Efficient Data Center Flow Scheduling without Starvation using Expansion Ratio

Sheng Zhang, *Member, IEEE,* Zhuzhong Qian, *Member, IEEE,* Hao Wu, and Sanglu Lu, *Member, IEEE*

Abstract—Existing data center transport protocols are usually based on the Processor Sharing (PS) policy and/or the Shortest Remaining Processing Time (SRPT) policy. PS divides link bandwidth equally between competing flows, thus it fails to achieve optimal average flow completion time (FCT). SRPT prioritizes flows that have the shortest remaining processing time and provides near-optimal average FCT, but it may cause long flows to suffer unfair delays, or even starve them. In fact, these two types of policies represent two directions in the design space: PS prefers fairness (in terms of starvation freedom) while SRPT favors efficiency (in terms of average FCT). In this paper, we propose a novel metric, expansion ratio, which enables us to strike a balance between SRPT and PS. We design MERP that achieves efficient flow scheduling without starvation. MERP takes care of both average and tail FCTs by minimizing the expansion ratio of competing flows in a lexicographically manner. MERP controls the sending rate of competing flows via synchronized virtual deadlines and routes flows in a downstream-aware manner that reacts quickly to link failures. We evaluate MERP using extensive NS2-based simulations. Results show that, under various traffic loads, MERP reduces the tail FCT significantly with a negligible increase of average FCT compared with pFabric, and MERP reduces the average FCT notably compared with ECMP and CONGA when link failures occur.

Index Terms—Data center, flow completion time, efficiency, starvation freedom, expansion ratio.

1 Introduction

Today's data centers usually organize online interactive services with the partition-aggregate pattern. The completion time of a large task largely depends on the flow completion times (FCT) of the flows generated in this process. In current data centers, the FCTs of these flows fluctuate dramatically [1]—they may experience 2x more mean FCT than its theoretical minimum [2], [3]. This is unacceptable, since the response to user requests for those interactive services must be quick enough: even fractions of a second could have an important influence on user experiences, which in turn impairs the business revenue of cloud service providers [4], [5].

To achieve low latency, many transport protocols have been proposed in recent years, among which two main techniques are Shortest Remaining Processing Time (SRPT) and Processor Sharing (PS).

SRPT-based policies [2], [6] focus on minimizing the average FCT by prioritizing flows that have the shortest remaining processing time and have near-optimal average FCT. However, applying SRPT-based policies has several problems. First, long flows are completely ignored by SRPT-based protocols [7]. When short flows contribute the majority of the overall traffic, SRPT could even unfairly "starve" long flows, which probably increases the response time of partition-aggregate-based online services. This is because, in such kind of services, there are flows with various sizes, and the overall response time is often dominated by the "lagger". In this sense, some online service may have better performance with short tail FCT rather than short average FCT. The second concern with SRPT is that, an malicious service could preempt bandwidth resources by simply splitting large flows into many small ones. If all services split their large flows into small ones, none of them can achieve better performance than no split.

PS-based policies [8], [9], [10] reduce the queueing length at switch ports through novel congestion detection and rate control mechanisms. These approaches usually divide link bandwidth equally among competing flows, and thus short flows are often blocked by long flows even though they can finish earlier. Therefore, the average FCT of PS-based policies is larger than that of SRPT-based policies [2], which implicitly implies that, we cannot simultaneously achieve the minimum average and the minimum tail FCTs.

We observe that, these two types of policies represent two extremes in the design space: SRPT favors efficiency (in terms of average FCT) while PS prefers fairness (in terms of starvation freedom). To strike a balance between them, in this paper, we propose a novel metric, i.e., expansion ratio, which is the ratio of the actual FCT of a flow to the optimal FCT. Under this definition, if two flows are delayed on the same link by the same amount of time, then the expansion ratio of the shorter flow is larger than that of the other one; and the expansion ratio of a flow increases as its waiting time increases. These properties enable us to design MERP, a distributed protocol that takes care of both average and tail FCTs.

The rate control in MERP decides the transmitting order of concurrent flows through minimizing the maximum expansion ratio of them. MERP senders insert

• *S. Zhang, Z.Z. Qian, H. Wu, and S.L. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, China. E-mail: {sheng, qzz, sanglu}@nju.edu.cn, wuhao@dislab.nju.edu.cn.*

some flow states (e.g., flow size, and expected sending rate) into each packet header, and such kind of states are updated by intermediate MERP switches. Each MERP switch maintains a flow table that contains the state information (e.g., virtual deadline) of current active flows that pass through the switch. Switches synchronize their decisions through the expected sending rate in each packet header. The flow with the earliest virtual deadline always monopolizes link resources; the other flows periodically send detecting packets to check whether they can start transmission, and the detecting rate is also effectively controlled by MERP. Extensive NS2-based evaluations show that, compared with pFabric [2], MERP reduces the expansion ratio and tail FCT by more than 15% and 12%, respectively, with only a negligible increase of average FCT.

The route control in MERP consists of a novel two-stage routing strategy, which first selects a subset of next-hop ports that minimize the expansion ratio of the newly-coming flow and then chooses one of them to maximize the sending rate of the tail flow of a path. NS2-based evaluations show that, in multi-path data center topologies (e.g., Fat-tree [11]), compared with ECMP and CONGA, MERP reduces the average FCT by up to 31% when there are link failures.

We summarize our contributions here as follows.

- We propose the expansion ratio metric that enables fast flow completion and starvation freedom.
- We design MERP, which controls the sending rates of competing flows through synchronizing virtual deadlines stored on intermediate switches, and routes flows in a downstream-aware manner that reacts quickly to link failures.
- We evaluate MERP using extensive NS2-based simulations that confirm our claims.

We now continue by motivating our work in Section 2. Section 3 gives an overview of MERP. We present rate control in Section 4 and route control in Section 5. We evaluate MERP in Section 6. Before concluding the paper in Section 8, we present related work in Section 7.

2 MOTIVATION

We start by presenting an example to demonstrate the drawbacks of SRPT and PS (§2.1). We then introduce the new metric and design intuitions of MERP (§2.2).

2.1 Drawbacks of SRPT and PS

Consider the example shown in Table 1, where three flows (A, B, and C) share a common link.

SRPT. SRPT seeks to minimize average FCT [12]: the flow with the smallest remaining size is always transmitted first. As shown in Fig. 1(a), a short flow (e.g., A) will monopolize the link resources until it is completed or preempted by another shorter flow. When flow A finishes, although flow C arrives earlier than flow B, flow B will occupy the link resources since it has a smaller remaining flow size than flow C. The average FCT of SRPT is $(3 + 3 + 10)/3 = 5\frac{1}{3}$.

TABLE 1
Motivating example

Flow ID	Arrival Time	Size
A	0	3
B	3	3
C	0	4

Although SRPT can achieve near-optimal performance in terms of average FCT [2], it may starve long flows. Consider Fig. 1(a), if there are a large number of flows with a size of 3 (i.e., similar to flow A), with SRPT, flow C would be delayed until all these short flows are completed. Performance of long flows may be hurt because they are scheduled later when competing with short flows. Unfortunately, many online services follow the partition-aggregate pattern, in which constructing the final response requires all intermediate results from smaller tasks. These intermediate results are flows with different sizes. Hence, the user-perceived latency depends on tail FCT as well as average FCT [13]. Therefore, *SRPT may starve long flows and thus impair user experience*.

PS. PS focuses on achieving fairness between competing flows. If we apply PS to the above example, the switch would divide the outgoing link bandwidth equally among three flows, as shown in Fig. 1(b).

PS can achieve fairness between competing flows with no need to know their sizes in advance. However, the average FCT of PS may be far from the minimum. For example, the average FCT of PS in Fig. 1(b) is $(7.5 + 9.5 + 7)/3 = 8$, which is much larger than that of SRPT. Therefore, *PS may hurt flow completion efficiency*.

In summary, current flow scheduling policies can hardly achieve flow completion efficiency (in terms of average FCT) and starvation freedom simultaneously, which are two primary goals in many applications such as web servers [14]. In this paper, we propose to schedule flows using the expansion ratio metric and design the MERP protocol.

2.2 Expansion Ratio and Design Intuition

MERP is intended to minimize average FCT without starving long flows. To realize this, MERP should *mimics* SRPT but meanwhile guaranteeing starvation freedom. Therefore, MERP should be able to:

- make short flows transmit data before long flows, since SRPT prioritizes the flow with the smallest remaining size and has near-optimal average FCT;
- guarantee that long flows have chance to transmit data when they wait for a sufficiently long time.

