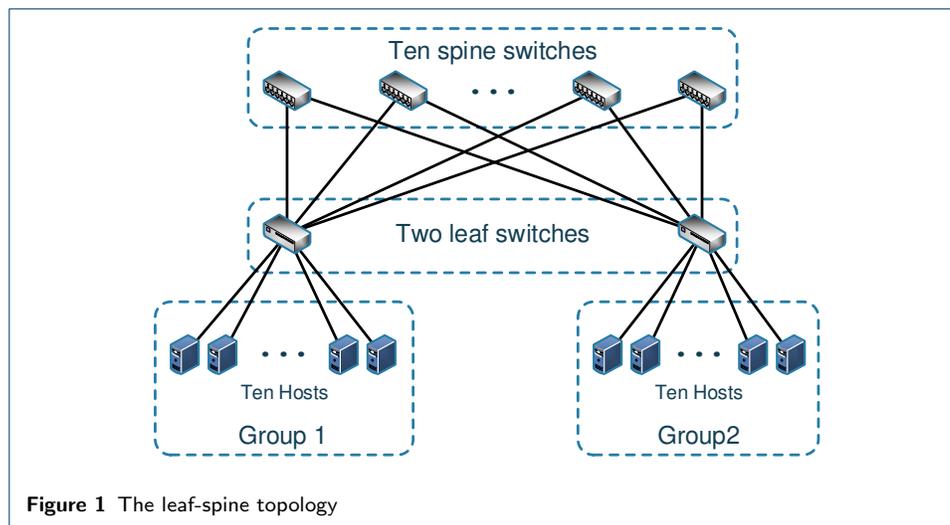
Our contributions are summarized as follows:

- 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 Section 2. The algorithm and details of TR are elaborated in Section 3. We make performance evaluation in Section 4, and discuss the related works in Section 5. Finally, we offer concluding remarks in Section 6.

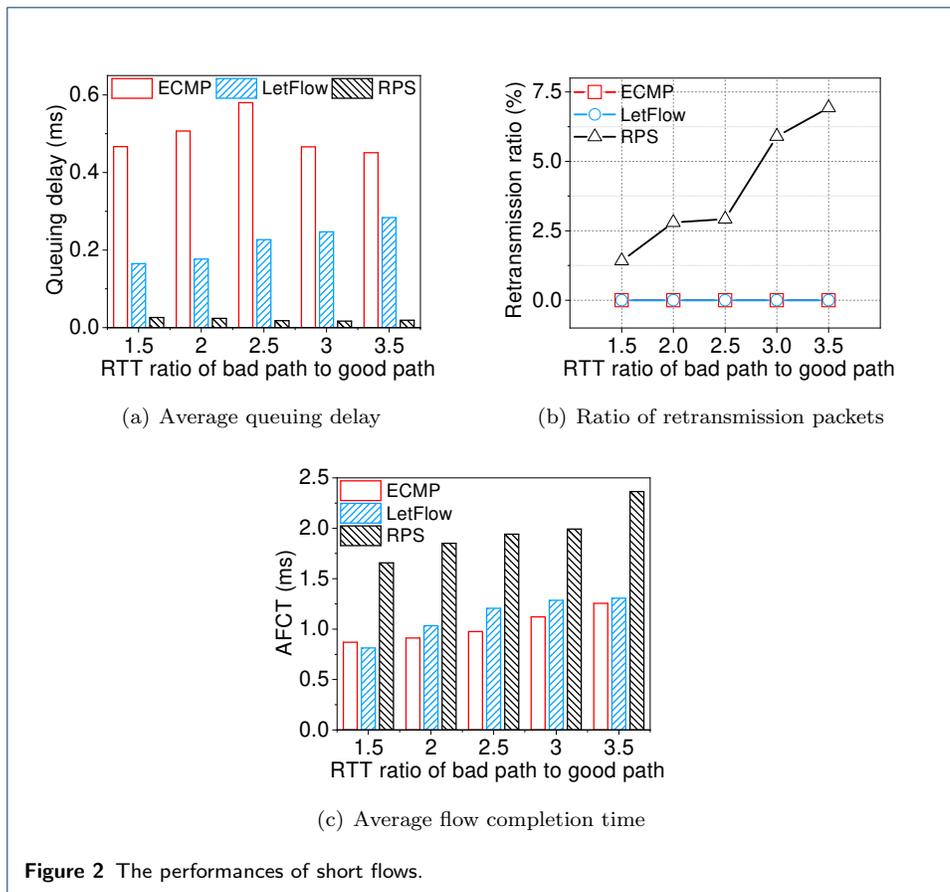
## Design Motivation

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 Figure. 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  $25\mu\text{s}$  propagation delay, and thus the round trip propagation delay (RTT) of good paths between two groups is  $200\mu\text{s}$ . 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 3) to the hosts in Group 2 by following a Poisson process. The threshold of flowlet using in LetFlow is set as  $500\mu\text{s}$  [8]. We evaluate the performance of three representative schemes in terms of the average flow completion times (AFCTs) of short flows ( $<100\text{KB}$ ), the total throughput of long flows ( $\geq 100\text{KB}$ ) [17, 18, 20], average queuing delay, the ratio of retransmission packets caused by packet reordering, etc.

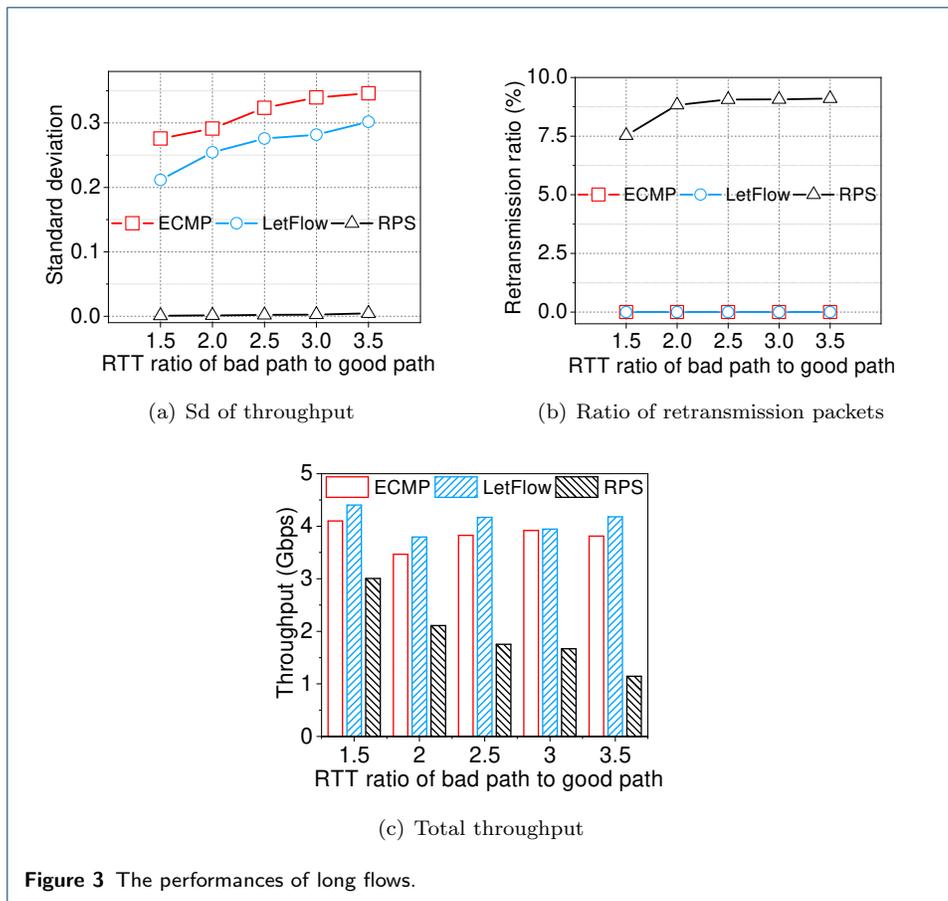
In Figure. 2(a), due to using fine-grained packet scattering, traffic under RPS is evenly distributed to all the available paths, hence the short flows experience the lowest queuing delay. Nonetheless, flows in the asymmetric scenario also have a high risk of experiencing packet reordering, which confuses the control logic of TCP stack deployed on end-hosts, generating many extra retransmission packets. As shown in Figure. 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 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 Figure. 3(b). Consequently, the results shown in Figure. 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 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, those output queues potentially lead to packet reordering are excluded from all the available ones, thus ensuring orderly transmission.

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. TR identifies flow types by counting how much data has been sent. When the amount of data has been sent in a flow exceeds 100KB, it is identified as a long flow. Otherwise, it is a short flow. 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 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 paths to maintain high link utilization and not block the short flows as much as possible.

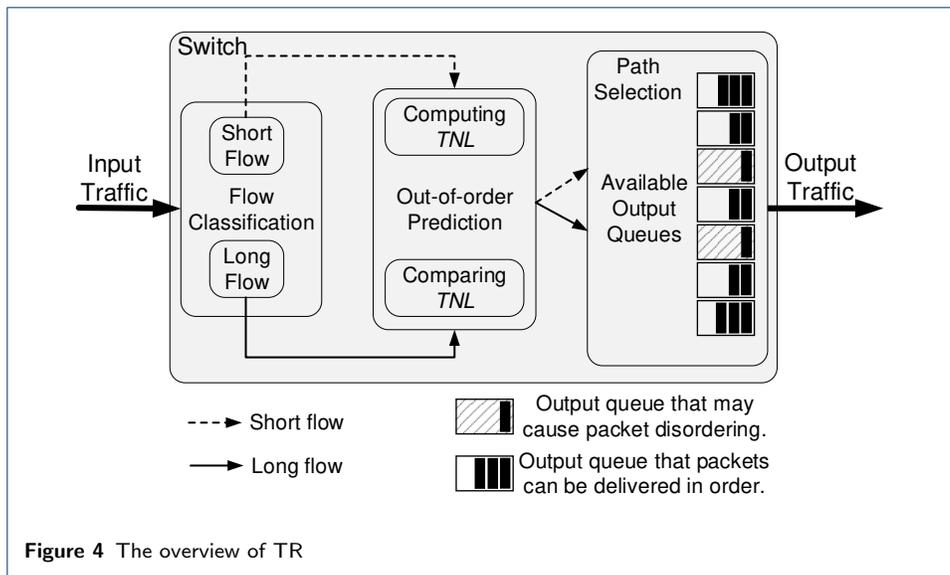
### **The algorithm of TR**

In this section, we first present the overview of TR, and then describe its pseudocode, in which the used variables are explained in Table 1.

#### ***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 filters out those output queues potentially cause packet reordering from all the available ones. 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 Figure. 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 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  $P$ .

**Table 1** 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

**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  $b_i$ ;

```

---

**Details**

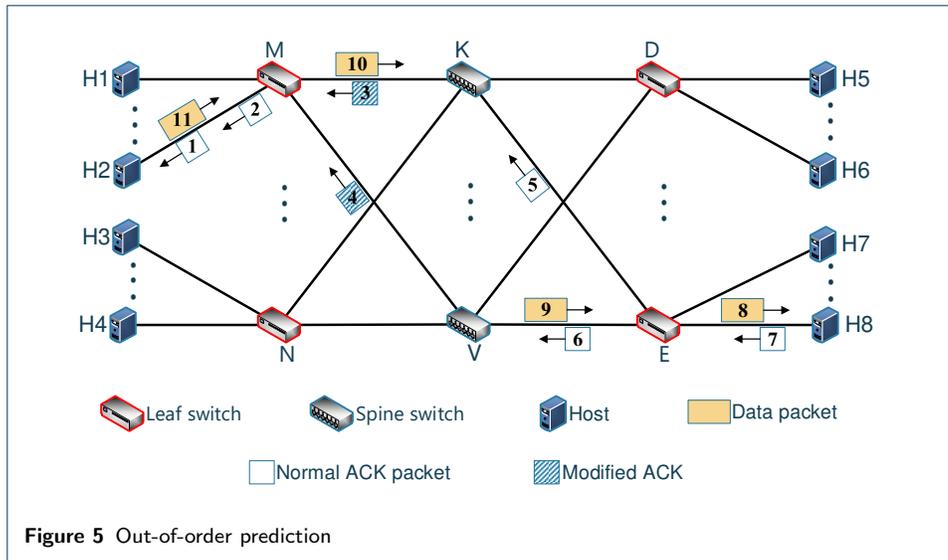
Next, we elaborate the implementation details of TR and answer the following questions: why and how to calculate TNL for each arriving packet, and how to obtain the number of short flows existing in each output queue.

***TNL calculation***

The two-level structure of leaf-spine topology has been widely used in modern data centers [7, 8, 9, 10, 11]. In this kind of network, the number of leaf switches on the path between any pairs of hosts is at most 2 [7, 8, 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 Figure. 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 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, 8, 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 Figure. 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 Figure. 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, 21], which just needs 3 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  $j$ th output queue in a leaf switch ( $TNL_j$ ) can be calculated by

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

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  $j$ th 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 2.

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,  $F_{RPS}^f$ , by

$$F_{RPS}^f = \frac{S}{\frac{3 \times w^2}{8}} \times \frac{w \times r}{2} = \frac{4 \times S \times r}{3 \times w}. \quad (2)$$

**Table 2** 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  $i$  ( $1 \leq i \leq S - H$ ) 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  $k$  ( $H \leq k \leq S - i$ ) 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 Equation (4), Equation (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  $(\frac{n_g}{n})^w$ . Accordingly, the probability that at least one packet in  $w$  selects the bad path is  $1 - (\frac{n_g}{n})^w$ . Thus, the average end-to-end propagation delay can be computed by  $(1 - (\frac{n_g}{n})^w) \times d_b + (\frac{n_g}{n})^w \times d_g$ , 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

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

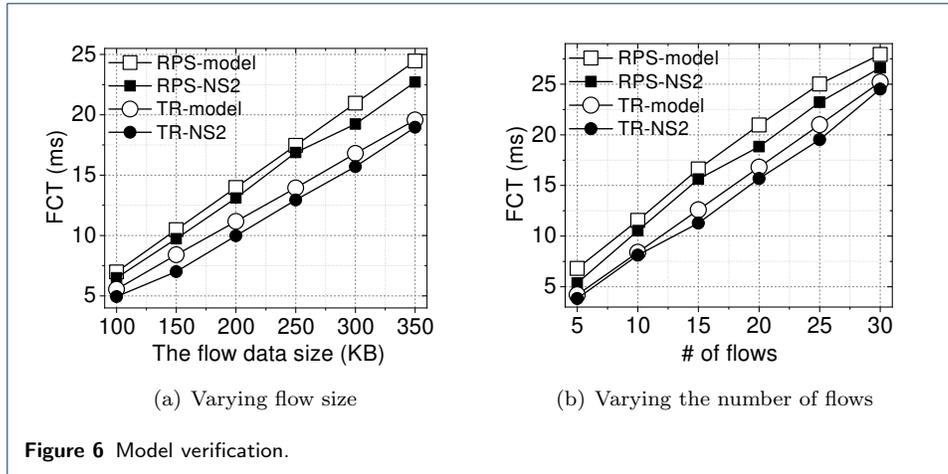
Finally, with Equations (2), (5), and (6), we obtain the flow completion time of  $f$  under RPS as

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

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 Equation (2), we get

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

Finally, we conduct both model and NS2 tests based on the scenario in Section 2 to verify our analysis. In Figure. 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 Figure. 6(b), the data size of flow  $f$  is fixed to 300KB, and we introduce



**Figure 6** Model verification.

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.

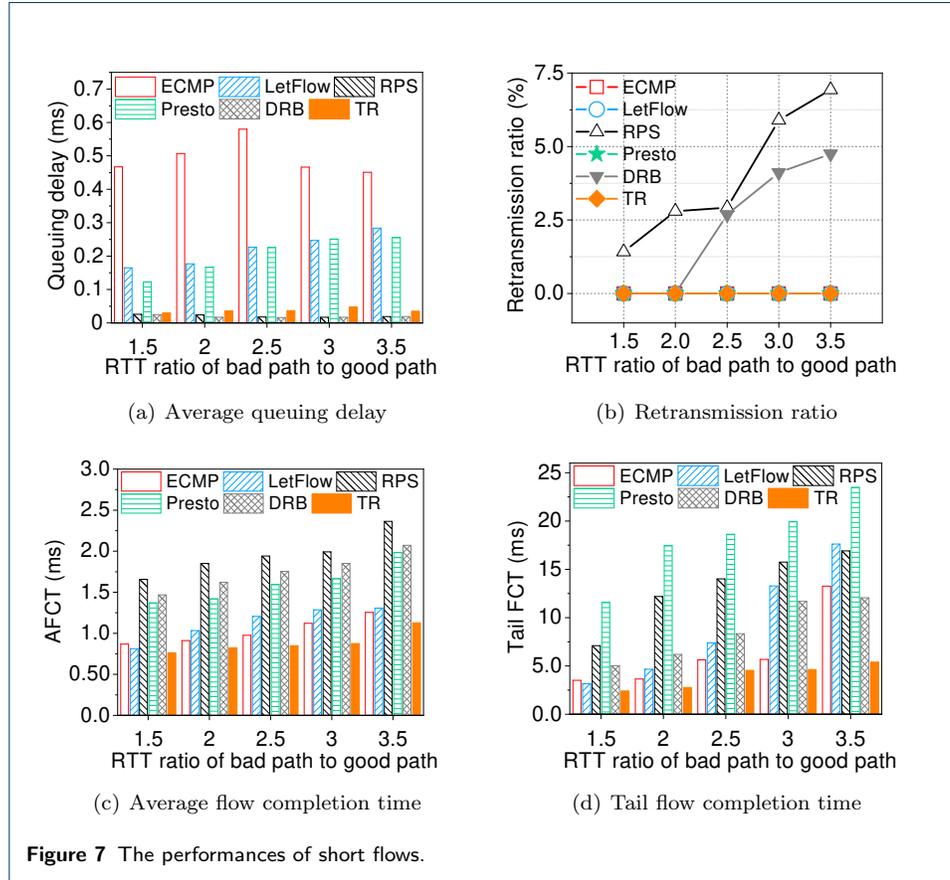
## Performance Evaluation

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  $500\mu\text{s}$  in LetFlow[8], and the data size of flowcell is 64KB for Presto[14].

### Micro-Benchmark

Firstly, we redo the micro-benchmark in Section 2 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 Figure. 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 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 Figures. 7(c) and 7(d).



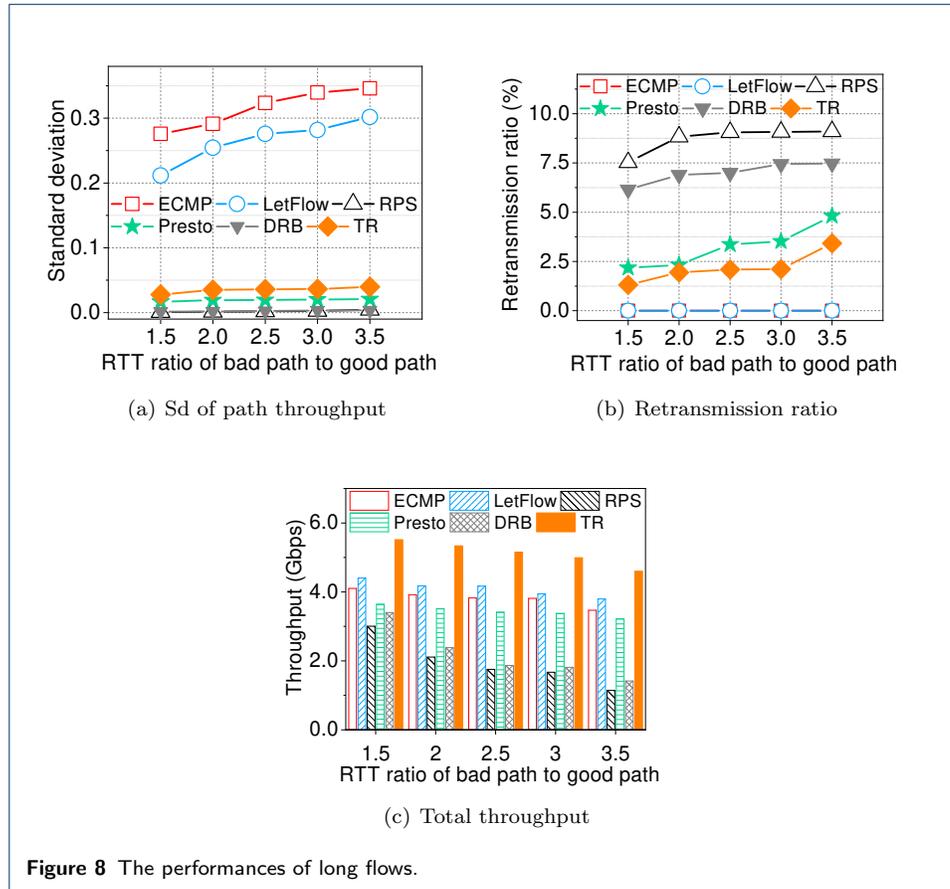
**Figure 7** The performances of short flows.

On the other hand, the long flows' performance is closely related to packet retransmission. As shown in Figure. 8, RPS and DRB can distribute load in the most balanced way, but possess high retransmission ratio, which in turn leads to the lowering 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.

### Large-scale evaluation

In this part, 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 is randomly

selected to send DCTCP flows to the remained hosts. All flows start by following a Poisson process. We also vary the load intensity by changing the total number of existing flows from 10000 to 50000 in each test. ECMP [12] is used as the baseline to illustrate all the test results.