MERP uses expansion ratio to scheduling flows, implying that the expansion ratio of a flow should satisfy the following two conditions: (i) if two flows are delayed on the same link by the same amount of time, then the expansion ratio of the shorter flow is larger than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems

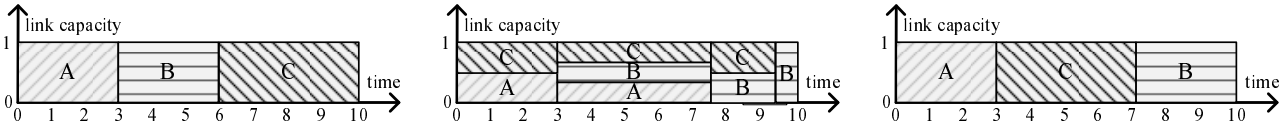


Fig. 1. Motivating example. (a) SRPT transmits the flow with the smallest remaining size first. (b) PS shares bandwidth between concurrent flows. (c) MERP schedules flows according to their dynamic expansion ratios.

that of the other one; and (ii) the expansion ratio of a flow increases as its waiting time increases. The formal definition of the expansion ratio of a flow is as follows:

***Definition** 1:* (Expansion ratio of a flow) Given a flow scheduling strategy S and a flow f, let $fct(S, f)$ be the actual FCT of f under S, and let $fct^*(f)$ be the optimal FCT of f when it monopolizes all link resources immediately after it arrives. The expansion ratio of f under S is defined as:

$$ER(S, f) = \frac{fct(S, f)}{fct^*(f)}. \quad (1)$$

For example, in Fig. 1(a), we have $ER(SRPT, A) = \frac{3}{3} = 1$ and $ER(SRPT, C) = \frac{10}{4} = 2.5$; in Fig. 1(b), we have $ER(PS, A) = \frac{7}{3}$ and $ER(PS, C) = \frac{9.5}{4} = 2.375$.

Let us look at whether Eq. (1) satisfies the above two conditions. Denote the size of flow f by $f.size$, and the capacity of the link of interest by C. Suppose flow f is delayed by d time before it can monopolize the link, then we have

$$ER(S, f) = \frac{fct(S, f)}{fct^*(f)} = \frac{d + \frac{f.size}{C}}{\frac{f.size}{C}} = \frac{Cd}{f.size} + 1. \quad (2)$$

We see that, given two flows, when C and d are fixed, the flow with a smaller size has a larger expansion ratio; when C and $f.size$ are fixed, the expansion ratio of any flow increases as d increases. Thus, Eq. (1) indeed satisfies the two conditions.

We now show how MERP schedules competing flows based on this metric. We first extend the definition of expansion ratio to a set of flows.

***Definition** 2:* (Expansion ratio of a set of flows) Given a scheduling strategy S and a set of n flows f_1, f_2, ..., f_n, the expansion ratio of these flows is defined below:

$$ER(S, f[1, n]) = \max_{i \in [1,n]} ER(S, f_i). \quad (3)$$

That is, the expansion ratio of a set of flows depends on the maximum expansion ratio of one of the flows. In fact, given a set of n competing flows, MERP tries to minimize $ER(S, f[1, n])$ (rather than the average FCT as in SRPT). Equivalently,

$$\text{MERP} = \arg\min_{S} ER(S, f[1, n]). \quad (4)$$

Take the scenario in Table 1 for example. If we use SRPT, then the expansion ratios of flows A, B, and C are $\frac{3}{3} = 1$, $\frac{3}{3} = 1$, and $\frac{10}{4} = 2.5$, respectively; the expansion ratio of them is 2.5. Can we do better? If we use MERP that tries to minimize the maximum expansion ratio, the optimal scheduling is shown in Fig. 1(c), and the expansion ratio of them is $\frac{7}{3}$. We find that, although flow C has a larger size than flow B, it monopolizes the link bandwidth before flow B does, which cannot happen in SRPT-based policies. We note that, the average FCT of MERP is $5\frac{2}{3}$, which is only a slightly higher than that achieved by SRPT.

With this example, we see that, MERP can gracefully navigate the tradeoff between SRPT and PS. Under MERP, short flows are typically scheduled before long flows like in SRPT, but as time goes on, the expansion ratio of a long flow increases; in order to minimize the maximum expansion ratio, MERP may prefer long flows that wait for a sufficiently long time to short flows that just wait for a while. In doing so, MERP can effectively prevent any starvation. Therefore, MERP could achieve efficient flow scheduling without starvation.

TABLE 2
MERP packet priorities

Packet type	Priority
SYN, SYNACK, BOOST	high
DATA, DATAACK, FIN, FINACK	middle
DET, DETACK	low

3 MERP Overview

In this section, we give an overview of MERP through presenting packet priorities, and the framework.

We define 9 packet types in MERP, including SYN, DET, DATA, FIN for MERP senders, and SYNACK, DETACK, BOOST, DATAACK, FINACK for MERP receivers. SYN/SYNACK packets are used in the handshaking phase to establish a connection. DET/DETACK packets are used by the senders and receivers of postponed flows to detect network status. DATA/DATAACK packets are used to transmit application-layer data. BOOST packets are used by a postponed flow receiver to quickly notify the corresponding sender that it can transmit DATA packets immediately. FIN/FINACK packets are used to close a connection.

We have 3 different packet priorities, as shown in Table 2. To reduce the connection setup delay of each flow and the waiting delay of each postponed flow, SYN, SYNACK, and BOOST packets have the highest priority. DATA, DATAACK, FIN, and FINACK packets are with middle priority. To enable postponed flows to quickly perceive network status, MERP allows each postponed flow to send detecting packets with a high rate, which is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems

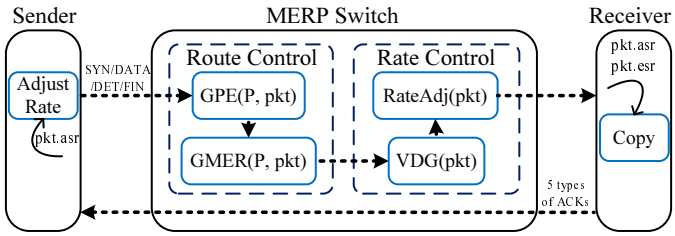


Fig. 2. MERP framework

effectively controlled by MERP. However, it may result in significant bandwidth overhead especially when there are massive flows on the same link. Therefore, when the buffer of a switch overflows, MERP should discard the least important packets. Which kinds of packets are the least important? Obviously, DET and DETACK packets, implying they have the lowest priority.

MERP provides route control as well as rate control. Fig. 2 shows the framework of MERP. MERP switch maintains a flow table that contains the state information of current active flows that pass through it. Different packet types trigger different actions/algorithms on MERP switches. Note that, the five types of ACK packets do not trigger actions on switches.

The rate control in MERP controls the sending rate of each flow. MERP decides the transmitting order of competing flows through minimizing the maximum expansion ratio of them. Since different network links have different contentions and capacities, we must synchronize such kind of heterogeneity between switches, which is achieved by inserting some flow states into packet headers (e.g., $pkt.asr$ and $pkt.esr$ in Fig. 2). MERP also reserves some bandwidth (controlled by Alg. 3) for postponed flows to detect current network status, and flows with different virtual deadlines have different detecting rates. The details can be found in Section 4.

The route control in MERP is responsible for selecting a path for each flow in the handshaking phase. When a SYN packet arrives, MERP first selects a subset of next-hops as candidates by minimizing the expansion ratio of this flow, then chooses one of the candidate paths by maximizing the expected sending rate of the tail flow on that path. In multi-path data center topologies, ECMP is the dominating protocol for load balancing. Compared with ECMP, the two-stage route control of MERP has several desirable properties. First, it is immune to hash collision; second, it is not sensitive to flow size distribution; and lastly, it reacts quickly to link failures. The details can be found in Section 5.

4 Rate Control

In this section, we assume the route path for each flow is known ahead of time and focus on presenting the rate control component. We first introduce MERP sender (§4.1), receiver (§4.2), and switch (§4.3), then briefly summarize this section by discussions (§4.4). We will present the route control component in the next section.

4.1 MERP Sender

MERP is implemented as a distributed protocol to keep the order in which competing flows are scheduled on different switches consistent. But how to make this happen? MERP utilize the flow states inserted in packet headers to synchronize decisions from different switches.