**Figure 8** The performances of long flows.

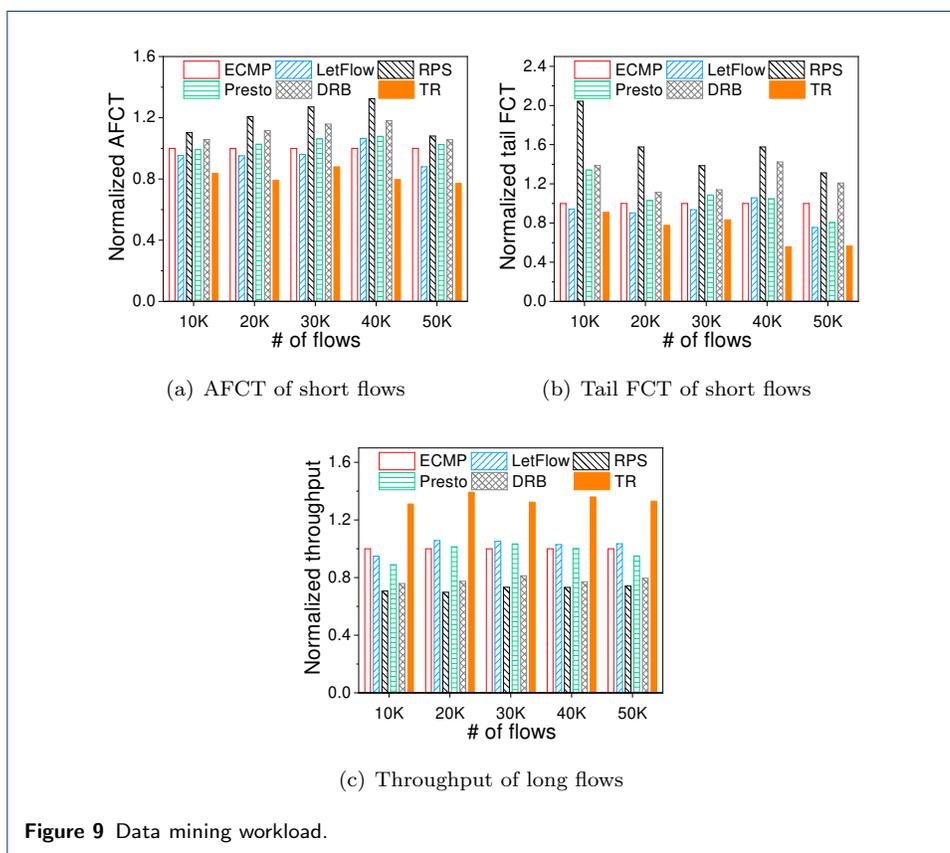
Besides, to make a comprehensive comparison, we install four representative data center workloads such as Data Mining, Web Search, Cache Follower, and Web Server [19]. Specifically, Web Server has the most short flows (86%), but its number of long flows is the smallest (14%). Besides, the data sizes of its long flows are all smaller than 1MB. Cache Follower is just the opposite for which has the most long flows (47%), and thus its number of short flows is the smallest (53%). Data Mining possesses the most short flows ( $\leq 10\text{KB}$ ), and the numbers of this kind of short flows in the other three workloads are all larger than 50%. Except Web Server, the ratios of short flows in the other three workloads are very low ( $\leq 5\%$ ). Overall, across four workloads, the ratios of short flows are always higher than those of long flows, following the heavy-tail distribution of data center traffic [17, 18, 20, 16]. The detailed distributions are listed in Table 3.

From Figure. 9 to Figure. 12, we can find that the performances of fine-grained schemes, such as RPS and DRB, are always worse than the other schemes whether for short flows or long flows. This is mainly caused by plenty of retransmission packets since in the asymmetric network, both short or long flows are very prone to experience packet reordering. By contrast, DRB performs a little better than RPS, which

**Table 3** The distributions of flow data size under 4 typical datacenter 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%	-

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 different performance requirements. As a consequence, its performance, whether for short and long flows, is always better than the other schemes under different load intensities.

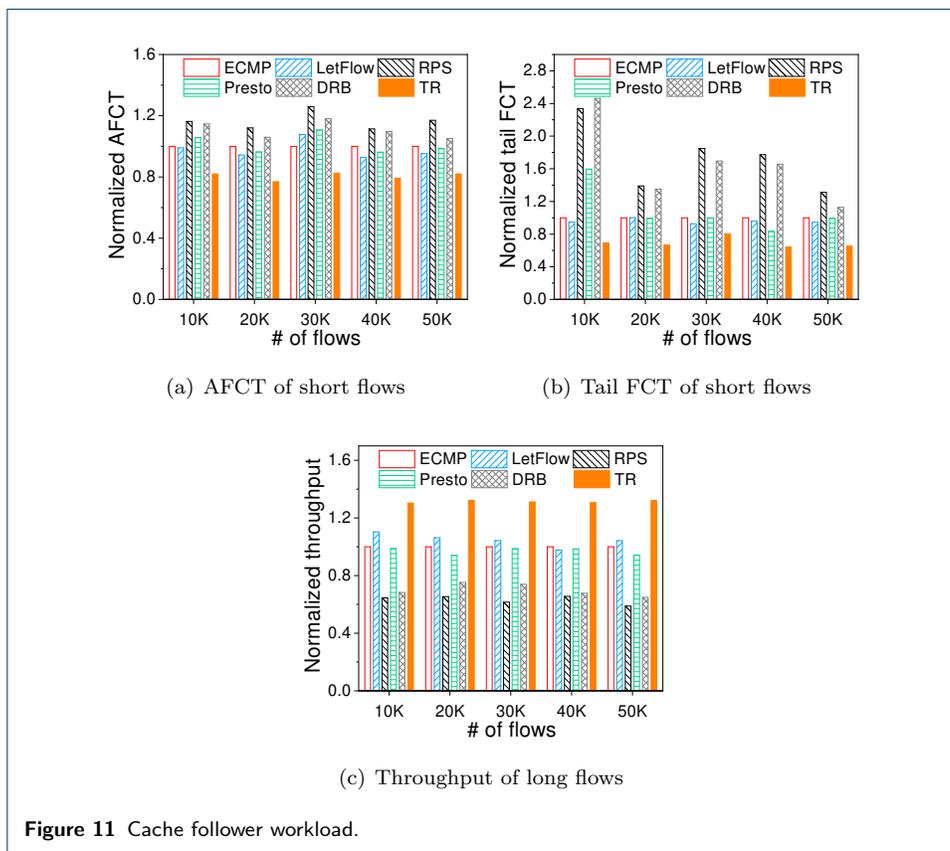
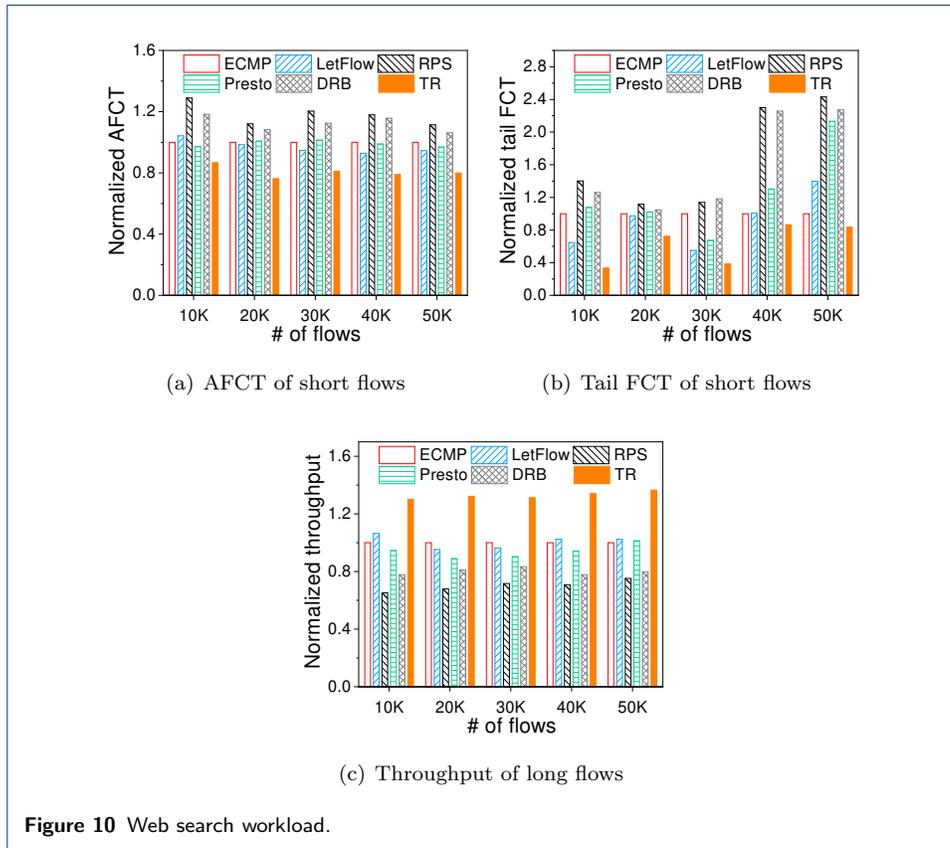


**Figure 9** Data mining workload.

### Related Works

In recent years, researchers have designed plenty of transport protocols to improve the transmission performance of datacenter networks [1, 2, 3, 4, 5, 6, 18, 22, 23, 24, 25, 26, 27, 28, 29]. 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



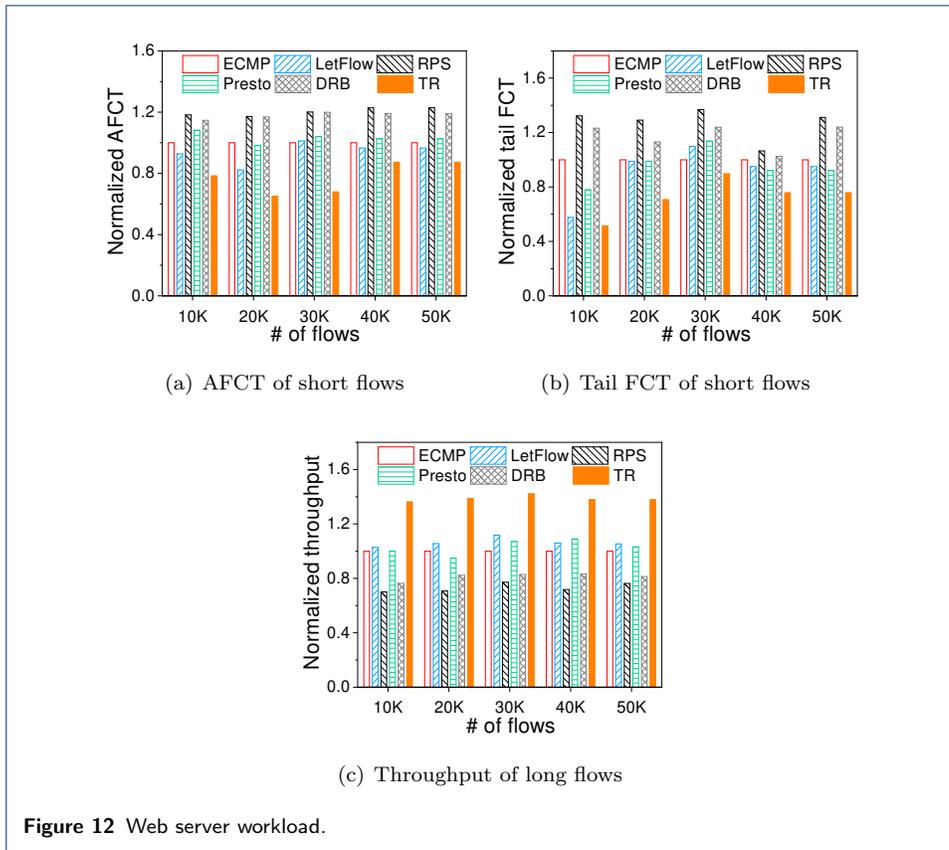


Figure 12 Web server workload.

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 [30] 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 link/switch failures [7, 31], which inevitably makes network topology become asymmetric and results in serious packet reordering.

MMPTCP [22] initially scatters packets to exploit all available paths and increases fast retransmission threshold to handle packet reordering. Then MMPTCP switches to the MPTCP [27] when the amount of data transmitted by sender is larger than a given threshold. JUGGLER [32] 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 [33] adopt isolate mechanism to achieve adaptive packet spraying. QDAPS [34] is a delay-aware mechanism. When a packet arrives at the switch, it is assigned to the output port, which has larger queuing 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 [35] 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 [36] 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 [37] 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.

### Availability of data and material

Not applicable

### Competing interests

The authors declare that they have no competing interests.

### 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.

### Author's contributions

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

### Acknowledgements

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

### Author details

<sup>1</sup>School of Computer Engineering and Applied Mathematics, Changsha University, 410022 Changsha, China. <sup>2</sup>Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, 410022 Changsha, China.

### References

1. G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, A. Vahdat, "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter," in *Proc. ACM SIGCOMM*, 2020, pp. 514-528.
2. A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani, and A. Vahdat, "Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates," in *Proc. ACM SIGCOMM*, 2020, pp. 735-749.
3. S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan, Y. Wang, "Aeolus: A Building Block for Proactive Transport in Datacenters," in *Proc. ACM SIGCOMM*, 2020, pp. 1-13.
4. T. Zhang, J. Huang, K. Chen, J. Wang, J. Chen, Y. Pan, G. Min, "Rethinking Fast and Friendly Transport in Data Center Networks," *IEEE/ACM Transactions on Networking*, 2020, 28(5): 2364-2377.
5. G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, and L. Cui, "Congestion control for cross-datacenter networks," in *Proc. IEEE ICNP*, 2019, pp. 1-12.
6. M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010, pp. 63-74.
7. H. Zhang, J. Zhang, W. Bai, C. Kai, M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. ACM SIGCOMM*, 2017, pp. 253-266.
8. E. Vanini, R. Pan, M. Alizadeh, P. Taheri, T. Edsall, "Let it flow: resilient asymmetric load balancing with flowlet switching," in *Proc. USENIX NSDI*, 2017, pp. 407-420.
9. M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503-514.
10. G. Michelogiannakis, K. Z. Ibrahim, J. Shalf, J. J. Wilke, S. Knight and J. P. Kenny, "APHiD: Hierarchical Task Placement to Enable a Tapered Fat Tree Topology for Lower Power and Cost in HPC Networks," *IEEE/ACM CCGRID*, 2017, pp. 228-237.
11. A. Dixit, P. Prakash, Y. C. Hu, R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, 2013, pp. 2130-2138.
12. C.E. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. In RFC 2992.
13. J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, D. Maltz, "Per-packet Load-balanced," Low-latency Routing for Clos-based Data Center Networks," in *Proc. ACM CoNEXT*, 2013, pp. 49-60.
14. K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akella, "Presto: edge-based load balancing for fast datacenter networks," in *Proc. ACM SIGCOMM*, 2015, pp. 465-478.
15. L. Zhou, C. Chou, L. N. Bhuyan, K. K. Ramakrishnan and D. Wong, "Joint Server and Network Energy Saving in Data Centers for Latency-Sensitive Applications," *IEEE IPDPS*, 2018, pp. 700-709.
16. J. Ye, L. Ma, C. Pang, Q. Xiao, W. Jiang, "Inferring Coflow Size based on Broad Learning System in Data Center Network," in *Proc. of IEEE ICCT*, 2020.
17. T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, 2010, pp. 267-280.
18. H. Xu, B. Li, "RepFlow: Minimizing Flow Completion Times with Replicated Flows in Data Centers," in *Proc. IEEE INFOCOM*, 2014, pp. 1581-1589.
19. I. Cho and K. Jang, D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*, 2017, pp. 239-252.
20. W. Bai, L. Chen, K. Chen, D. Han, C. Tian, H. Wang, "PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers," *IEEE/ACM Transactions on Networking*, 2017, 25(4): 1954-1967.
21. T. Wang, L. Wang, M. Hamdi, "A Cost-effective Low-latency Overlaid Torus-based Data Center Network Architecture," *Computer Communications*, 2018, 129(SEP.): 89-100.
22. M. Kheirkhah, I. Wakeman, G. Parisi, "MMPTCP: A multipath transport protocol for data centers," in *Proc. IEEE INFOCOM*, 2016, pp. 1-9.
23. K. He, E. Rozner, K. Agarwal, Y. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks," in *Proc. ACM SIGCOMM*, 2016, pp. 244-257.
24. B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy, "Virtualized Congestion Control," in *Proc. ACM SIGCOMM*, 2016, pp. 230-243.
25. C Lee, C Park, K Jang, S Moon, and D Han, "DX: Latency-Based Congestion Control for Datacenters," *IEEE/ACM Transactions on Networking*, 2017, 25(1): 335-348.
26. R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *Proc. ACM SIGCOMM*, 2015, pp. 537-550.
27. C. Raiciu, S. Barre, C. Pluntke, and A. Greenhalgh, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266-277.

28. S. Zou, J. Huang, J. Wang, T. He, "Flow-Aware Adaptive Pacing to Mitigate TCP Incast in Data Center Networks," *IEEE/ACM Transactions on Networking*, 2021, 29(1): 134-147.
29. T. Wang, and M. Hamdi, "eMPTCP: Towards High Performance Multipath Data Transmission By Leveraging SDN," in *Proc. IEEE GLOBECOM*, 2018, pp. 1-5.
30. S. Ghorbani, Z. Yang, P. Godfrey, Y. Ganjali, A. Firoozshahian, "DRILL: micro load balancing for low-latency data center networks," in *Proc. ACM SIGCOMM*, 2017, pp. 225-238.
31. Z. Zhou, J. Abawajy, M. Chowdhury, Z. Hu, K. Li, H. Cheng, A. A. Alelaiwi, F. Li, "Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms," *Future Generation Computer Systems*, 2018, 86: 836-850.
32. Y. Geng, V. Jeyakumar, A. Kabbani, M. Alizadeh, "JUGGLER: a practical reordering resilient network stack for datacenters," in *Proc. ACM EuroSys*, 2016, pp. 1-16.
33. J. Liu, J. Huang, W. Lv, J. Wang, "APS: Adaptive Packet Spraying to Isolate Mix-flows in Data Center Network," *IEEE Transactions on Cloud Computing*, 2020, 1-14. DOI: 10.1109/TCC.2020.2985037.
34. J. Huang, W. Lv, W. Li, J. Wang, T. He, "Mitigating Packet Reordering for Random Packet Spraying in Data Center Networks," *IEEE/ACM Transactions on Networking*, 2021. <https://doi.org/10.1109/TNET.2021.3056601>.
35. M. Pizzutti, A. E. Schaeffer-Filho, "Adaptive multipath routing based on hybrid data and control plane operation," in *Proc. IEEE INFOCOM*, 2019, pp. 730-738.
36. F. Fan, B. Hu, K. L. Yeung, "Routing in black box: modularized load balancing for multipath data center networks," in *Proc. IEEE INFOCOM*, 2019, pp. 1639-1647.
37. P. Wang, G. Trimponias, H. Xu, Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Transactions on Parallel and Distributed Systems*, 2019, 30(1): 2230-2241.