MERP sender inserts four flow states into each MERP packet header: flow size ($size$), actual sending rate (asr), expected sending rate (esr), and detecting rate threshold (Θ). Similar to previous studies [2], [6], [15], [16], [17], [18], we assume flow sizes are known ahead of time at the transport layer. The actual sending rate is updated by intermediate switches and used by a MERP sender to send DATA or DET packets using this rate. It is initialized to the capacity of the link that is adjacent to the sender. The expected sending rate is updated by intermediate switches and used by these switches to synchronize their decisions on flow scheduling. It is initialized to infinity, and it is *not* used by any MERP sender to adjust its sending rate of DATA or DET packets. We will explain how to update asr and esr shortly in Section 4.3. The detecting rate threshold is used by switches to limit the sending rate of DET packets.

MERP sender tries to establish a connection by sending SYN packets. If the asr in the SYNACK or DETACK packets is larger than Θ, then the sender transmits DATA packets, otherwise it transmits DET packets, both of which are transmitted with a rate of asr. When receiving BOOST packets, the sender immediately starts to transmit DATA packets. The connection is terminated by FIN packets. Note that, MERP is preemptive, i.e., a flow may be postponed before it completes its transmission.

Retransmission is controlled by retransmission timeout (RTO). We adopt a similar approach to estimate round-trip time (RTT) and RTO as that in TCP [19]: the exponential weighted moving average over sample RTTs is used as the current RTT, and RTO is set to the sum of the current RTT and four times RTT deviation.

4.2 MERP Receiver

When receiving a SYN/DATA/FIN packet, the receiver copies asr and esr from the received packet into the corresponding ACK packet, and send the ACK packet. When receiving a DET packet, if the asr in the DET packet is larger than Θ, a BOOST packet will be sent back other than a DETACK packet, otherwise, the receiver does the same as that for SYN/DATA/FIN packet.

4.3 MERP Switch

To minimize the maximum expansion ratio of competing flows, MERP must consider how to synchronize rate control decisions among MERP switches in a distributed way. To achieve this, we design three main components in each MERP switch:

- Distributed rate controller (DRC): this component decides the actions a MERP switch takes when there is a packet arriving at a switch.

Algorithm 1 Distributed Rate Controller, DRC($pkt \in f$)

Input: flow state table T
1: **if** $pkt.type$ = FIN **then**
2: T.delete(f)
3: **return**
4: **end if**
5: **if** $pkt.type$ = SYN **then**
6: VDG(pkt) ▷ Alg. 2
7: **end if**
8: **if** $pkt.type$ = SYN/DATA/DET **then**
9: $pkt.esr \leftarrow \min\{pkt.esr, T.f.esr\}$
10: $T.f.esr \leftarrow pkt.esr$
11: $T.f.vdl \leftarrow T.f.aT + \frac{T.f.size}{T.f.esr}$
12: RateAdj(pkt) ▷ Alg. 3
13: **end if**
14: **return**

- Virtual deadline generator (VDG): whenever a MERP switch receives a SYN packet (i.e., a new flow arrives), VDG is invoked to update virtual deadlines of all flows.
- Rate adjustment (RateAdj): whenever a SYN, DATA, or DET packet leaves a MERP switch, RateAdj is invoked to update the actual sending rate in the packet header.

4.3.1 Distributed Rate Controller

Each MERP switch maintains a flow state table T that stores the ID (id), flow size ($size$), size of the rest of the flow (rs), arrival time (aT), virtual deadline (vdl), and expected sending rate (esr) for each flow f. When there is a packet arriving at a switch, if the buffer (priority queue) is not full, MERP adds the new packet into the queue, otherwise, MERP discards the packet with the lowest priority in the queue and then adds the new packet into the queue. As for dequeue, MERP always takes out the packet with the highest priority.

Alg. 1 gives the details of the Distributed Rate Controller (DRC). When a switch receives a packet, if the type of this packet is FIN, then we delete the corresponding flow states from the state table T. If it is a SYN packet, we need to update virtual deadlines of all flows using Alg. 2.

However, different network links along the path between a pair of sender and receiver may have heterogeneous capacities, hence, a flow may have different virtual deadlines on different switches. We leverage esr to make these virtual deadlines consistent. Lines 8-13 show the trick. Whenever there is a SYN, DATA, or DET packet, we update the esr in the packet header and the esr in the flow state table T to be the minimum of them, recalculate the virtual deadline vdl, and update the actual sending rate asr using Alg. 3.

4.3.2 Virtual Deadline Generator

Each MERP switch updates the flow state table using the virtual deadline generator (VDG) whenever there is a new flow (indicated by SYN packets).

Algorithm 2 Virtual Deadline Generator, VDG($pkt \in f$)

Input: flow state table T, link capacity C, and current time cT
Output: new flow state table T'
1: $f.aT \leftarrow cT$
2: T.insert(f)
3: $tRestSize \leftarrow \sum_{f_i \in T} f_i.rs$
4: $T' \leftarrow \emptyset$
5: **while** $T \neq \emptyset$ **do**
6: $minExpRatio \leftarrow \infty$
7: **for** each flow $f_i \in T$ **do**
8: $expRatio \leftarrow \frac{(cT - f_i.aT)\cdot C + tRestSize}{f_i.size}$
9: **if** $expRatio < minExpRatio$ **then**
10: $minExpRatio \leftarrow expRatio$
11: $lastFlow \leftarrow f_i$
12: **end if**
13: **end for**
14: T.delete($lastFlow$)
15: $lastFlow.vdl \leftarrow cT + \frac{tRestSize}{C}$
16: $lastFlow.esr \leftarrow \frac{lastFlow.size}{lastFlow.vdl - lastFlow.aT}$
17: T'.insert($lastFlow$)
18: $tRestSize \leftarrow tRestSize - lastFlow.rs$
19: **end while**
20: **return** T'

The objective of VDG is to minimize the maximum ratio of a set of flows. Let us first analyse what the expansion ratio of flow f_i looks like. Without loss of generality, we assume there are n flows that are passing through a switch of interest, $F = \{f_1, f_2, ..., f_n\}$. Note that, VDG does not need to know this set of flows ahead of time. Denote by B_i the set of flows that are scheduled before f_i, and by A_i the set of flows that are scheduled after f_i. Then, we can simply use $\{B_i, f_i, A_i\}$ to represent a scheduling order. Let us further denote the link capacity is C, and the current time is cT. According to Eq. (2), the expansion ratio of f_i with respect to a scheduling order $\{B_i, f_i, A_i\}$ can be represented as:

$$\begin{aligned} &ER(\{B_i, f_i, A_i\}, f_i) \\ &= \frac{(cT - f_i.aT) + \frac{\sum_{f_j \in B_i} f_j.rs}{C} + \frac{f_i.rs}{C}}{\frac{f_i.size}{C}} \\ &= \frac{(cT - f_i.aT)C + \sum_{f_j \in B_i \cup \{f_i\}} f_j.rs}{f_i.size}. \end{aligned} \tag{5}$$

It is not hard to see that, $ER(\{B_i, f_i, A_i\}, f_i)$ is maximized when $B_i = F \setminus \{f_i\}$ and $A_i = \emptyset$. That is, when the set of flows is fixed, the expansion ratio of a flow is maximized if it is scheduled as the last flow, which also means, the maximum expansion ratio of these n flows is then bounded by

$$\max_{i \in \{1,2,...,n\}} ER(F \setminus \{f_i\}, f_i, \emptyset). \tag{6}$$

We then have an *optimal* algorithm, as shown in Alg. 2,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems

to determine the scheduling order of a set of flows that leads to the minimum maximum expansion ratio. The basic idea is to find the flow that could be scheduling as the last flow with the smallest expansion ratio. Specifically, for each flow, we compute its expansion ratio by assuming it is the last flow, then the flow with the smallest expansion ratio is set as the last flow; by doing so, we reduce the problem size by one, and we repeat this process on the rest $(n-1)$ flows.

Here are some brief notes on Alg. 2. Line 3 gives the total size of the rest of flows; lines 6-13 determine the last flow that incurs the smallest expansion ratio; line 8 is due to Eq. (5); line 15 updates the virtual deadline of the last flow, and this information will be used to determine the actual sending rate of each flow (see Alg. 3); line 16 updates the expected sending rate, and this information is used to synchronize virtual deadlines among different switches (see lines 8-13 in Alg. 1). Denote by $t(n)$ the running time of Alg. 2 with respect to the number of flows n. It is not hard to see $t(n) = t(n-1) + O(n)$, i.e., the time complexity of VDG is $O(n^2)$.

Today's high-end data center switches usually are equipped with fast CPUs, e.g., Cisco Nexus 9300 switch has dual-core 2.5-GHz x86 CPUs. According to the measurement in [3], the number of active flows is no more than 3,000 on more than 70% switches in 4 representative data centers. For this scale of flows, VDG would be very efficient and fast.