# Figures

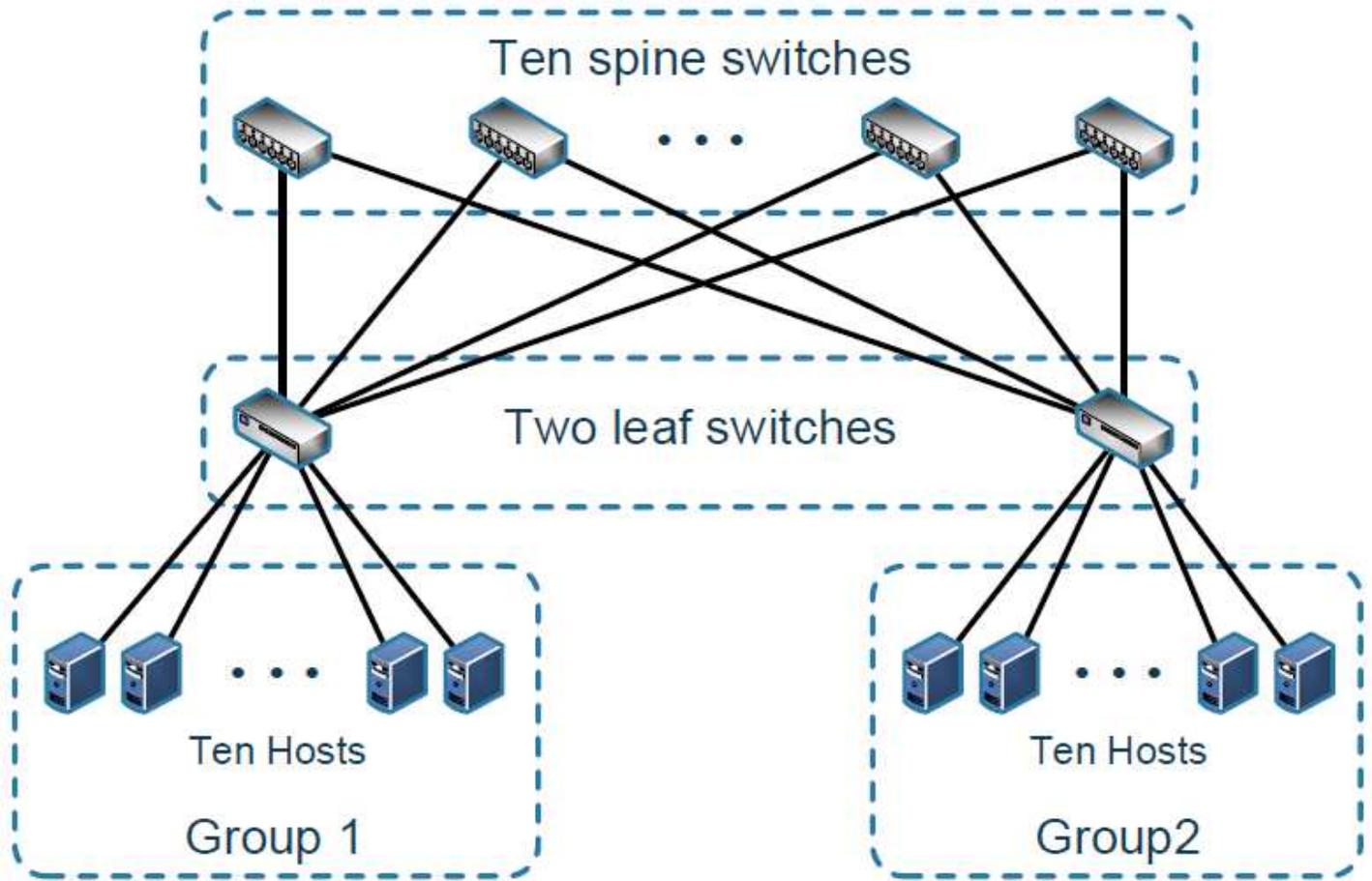
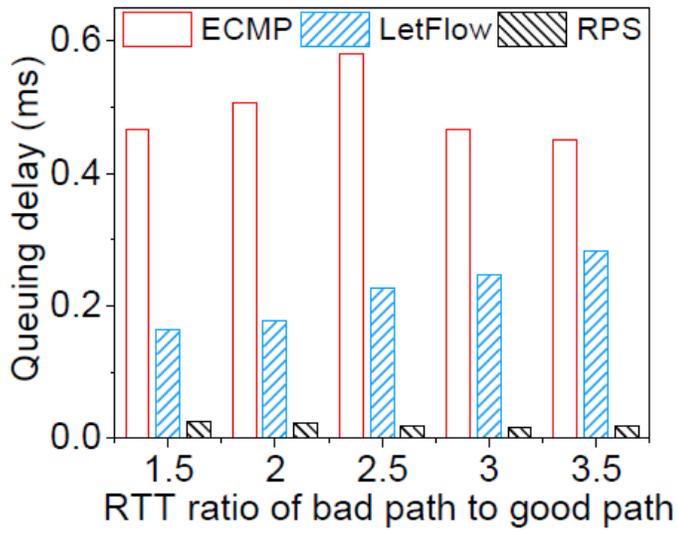
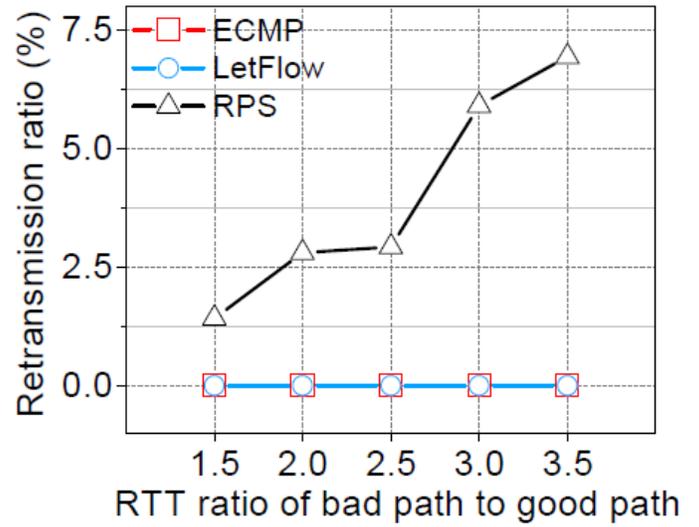


Figure 1

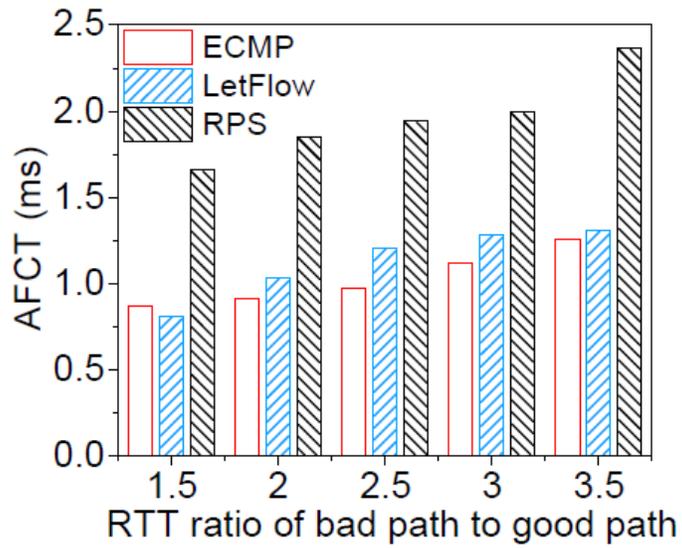
The leaf-spine topology



(a) Average queuing delay



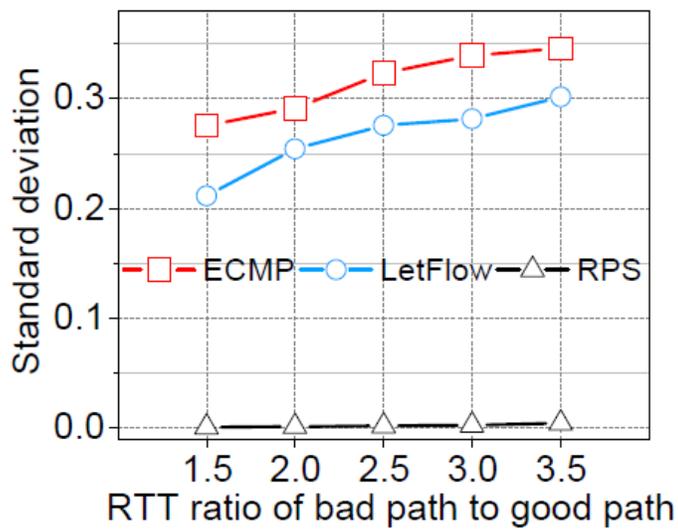
(b) Ratio of retransmission packets



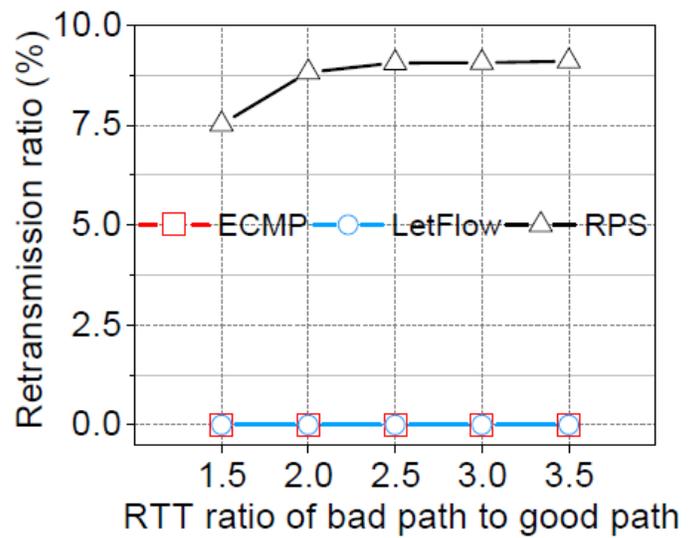
(c) Average flow completion time

**Figure 2**

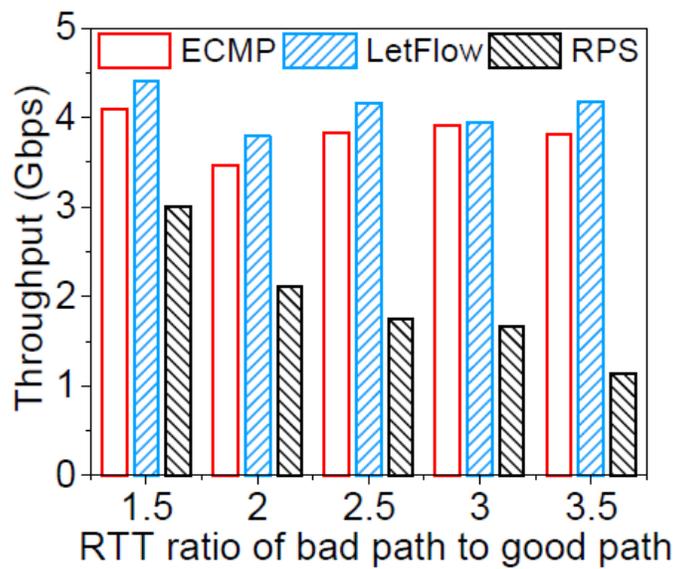
The performances of short flows.



(a) Sd of throughput



(b) Ratio of retransmission packets



(c) Total throughput

**Figure 3**

The performances of long flows.

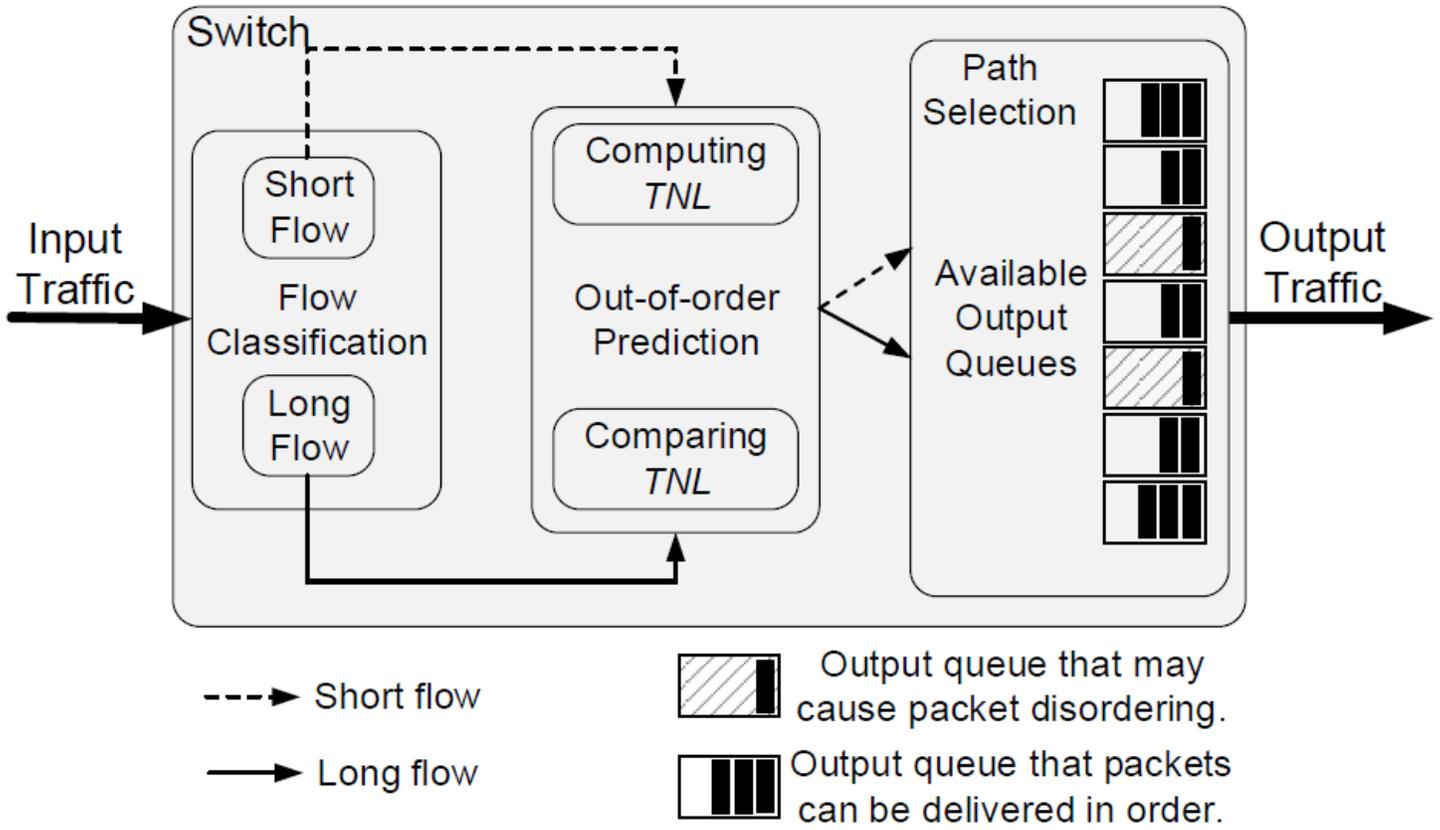


Figure 4

The overview of TR

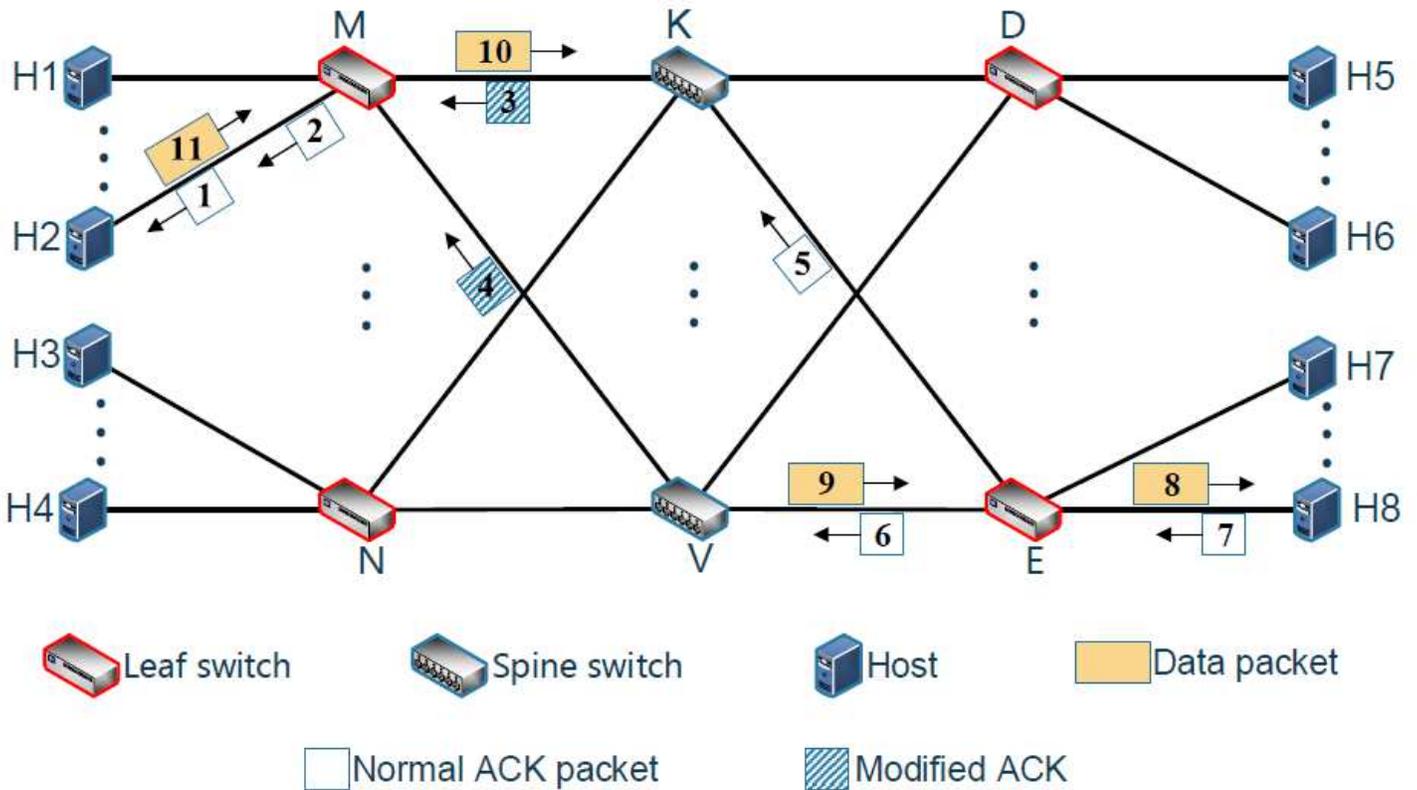
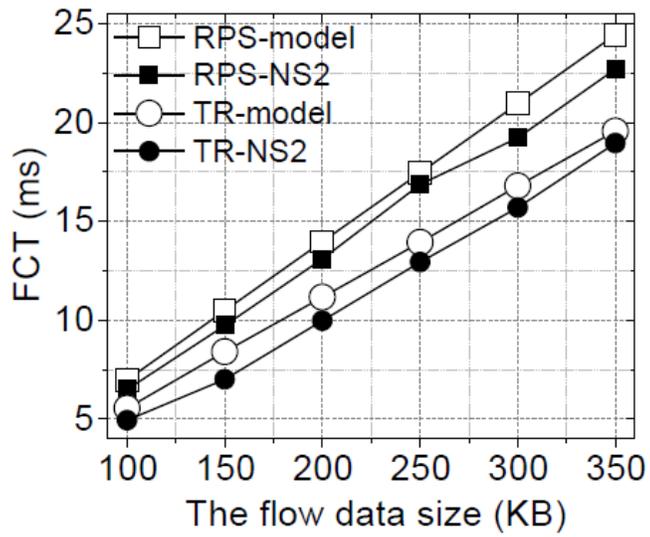
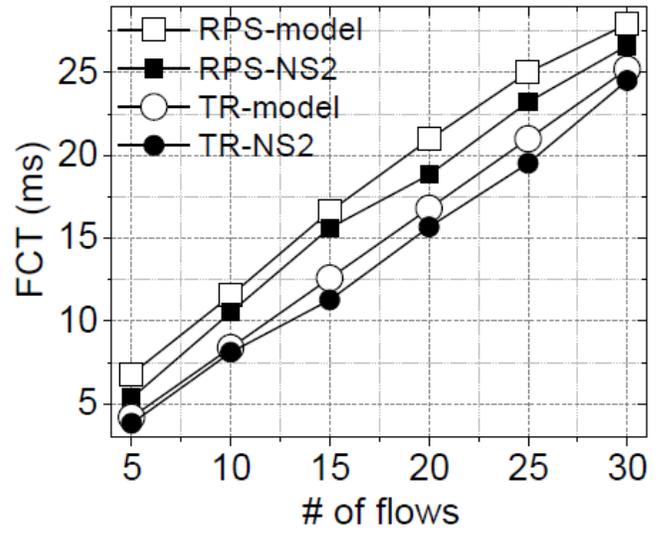


Figure 5

Out-of-order prediction



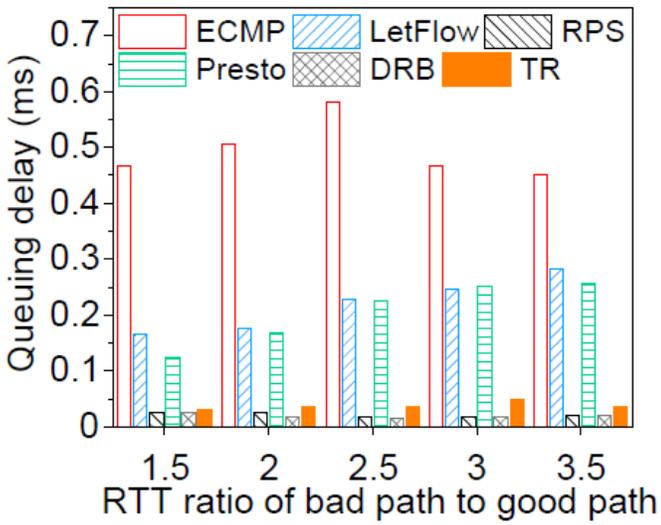
(a) Varying flow size



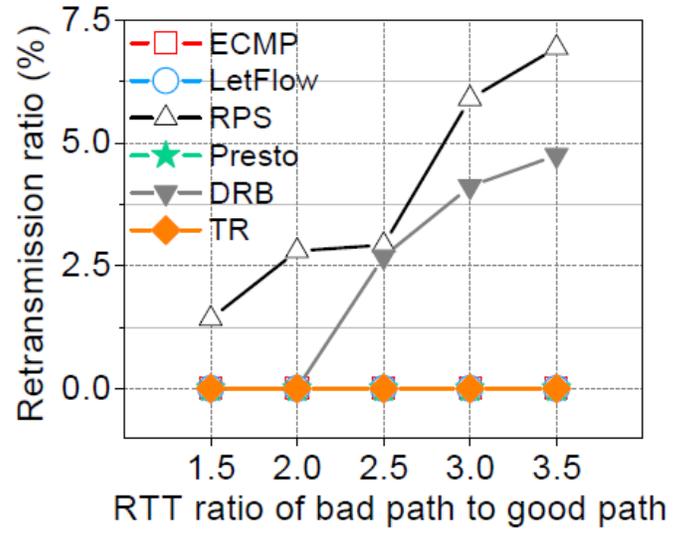
(b) Varying the number of flows

Figure 6

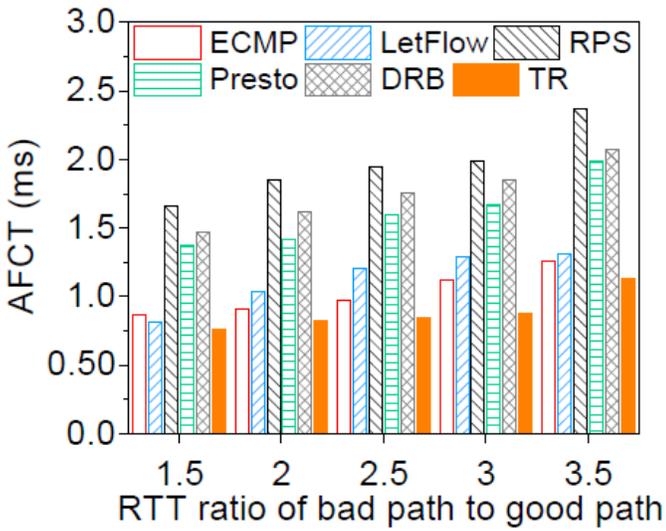
Model verification.