4.3.3 Rate Adjustment

Whenever a SYN, DATA, or DET packet leaves a MERP switch, the switch needs to update the actual sending rate in the packet header. MERP relies on DET/DETACK/BOOST packets to inform MERP senders of the current network status. To make senders perceive network status change in time, these detecting packets must be sent frequently. We therefore reserve a portion of network resources for them.

Denote by α the maximum proportion of actual sending rate to the link capacity, by n the number of concurrent flows, and by RTT the round-trip time.

Alg. 3 gives the rate adjustment (RateAdj) algorithm. As we mentioned before, MERP uses virtual deadlines to indicate the scheduling order of competing flows. If a flow has the smallest virtual deadline (hereafter we call this flow the first-flow), that is, it is now sending DATA packets, its actual sending rate is update as:

$$pkt.asr \leftarrow \min\{\alpha \times C, pkt.asr\}. \quad (7)$$

For other flows, i.e., postponed flows, $pkt.asr$ will be updated for the senders to transmit DET packets. For detecting packets, traditional protocols usually send one detecting packet per RTT; however, RTT in nowadays data centers is pretty small, when there are massive concurrent flows, this method would incur unnecessary bandwidth waste. Therefore, RateAdj proactively controls detecting rates according to the virtual deadlines of postponed flows. To enable the flow with the second smallest virtual deadline (hereafter we call this flow the second-flow) to perceive network status change more frequently than the other flows, the actual sending rate of the second-flow is updated as:

$$pkt.asr \leftarrow \min\{(n-1)\cdot(1-\alpha)\cdot C, \Theta, pkt.asr\}, \quad (8)$$

and the rate for the other flows is updated as:

$$pkt.asr \leftarrow \min\{(1-\alpha)\cdot C, \Theta, pkt.asr\}. \quad (9)$$

Note that, DET/DETACK packets have the lowest priority. When the buffer at a switch is full, they will be discarded, if any. Therefore, the "aggressive" sending manner of the second-flow would not block DATA packets from the first-flow.

We observe that, even with such kind of strategy, the second-flow still needs to wait one RTT to transmit DATA packets after the first-flow finishes. To make matters worse, if the first-flow is changed very frequently, the utilization of the network would decrease. Inspired by PDQ [6], MERP introduces an early start mechanism (lines 5-8 in Alg. 3). If the first-flow will finish within an RTT, MERP updates the actual sending rate of the second-flow to be $\min\{\alpha \times C, pkt.asr\}$. When the receiver of the second-flow finds that the asr is larger than Θ, it would send a BOOST packet to the sender, and the sender would start transmitting DATA packets immediately after it receives the BOOST packet.

Imagine the scenario without BOOST packets. When the receiver of the second-flow finds that the asr is larger than Θ, it would send a DETACK packet to the sender. Remember that DET and DETACK packets have the lowest priorities and may be discarded whenever buffer overflows. Therefore, a DETACK packet would not take a shorter time to arrive at the sender than a BOOST packet, which has the highest priority. Therefore, the BOOST packet can let the sender of the second-flow to quickly start transmitting DATA packets, and it may save one RTT latency.