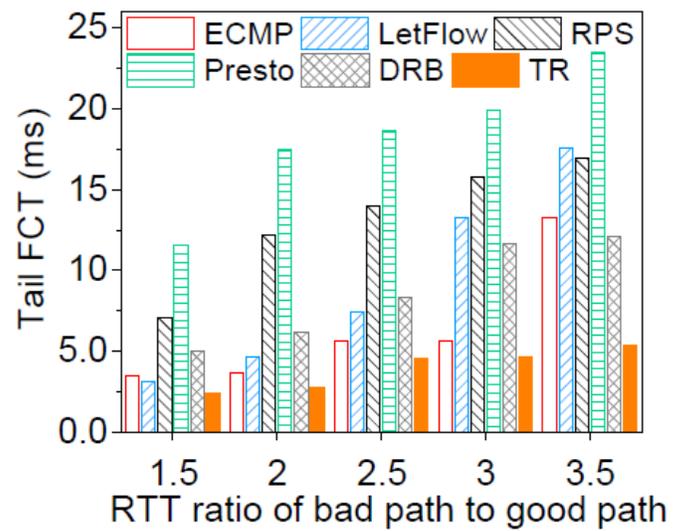
(a) Average queuing delay



(b) Retransmission ratio



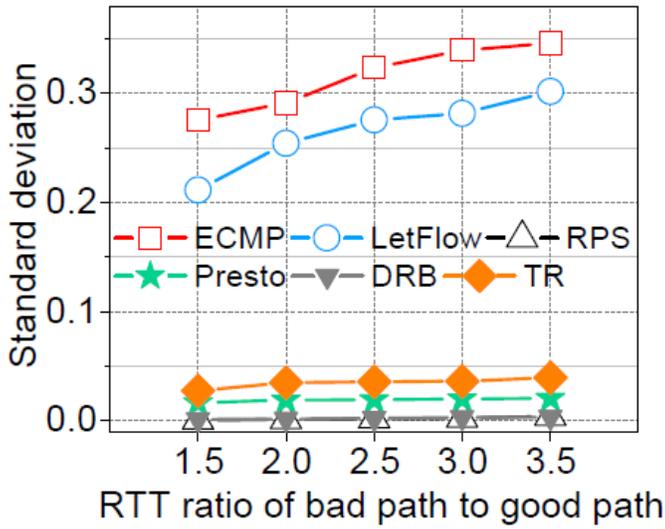
(c) Average flow completion time



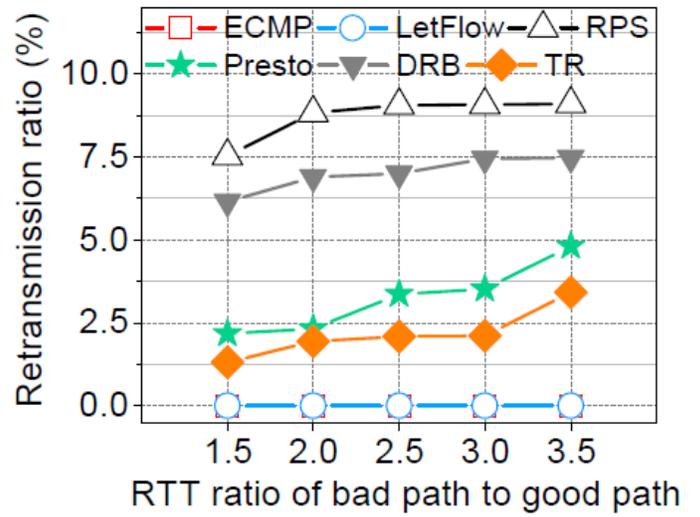
(d) Tail flow completion time

Figure 7

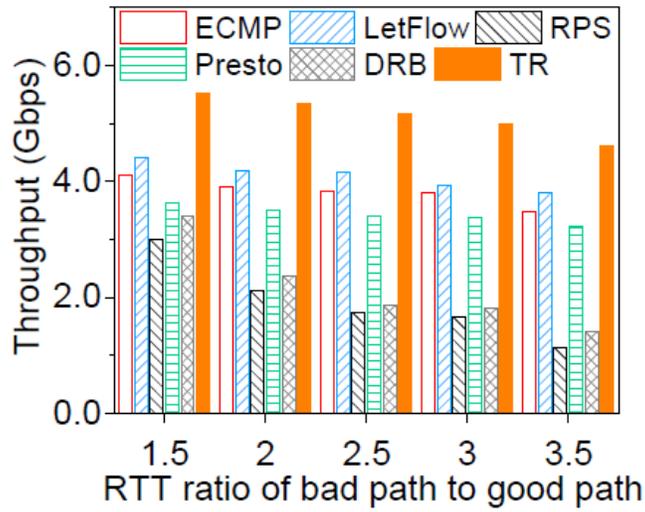
The performances of short flows.



(a) Sd of path throughput



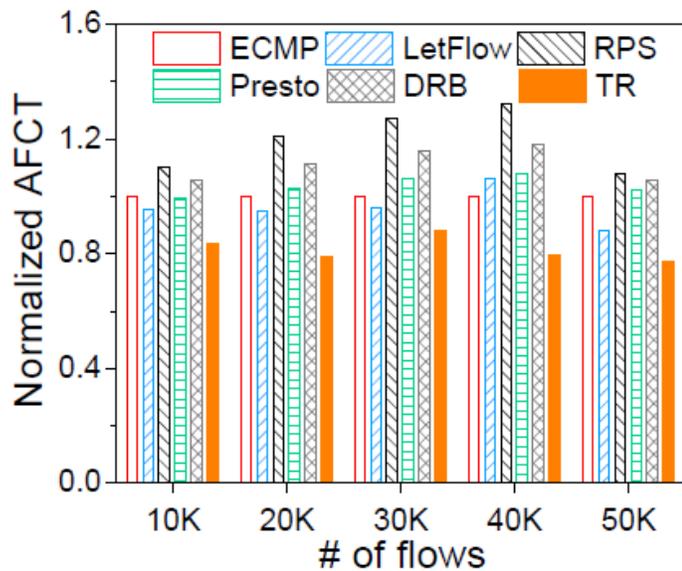
(b) Retransmission ratio



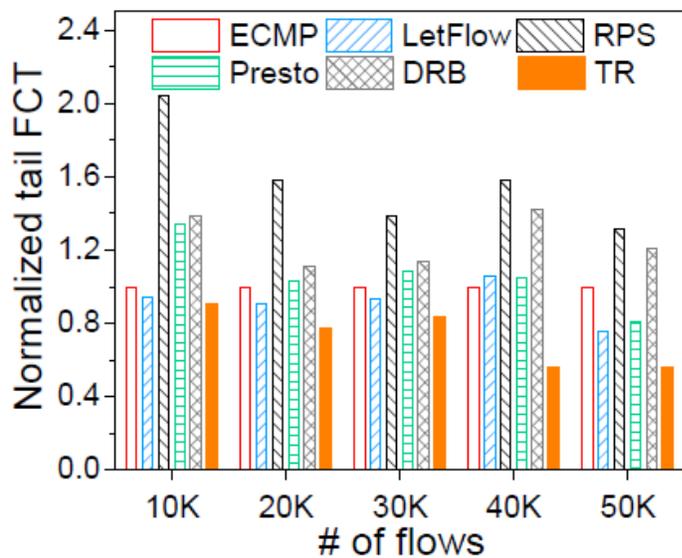
(c) Total throughput

Figure 8

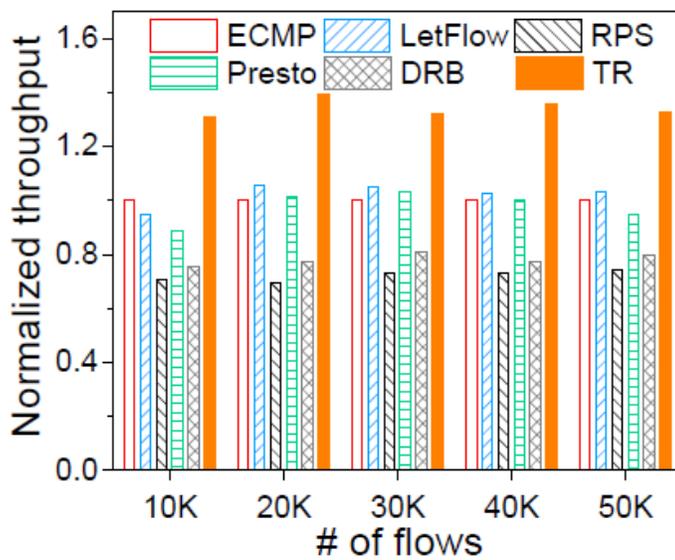
The performances of long flows.