Algorithm 3 Rate Adjustment, RateAdj($pkt \in f$)

```
1:  if f.vdl is the smallest then
2:    pkt.asr ← min{α · C, pkt.asr}
3:  else
4:    if f.vdl is the second smallest then
5:      f_k ← the flow with the smallest virtual deadline
6:      if f_k.rs < α · C · RTT then
7:        pkt.asr ← min{α · C, pkt.asr}        ▷ early start
8:      else
9:        pkt.asr ← min{(n − 1) · (1 − α) · C, Θ, pkt.asr}
10:     end if
11:   else
12:     pkt.asr ← min{(1 − α) · C, Θ, pkt.asr}
13:   end if
14: end if
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems

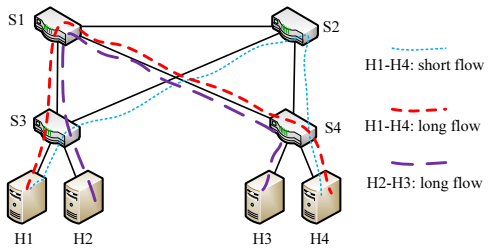


Fig. 3. ECMP limitation

4.4 Discussions

Flow table size. Each MERP switch maintains a flow state table T that stores 6 states for each flow (see Section 4.3.1). Under a pessimistic scenario where each of these states has to use 2 Bytes to store, a flow will need 12 Bytes at a switch. Modern switches usually have 4 to 16 MByte of shared high-speed memory, i.e., a switch can store states for at least $\frac{4MB}{12B} =\approx 349,525$ flows, which is large enough for normal production data centers[1]. In our route control simulations, the maximum number of active flows at each switch never exceeds 1,000. Still, suppose there are more flows than a switch can handle. In this case, we can run another flow scheduler that does not store flow states at switches alongside MERP. We inform this scheduler that the maximum link capacity is the amount of capacity not used by MERP. Packets from flows that are handled by this scheduler are associated with a priority that is between low and $middle$ (see Table 2). Therefore, these packets would not be blocked by DET packets or block any DATA packets.

How to set α and Θ? On one hand, the detecting rate of all flows except the first-flow is $n(1-\alpha)C$. This rate should not exceed the data sending rate of the first-flow, which is αC. So we have

$$n(1-\alpha)C < \alpha C \Rightarrow \alpha > \frac{n}{n+1}.$$

On the other hand, α cannot be too large, otherwise there would be too little bandwidth for detecting packets. We want to make full utilization of the bandwidth, thus, the detecting rate of the second-flow should not be too small. Without loss of generality, we assume that the detecting latency of the second-flow should not be larger than d. Then we have

$$\frac{pkt.size}{(n-1)(1-\alpha)C} \leq d \Rightarrow \alpha < 1 - \frac{pkt.size}{(n-1)dC}.$$

Θ is mainly used to limit the detecting rates. First, it should not be larger than αC. Otherwise, the detecting rate of some flow may be larger than αC. Second, it should not be too small. Otherwise, the second-flow could not immediately detect the finish of the first-flow. Therefore, we suggest that $\frac{\alpha}{2}C \leq \Theta \leq \alpha C$.

1. As Greenberg et al. [20] and Hong et al. [6] demonstrated that, in a production data center of a large scale cloud service, the average number of active flows at each switch is around 12,000.

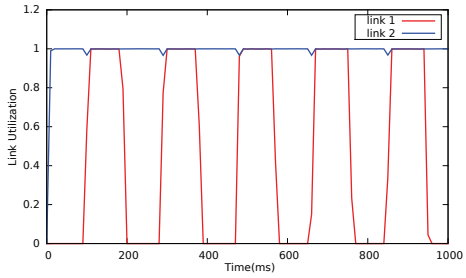


Fig. 4. The utilizations of S3-S1 (link 1) and S3-S2 (link 2) in the topology shown in Fig. 3 under ECMP.

5 Route Control

Nowadays data centers often use multi-path topologies, so it is also essential to design an appropriate routing scheme which can wisely decide the routing path for each flow among multiple equal-cost paths. We first show the limitations of ECMP (§5.1), then give the design rationale of route control in MERP (§5.2), lastly, we present design details (§5.3).

5.1 Limitations of ECMP

To meet the need for large aggregate bandwidth and fault-tolerance, data centers are designed to have many equal-cost paths, where ECMP is widely deployed for routing [11], [20]. To avoid out-of-order packets, which may cause obvious throughput degradation of TCP [21], ECMP ensures the packets belonging to the same TCP flow traverse the same path by hashing several TCP/IP fields of each packet to one of the available equal-cost routes to the destination [22]. The widely used hash algorithm is Hash-Threshold [23]. It first selects a key by performing hash over TCP/IP 5-tuple of a packet. Then a region size is obtained by dividing the codomain size of the hash function to the number of available next hops. Lastly the hash key is divided by the region size to get the output port number. Note that the algorithm does not specify the hash function to obtain the key but typically uses Cyclic Redundancy Check [22].

Although ECMP provides load balancing to some extent, it still has several limitations. Consider the example in Fig. 3, there are one short and two long flows in the simple network. Suppose two flows from H1 to H4 are routed to different paths according to ECMP. Now, a new flow from H2 to H3 arrives. Since ECMP is stateless, this new flow may be routed to S1 by S3. When the short flow finishes, two long flows still compete with each other. We also conducted experiments to verify our findings. We let H1 send 1GB data to H3 via many short flows, and meanwhile, let H2 send 1GB data to H4 via one single long flow. We periodically check the utilizations of two links: S3-S1 (link 1), and S3-S2 (link 2). The results are shown in Fig. 4. We notice that, the utilization of link 2 is stable and high, while that of link 1 varies significantly over time.

In fact, ECMP performance largely depends on flow size distribution and traffic pattern [24], and it achieves

Algorithm 4 Minimize Expansion ratio and Maximize tail Rate, MEMR($pkt \in f$)

Input: path set S
Output: a path P_f for flow f
1: $m \leftarrow \min_{P_i \in S}$GPE($P_i$, pkt) ▷ Alg. 5
2: $C \leftarrow \{P | P \in S$, and if f is routed to P, the expansion ratio of f is m $\}$
3: $P_f \leftarrow \arg\max_{P_i \in C}$GMER($P_i$, pkt) ▷ Alg. 6
4: **return** P_f

the best performance when flow sizes follow a uniform distribution and traffic pattern is all-to-all. However, in reality, flow sizes vary significantly in data centers: 90% of all flows are less than 100KB, and they contribute only 5% to the overall traffic [2], [25].

In summary, the limitations of stateless ECMP mainly include the following aspects.

- Collision may occur. Different flows may be hashed to the same path. For example, in Fig. 3, even if there are only two flows (one is from H1 to H4, and the other is from H2 to H3), the probability that they are routed to the same path is as high as 50%.
- ECMP is sensitive to flow size distribution. When flow sizes vary significantly, long flows may be scheduled to the same path. When short flows finish, link utilizations may be quite imbalanced.
- ECMP does not take downstream link conditions into account. When downstream network links have congestions or fail, upstream ECMP switches cannot perceive such changes. Taking Fig. 3 for example, if link S2-S4 is congested, and S3 uses ECMP to route flows, S3 cannot avoid routing packets to this congested link.

5.2 Design Rationale

Appropriate design rationales are quite important to the overall performance. Hence, before introducing our specific design, we first present three design rationales.

First, the primary purpose of the route control is to reduce latency, not to improve utilization. Many existing data center routing protocols focus on load balancing. Such kind of design often puts network utilization in the first place. However, increasing utilization is not equivalent to reducing latency. Here is an example: suppose there are 2 equal-cost paths, P_1 and P_2, starting from a router R; the capacity of either path is 1. There are 3 flows, say, f_1, f_2, and f_3; the sizes of them are 4, 1, and 1. Suppose f_1 is routed to P_1, and the others are routed to P_2. Now, a new flow f_4 with a size of 1 arrives at R. If we aims to maximize link utilization, f_4 will be routed to P_2: under SRPT, the FCTs of them are 4, 1, 2, and 3, and the average FCT is 2.5. However, if f_4 is routed to P_1, the FCTs of them become 4, 1, 2, and 1, and the average FCT becomes 2.25. As we mentioned before, reducing average and/or tail FCT is very important to improving user experiences, thus, the route control in MERP focuses on reducing latency.

Algorithm 5 Get Path Expansion Ratio, GPE(P, $pkt \in f$)

Input: flow state table $T[P]$ of P, link capacity C, and current time cT
Output: the expansion ratio of f on P
1: $tRestSize \leftarrow \sum_{f_i \in T[P]} f_i.rs$
2: **while** $T[P] \neq \emptyset$ **do**
3: $minExpRatio \leftarrow \frac{tRestSize + f.size}{f.size}$ ▷ due to Eq. (1)
4: $lastFlow \leftarrow f$ ▷ assuming f is the last flow
5: **for** each flow $f_i \in T[P]$ **do**
6: $expRatio \leftarrow \frac{(cT - f_i.aT) \cdot C + tRestSize + f.size}{f_i.size}$
7: **if** $expRatio < minExpRatio$ **then**
8: $minExpRatio \leftarrow expRatio$
9: $lastFlow \leftarrow f_i$ ▷ updating $lastFlow$
10: **end if**
11: **end for**
12: **if** $lastFlow = f$ **then**
13: **return** $minExpRatio$
14: **else**
15: $T[P]$.delete($lastFlow$)
16: $tRestSize \leftarrow tRestSize - lastFlow.rs$
17: **end if**
18: **end while**

Second, the route control should be per-flow, not per-packet. Per-packet load balancing can fully utilize the multi-path nature of data centers. However, most of current transport protocols schedule flows based on flow sizes; if we split flows into multiple sub-flows in the network layer, it may hurt flow scheduling efficiency in the transport layer. Moreover, per-packet strategies probably incur out-of-order packets, which may affect the performance of transport layer protocols, again.

Third, the route control should have ability to perceive downstream link conditions. The bursty data center traffic makes centralized route control inefficient to perceive network status changes, thus the route control in MERP must be distributed. If a switch makes route decisions only based on its local information, it cannot quickly react to downstream condition changes.

5.3 Design Details

We design the route control in MERP following these rationales. It is named as Minimize Expansion ratio and Maximize tail Rate (MEMR) and consists of two stages, as shown in Alg. 4.

The first stage (lines 1-2) is to check each possible path and calculate the expansion ratio of the current flow, we then select a subset C of these paths that minimize the expansion ratio of the current flow. Specifically, Alg. 5 shows GPE that calculates the expansion ratio of flow f on a path P. In each iteration, we first assume f is the last flow (lines 3-4), then for each flow f_i in $T[P]$ we compute its expansion ratio by taking it as the last flow (line 6). If the expansion ratio of f_i is smaller

Algorithm 6 Get Min Expected Rate, GMER(P, $pkt \in f$)

Input: flow state table $T[P]$ of P, link capacity C, and current time cT
Output: the expansion ratio of f on P

1: $tRestSize \leftarrow \sum_{f_i \in T[P]} f_i.rs$
2: $minExpRatio \leftarrow \frac{tRestSize+f.size}{f.size}$ ▷ due to Eq. (1)
3: **for** each flow $f_i \in T[P]$ **do**
4: $expRatio \leftarrow \frac{(cT-f_i.aT)\cdot C+tRestSize+f.size}{f_i.size}$
5: **if** $expRatio < minExpRatio$ **then**
6: $minExpRatio \leftarrow expRatio$
7: $lastFlow \leftarrow f_i$
8: **end if**
9: **end for**
10: $minExpectedRate \leftarrow \frac{C}{minExpRatio}$ ▷ tail rate
11: **for** each flow $f_i \in T[P]$ **do**
12: **if** $minExpectedRate > f_i.esr$ and $f.size \leq f_i.size$ and $f.dst = f_i.dst$ **then**
13: $minExpectedRate \leftarrow f_i.esr$
14: **end if**
15: **end for**
16: **return** $minExpectedRate$

than $minExpRatio$, we update $lastFlow$ (lines 7-9). The algorithm will terminate when we find the expansion ratio of f on P (lines 12-13).

The second stage (line 3) selects one path from the candidate path set C. GMER is shown in Alg. 6, it enables MERP to have the ability to perceive downstream link conditions. Taking Fig. 3 for example, when link S2-S4 becomes congested, if S3 does not know this, S3 may route upcoming flows to S2, which may impair the overall routing performance. But how to enable MERP to quickly learn downstream conditions?

Remember that, the esr of each packet is updated by each switch and is used to synchronize the virtual deadline of each flow, which means that, this kind of information captures the link conditions along each path. Therefore, we first compute the tail rate of path P (lines 2-10), then we additionally take a look at the expected sending rate (esr) of each flow that has the same destination of f, then let the expected sending rate of f to be the minimum among them (lines 11-15). In doing so, we can conservatively decrease the expected sending rate of f when there is a downstream link failure.

6 Performance Evaluation

In this section, we evaluate the performance of MERP using NS2-based simulations. We first evaluate the rate control of MERP on top of a single-path data center (§6.1), we then focus on the route control component using a multi-path data center topology (§6.2), finally we evaluate how quick MERP can perform flow switching and whether it is robust to Incast (§6.3).

6.1 Rate Control Evaluation

To evaluate the rate control component of MERP, we use a single tree topology including 9 servers arranged across three racks: each Top-of-Rack (ToR) switch connects three servers, and three ToR switches are connected by a root switch.

This topology uses 1 Gbps links and the length of the buffer at each port of each switch is 100 packets. By default, we generate flows with sizes following the Pareto distribution which gives a better fit for realistic data center workloads. The arrivals of flows follow the Poission process. The size of each packet is 1500 bytes, including a packet header of 40 bytes. The per-hop link delay is set to $10\mu s$. The maximum sending rate of DATA packets is set to 99.9% of link capacity, i.e., $\alpha = 0.999$. The detecting rate threshold is set to 80% of the link capacity, i.e., $\Theta = 0.8$ Gbps.

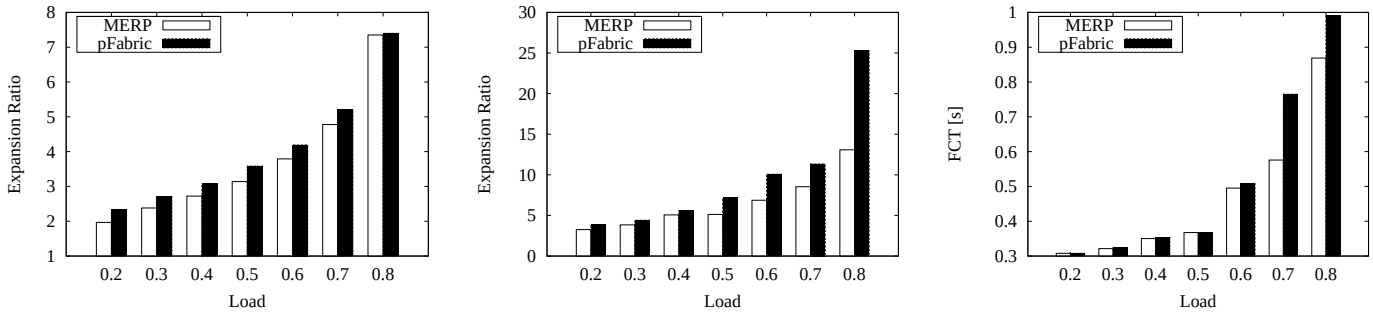
MERP is compared with TCP-DropTail [26], in which we use TCP NewReno with DropTail queues, and pFabric [2], which is a state-of-art approach for minimizing flow completion times: each packet carries a single priority number that depends on flow sizes, and each switch schedules and drops packets based on their priorities. Performance metrics include expansion ratio and tail FCT (i.e., the 99.9^{th} percentile FCT) [10].

6.1.1 Impact of Load

We generate flows with sizes following the Pareto distribution with a mean size of 1500KB (including the packet header) for this set of experiments. Fig. 5 shows that the long-tailed expansion ratio is significantly reduced by MERP. When the load is 80%, i.e., the total size of all flows is about 80% of the network capacity, the expansion ratio is reduced by about 48% compared with pFabric, indicating that MERP could provide a more stable flow completion time for each flow; besides, MERP provides a more than 12% reduction of the 99.9^{th} percentile FCT compared with pFabric. In the worst case, a flow is completed within no more 10 times of its optimal FCT. Furthermore, the 99.9^{th}th percentile completion times are also reduced significantly by MERP, especially under high loads.

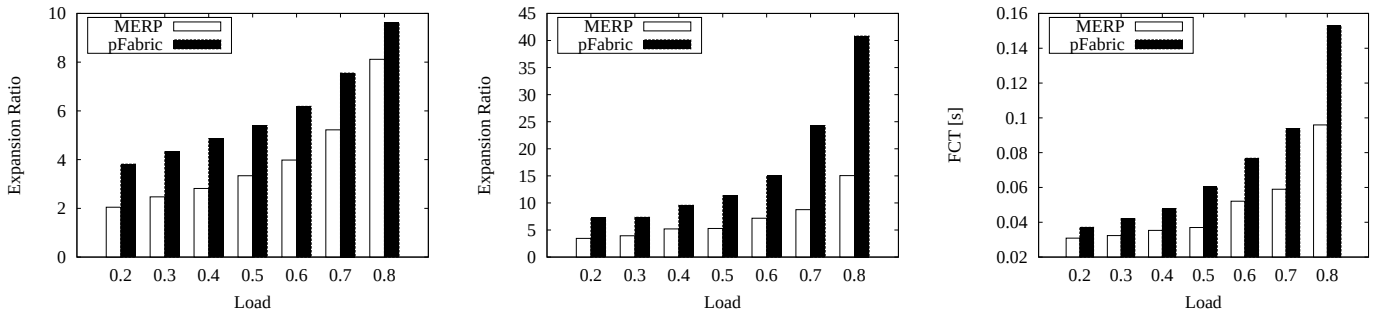
However, we must mention that, MERP targets to strike a balance between efficiency and fairness. It reduces tail FCT and expansion ratio at the price of a little increase in average FCT. Fig. 7 shows how the average FCT changes under different loads. We find that, the average FCT of MERP is almost the same as that of pFabric, while the average FCT of TCP-DropTail is much larger than either of them. The gap between MERP and pFabric increases as the load increases, and it is about 8% for 80% load, which is almost negligible, especially compared to the performance of TCP-DropTail. It is worth noting again that, MERP tries to balance fairness and efficiency; some long flows could be scheduled earlier than short flows to avoid starvation, thus, it is inevitable for MERP to achieve a slightly longer average FCT than pFabric.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems



(a) 99^{th} percentile expansion ratio (b) Maximum expansion ratio (c) 99.9^{th} percentile FCT

Fig. 5. MERP versus pFabric under varying loads (the mean of flow sizes is 1500KB)



(a) 99^{th} percentile expansion ratio (b) Maximum expansion ratio (c) 99.9^{th} percentile FCT

Fig. 6. MERP versus pFabric under varying loads (the mean of flow sizes is 150KB)

6.1.2 Impact of Flow Size

We are also interested in evaluating the impact of flow sizes on the performance of MERP, pFabric, and TCP-DropTail. To achieve this, we generate flows with sizes following the Pareto distribution with a mean size of 150KB (including the packet header), and the other settings are the same as those in Fig. 5. The results are shown in Fig. 6. Comparing Fig. 5 with Fig. 6, we find that, the performance gap between MERP and pFabric is larger when flow sizes vary more significantly, because pFabric may starve long flows while MERP fairly treat flows with different sizes. We note that, MERP can achieve better performance when small flows contribute the majority of the overall traffic. When the load is 80%, the maximum expansion ratio is reduced by more than 50%, and the maximum expansion ratio is about 15, i.e., the average FCT is within 15 times of its optimal average FCT; the 99.9^{th} percentile FCT is also reduced by more than 30% compared with pFabric.