(a) AFCT of short flows



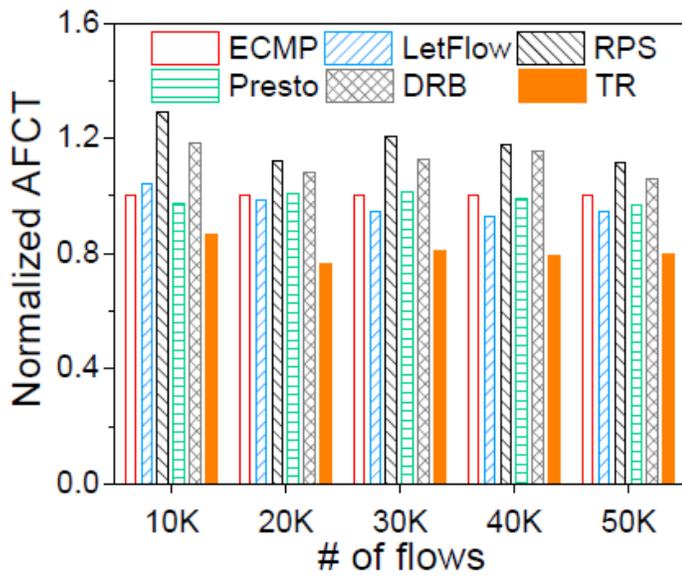
(b) Tail FCT of short flows



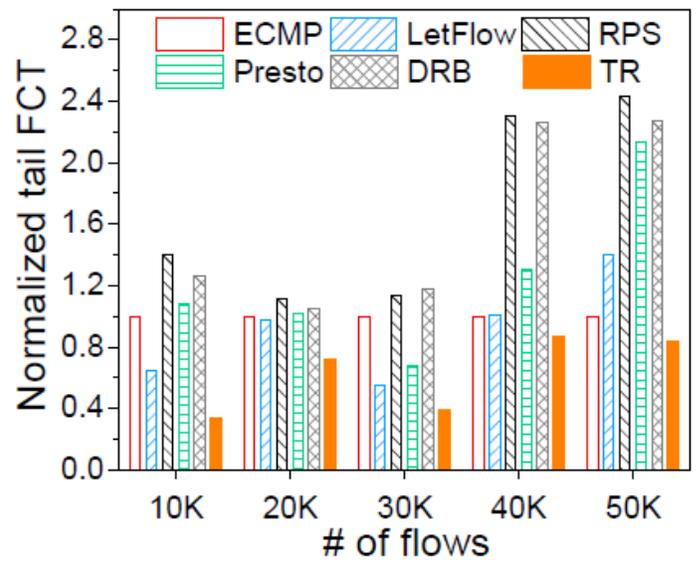
(c) Throughput of long flows

Figure 9

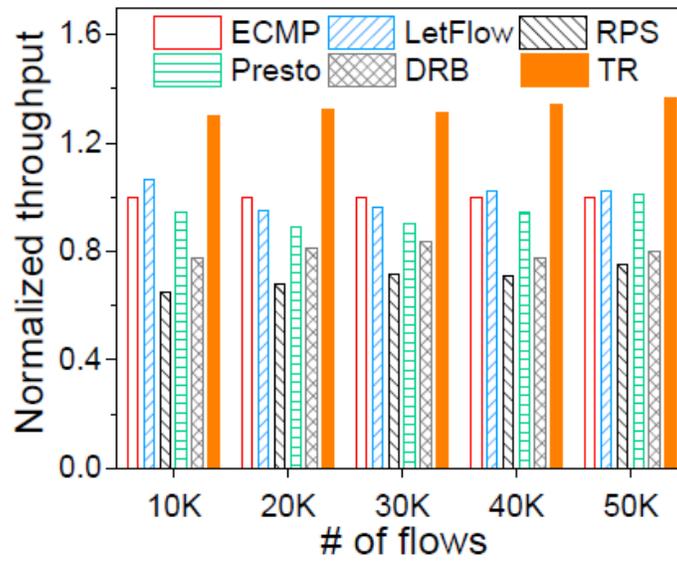
Data mining workload.



(a) AFCT of short flows



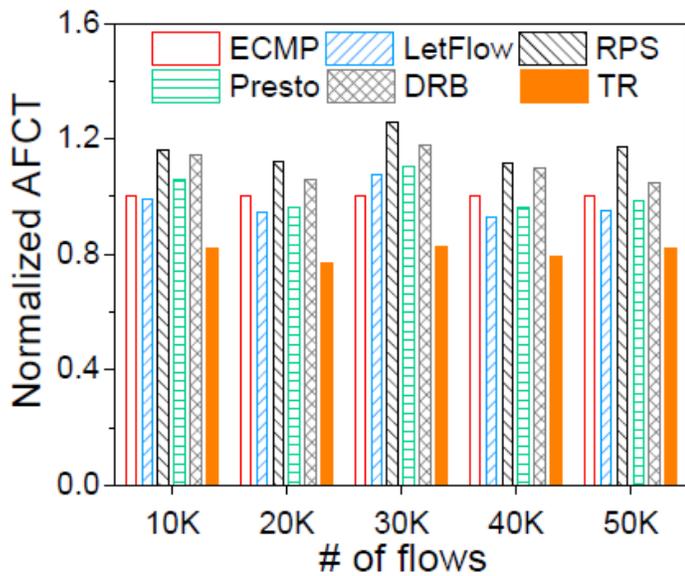
(b) Tail FCT of short flows



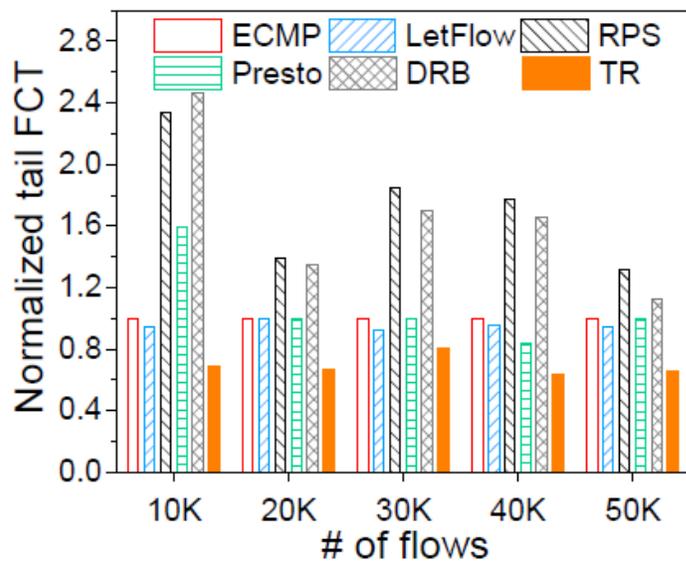
(c) Throughput of long flows

Figure 10

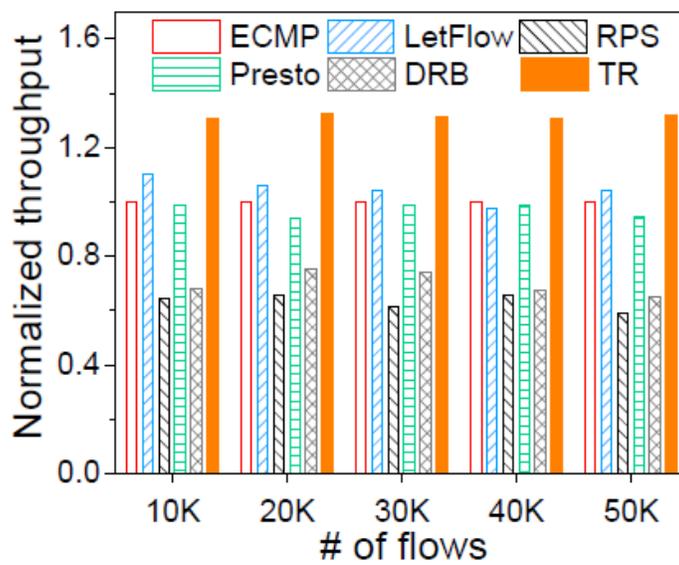
Web search workload.



(a) AFCT of short flows



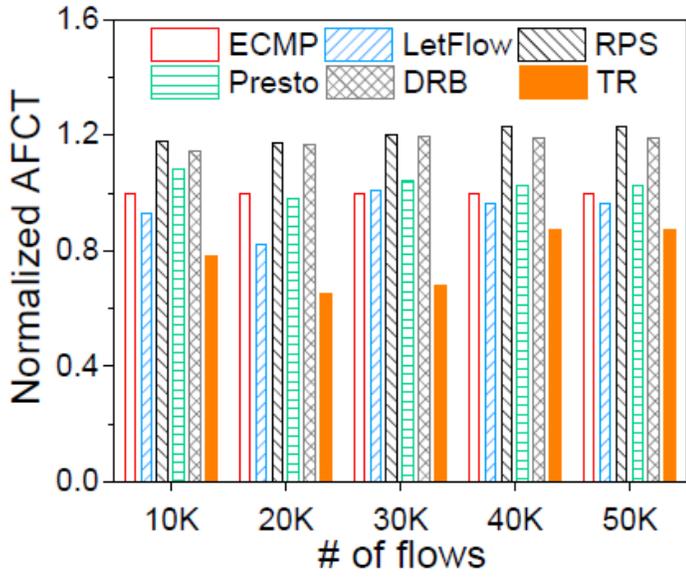
(b) Tail FCT of short flows



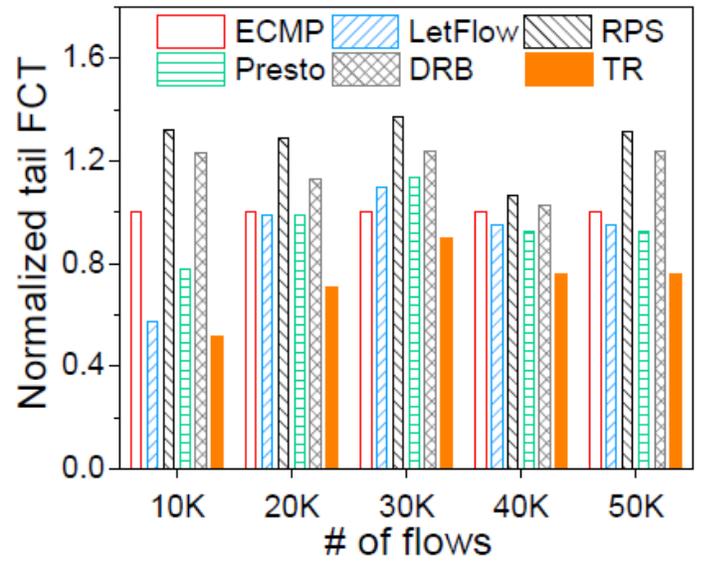
(c) Throughput of long flows

Figure 11

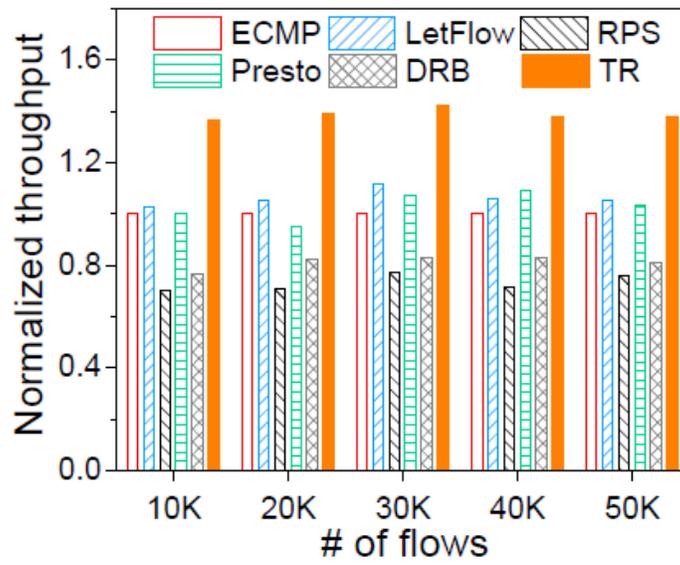
Cache follower workload.



(a) AFCT of short flows



(b) Tail FCT of short flows



(c) Throughput of long flows

Figure 12

Web server workload.