6.1.3 Impact of α and Θ

MERP sends DATA packets using at most αC bandwidth, and sends DET packets using at most Θ bandwidth. MERP uses these two parameters to effectively control the sending rates of these packets, therefore, we are also interested in evaluating the effect of them on the performance of MERP. Figs. 8 and 9 show the results, where the mean of flow sizes is 150KB. In Fig. 8, when the load is light, these 4 values of α do not result in different tail FCTs in MERP; when the load increases, a large α has smaller tail FCT than a small α. This is reasonable, since a large α implies MERP can use more link capacity. However, in our simulations, we observed that, switch buffer overflows happen very frequently when α is large, e.g., 0.9999. Similarly, in Fig. 9, MERP with different Θ's has the same performance, when the load is light. However, when Θ is small, postponed flows in MERP may not be able to quickly perceive network status change, thus cannot immediately start transmission after the first-flow finishes, which implicitly wastes bandwidth resources. Therefore, when we increase the load, MERP with a small Θ has longer tail FCTs than MERP with a large Θ.

We note that, however, the performance difference due to α and Θ is very small. As long as we set them as discussed in Section 4.4, MERP can have a reasonably good performance.

6.2 Route Control Evaluation

We now focus on evaluating the route control component of MERP by examining whether MERP can effectively overcome the limitations of ECMP (i.e., hash collision, sensitivity to flow size distribution, and lack of upward feedback). In these experiments, we use fat-tree as the data center topology. There are, in all, 20 switches, and each of them has 4 ports. Four switches are used as root switches, therefore, there are a total of 16 servers. Every network link has a bandwidth of 1Gbps. The buffer size of each port of each switch is 150KB, and the single-hop delay is set to 10μs. Similar to the settings in rate control

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems

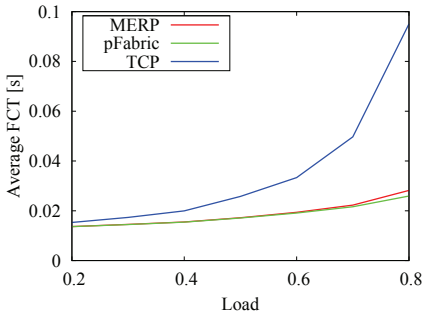


Fig. 7. Average FCT

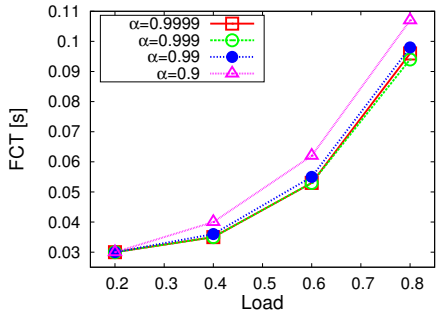


Fig. 8. Impact of α

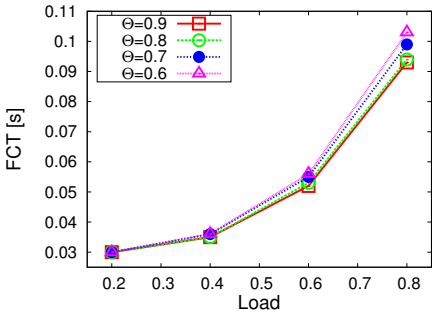
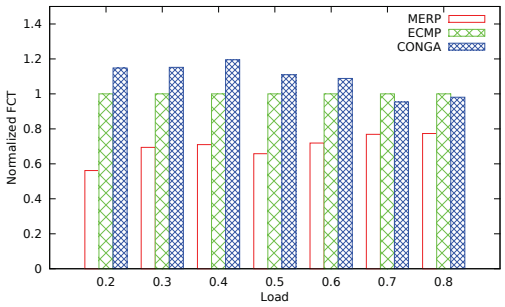
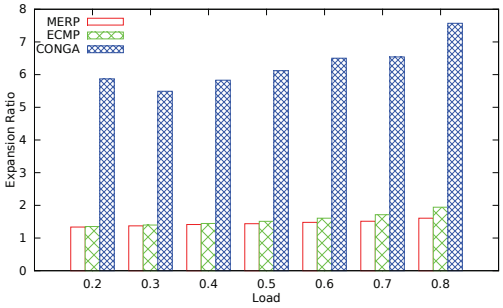


Fig. 9. Impact of Θ



(a) Average FCT



(b) Average Expansion Ratio

Fig. 10. Comparison results of MERP, ECMP, and CONGA under varying loads

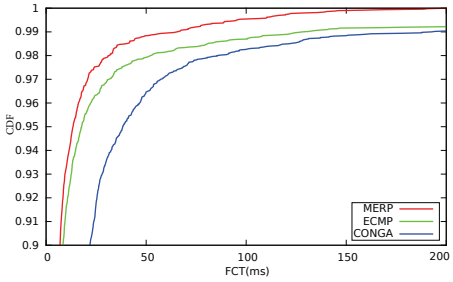
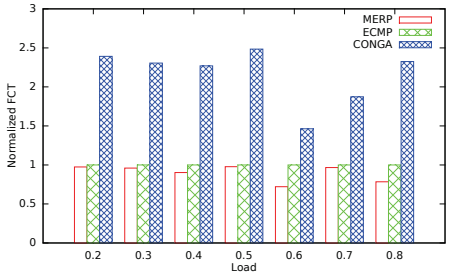
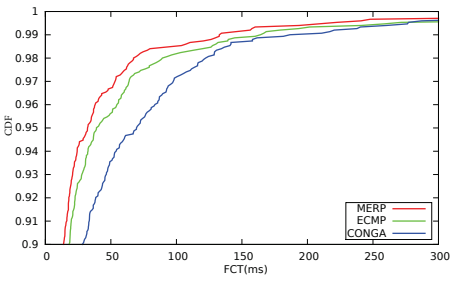


Fig. 11. Tail FCT CDF (the load is 0.6)



(a) Average FCT



(b) CDF of Tail FCT Under Load 0.6

Fig. 12. Comparison results of MERP, ECMP, and CONGA under link failures

evaluation, flow sizes follow the Pareto distribution, and flow arrivals follow the Poisson distribution.

MERP is compared with ECMP and CONGA [27], which is a state-of-art approach of load balancing using sub-flows. Since ECMP does not provide rate control mechanisms, in our experiments ECMP also uses MERP rate control component as the transport layer protocol. Performance metrics include expansion ratio and FCT.

6.2.1 Impact of Load

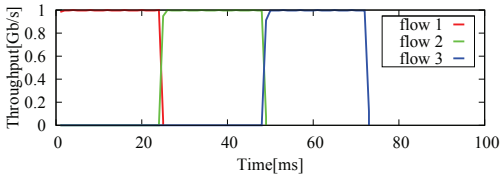
MERP makes a good balance between average and tail FCTs. Fig. 11 shows the cumulative distribution function (CDF) of tail FCTs of MERP, ECMP, and CONGA when the load is 60%. In general, the tail FCT of MERP is significantly smaller than that of ECMP or CONGA. For example, more than 99% of all flows are finished within 100ms by MERP, while the FCT of more than 1% of all flows are larger than 200ms in ECMP or CONGA. Considering the significant impact of the tail FCT on the user experience of online services (due to the aggregate/partition pattern), MERP is more suitable than ECMP or CONGA for most online interactive services deployed in data centers.

We are also interested in how these three protocols perform in terms of average FCT and average expansion ratio. Fig. 10 shows the comparison results. Under varying loads, MERP can reduce the average FCT by at least 20% compared to ECMP or CONGA, and not surprisingly, it can reduce the average expansion ratio by about 75% compared to ECMP or CONGA. We note that, CONGA has a worse performance than ECMP or MERP, because CONGA focuses on load balancing; besides, CONGA relies on sub-flow level routing, and it cannot take advantage of upper layer flow scheduling policies. These results show that, balancing traffic load is not equivalent to minimizing FCT.

6.2.2 Impact of Link Failure

When network conditions change, especially when link failures occur, a good flow scheduling protocol should

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2706290, IEEE Transactions on Parallel and Distributed Systems



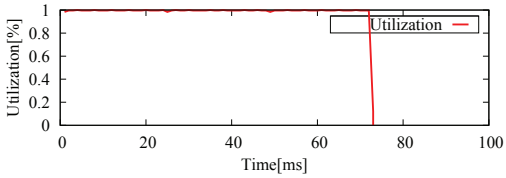
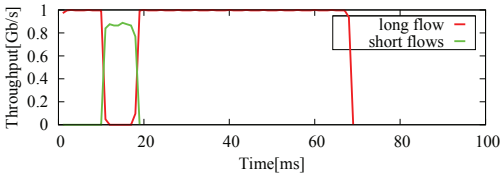


Fig. 13. Seamless flow switching



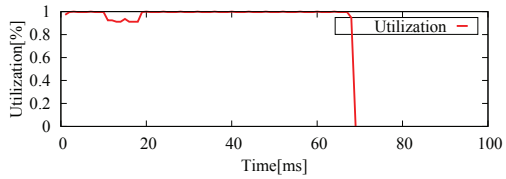


Fig. 14. Incast Robustness

be able to quickly and adaptively adjust its scheduling decisions. To evaluate how MERP, ECMP, and CONGA react when there is a link failure, in this subsection, we intentionally remove a link from an aggregate switch to a ToR switch in the fat-tree topology mentioned in Section 6.2. We then send data to the pod with the link failure. Flow sizes follow the Pareto distribution with a mean of 1500KB. The results are shown in Fig. 12.

We find that, MERP can efficiently reduce the average FCT under varying loads even when there is a link failure. This is because, when deciding the outgoing path for flow f, the route control component of MERP (i.e., MEMR in Alg. 4) estimates downstream link congestion conditions through examining the expected sending rate of each flow that has the same destination of f, then adjusts the expected sending rate of f to be the minimum among them. By doing so, MERP conservatively decrease the expected sending rate of f when there is a downstream link failure. Note that, MERP can reacts quickly to changing network conditions including link failures, network congestion, etc.

6.3 Flow Switching and Incast Robustness

6.3.1 Flow Switching

Seamless flow switching is critical to flow scheduling, as there are massive flows in data centers. We assume that, three flows all start at time 0 and each flow has 2000 packets. Three different servers send data to the same server. All the four servers are connected by one switch and the per hop delay is set to $10\mu s$. Therefore, these flows share the same link resources.

Fig. 13 shows the throughput (i.e., DATA packets) achieved by each flow over time. The theoretic time needed for 1 Gbps switch to process 2000 packets is $(2000 \times 1500 \times 3bytes) \div 1Gbps = 72ms$. In fact, MERP takes $72.25ms$ to complete three flows which is only $0.25ms$ larger than the best possible value. Such quick switch is due to the early start mechanism (see Section 4.3.3). The reader may also wonder what the link utilization looks like. Fig. 13 also gives the answer. It shows the link utilization over time. We find that, even MERP reserves a small portion (which is effectively and adaptive controlled by MERP using Alg. 3) of bandwidth for detecting packets, link utilizations under MERP is still very high.

6.3.2 Incast Robustness

Incast is the many-to-one communication pattern, which is widely observed in today's production data centers. We now show how MERP performs under the Incast traffic pattern. In this experiment, eight sending servers, each associated with several flows, simultaneously send data to the same receiving server. A long flow starts at 0ms as the background flow. And 30 short flows start at 10ms with each having 20 packets (approximately 30 KB). Under ideal conditions, it takes about 0.24ms for a 1Gps link to complete one single short flow. When there are bursty short flows, maintaining stable throughput is pretty challenging for most existing transport protocols. However, as shown in Fig. 14, MERP can quickly adapt to the busty Incast traffic, and still achieve high utilization of link resources. In fact, the average link utilization during the preemption period (between 10 and 18ms) is as high as 91.8%.

7 Related Work

PS-based flow schedulers. RCP [28] is the first flow scheduling protocol that introduces explicit rate control. DCTCP [1] leverages Explicit Congestion Notification (ECN) to limit the queue lengths to reduce FCTs of short flows. To avoid switch head-of-line blocking, HULL [8] leaves bandwidth headroom using phantom queues that deliver congestion signals before network links are fully utilized. XCP [29] decouples fairness control from utilization control using ECN. CUBIC [30] modifies the linear window growth of traditional TCP standards to be a cubic function so as to improve scalability. However, as these studies are similar to PS, the average FCT is far from the minimum.

SRPT-based flow schedulers. In pFabric [2], each packet carries a single priority number that depends on flow sizes, and each switch schedules and drops packets based on their priorities. Flows start at line rate and throttle back only under high and continuous packet

loss. Although pFabric achieves near-optimal average FCT, it has to maintain lots of priorities at switches; unfortunately, switches nowadays only support a small number of priorities, usually no more than 10 [15]. PDQ [6] attaches a scheduling header to every packet, and each switch maintains states for critical flows and decides the explicit sending rate for each flow. However, long flows may be unfairly treated since PDQ focuses on efficiency. L^2DCT [31] approximates the least attained service scheduling policy. When congestion occurs, long flows back off aggressively while short flows do conservatively. This approach intuitively favors short flows to finish quickly without causing any starvation of long flows. However, it cannot provide as small average FCT as other SRPT-based policies.

Coflow schedulers. A coflow represents a collection of parallel flows that convey job-specific communication requirements [32]. Varys [17] schedules coflows using the smallest-effective-bottleneck-first heuristic based on the assumption that coflow characteristics (e.g., the number of flows, and their sizes) are known a prior. Non-clairvoyant schedulers, Aalo [33] and PIAS [34], use multi-level feedback queues (MLFQ) to simulate the shortest-job-first (SJF) heuristic: a flow is gradually demoted from a higher-priority queue to a lower-priority one according to the bytes it has sent. Varys, Aalo, and PIAS need to modify applications to extract coflows, which makes them hard to deploy. CODA [35] overcomes this by automatically identifying coflows using machine learning technique in an application-transparent manner. These approaches focus on application-aware coflow-level efficiency, while our work intends to improve flow-level efficiency.

Multi-path-based schedulers. Hedera [24] computes non-conflicting paths according to flow information and instructs switches to re-route traffic. MPTCP [36] splits a flow into multiple subflows and the rate of subflows is adapted based on congestion information. DeTail [10] targets to reduce the tail FCT for long flows. It has a cross-layer network stack, with congestion-aware packet-level routing in the network layer and priority queue management in the link layer. RepFlow [18], [37] replicates each short flow, and wishes that the replicated flow and the original one traverse different paths, since the probability that all these short flows experience long queueing delay is pretty smaller. OminiFlow [38] couples load balancing with flow control through finely detecting and controlling the queue length along each path. Overall, these approaches are sensitive to flow size and do not utilize downstream conditions. Instead, MERP can effectively avoid hash collisions and react quickly to link failures.

Deadline-aware schedulers. D^3 [15] allocates explicit rate on a greedy first-come-first-served principle, but it does not support preemptive rescheduling. D^2TCP [39] uses ECNs and deadlines to modulate the congestion window size and prioritize flows with nearer deadlines. However, in either D^3 or D^2TCP, short flows may starve if they arrive after long flows. The recently proposed Karuna [40] simultaneously deals with flows with and without deadlines through handling deadline flows with as little bandwidth as possible. These designs can be used together with MERP to improve application-level performance.

8 Conclusions

In this paper, we design and evaluate MERP, which explores the tradeoff between average FCT and starvation freedom. By greedily minimizing the maximum expansion ratio of competing flows, MERP guarantees that, no flow is unfairly treated while not sacrificing too much average FCT. MERP uses explicit rate control to synchronize flow virtual deadlines. The route control of MERP routes each flow by first minimizing its expansion ratio and then maximizing the minimum expected sending rate, making MERP have the ability to perceive downstream link conditions. NS2-based evaluations confirm the advantages of MERP. We are going to enable MERP to adaptively optimize for fairness and efficiency.

Acknowledgments

This work was supported in part by NSFC (61502224, 61472181, 61472185, 61321491), China Postdoctoral Science Foundation (2015M570434, 2016T90444), CCF-Tencent Open Research Fund (AGR20160104), Jiangsu NSF (BK20151392, BK20151390), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 63–74, 2011.

[2] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM IMC 2010*. ACM, 2010, pp. 267–280.

[4] J. Brutlag, "Speed matters for google web search," *Google. June*, 2009.

[5] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, no. 9, pp. 103–105, 2007.

[6] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[7] N. Bansal and M. Harchol-Balter, "Analysis of srpt scheduling: Investigating unfairness," *Proc. ACM SIGMETRICS 2001*, pp. 279–290, 2001.

[8] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *Proc. USENIX NSDI 2012*. USENIX Association, 2012, pp. 19–19.

[9] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: incast congestion control for TCP in data-center networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 345–358, 2013.

[10] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "vaDeTail: reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 139–150, 2012.

[11] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM 2008*. ACM, pp. 63–74.
[12] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.
[13] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM 2014*. ACM, 2014, pp. 431–442.
[14] E. J. Friedman and S. G. Henderson, "Fairness and efficiency in web server protocols," in *Proc. of ACM SIGMETRICS 2003*. ACM, 2003, pp. 229–237.
[15] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41. ACM, 2011, pp. 50–61.
[16] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 98–109.
[17] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 443–454.
[18] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE INFOCOM 2014*. IEEE, 2014, pp. 1581–1589.
[19] "TCP retransmission timer," *https://tools.ietf.org/html/rfc2988*.
[20] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.
[21] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, pp. 20–30, 2002.
[22] C. E. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC Editor*, 2000.
[23] D. Thaler and C. Hopps, "Multipath issues in unicast and multicast next-hop selection," *RFC Editor*, 2000.
[24] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *Proc. USENIX NSDI 2010*, vol. 10, 2010, pp. 19–19.
[25] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. of ACM SIGCOMM IMC*. ACM, 2009, pp. 202–208.
[26] N. Parvez, A. Mahanti, and C. Williamson, "An analytic throughput model for tcp newreno," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 448–461, 2010.
[27] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM 2014*. ACM, 2014, pp. 503–514.
[28] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.
[29] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 89–102, 2002.
[30] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.
[31] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM 2013*. IEEE, 2013, pp. 2157–2165.
[32] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM HotNets XI*. ACM, 2012, pp. 31–36.
[33] ——, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 393–406.
[34] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers." in *Proc. USENIX NSDI 2015*, 2015, pp. 455–468.
[35] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *Proc. ACM SIGCOMM 2016*. ACM, 2016, pp. 160–173.
[36] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 266–277, 2011.
[37] F. Wang, Z. Qian, S. Zhang, M. Dong, and S. Lu, "SmartRep: Reducing flow completion times with minimal replication in data centers," in *Proc. IEEE ICC 2015*. IEEE, pp. 460–465.
[38] K. Wen, Z. Qian, S. Zhang, and S. Lu, "OmniFlow: Coupling load balancing with flow control in datacenter networks," in *Proc. IEEE ICDCS 2016*. IEEE.
[39] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115–126, 2012.
[40] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM 2016*. ACM, 2016, pp. 174–187.

Sheng Zhang received his BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. He is an assistant professor in Nanjing University and a member of the State Key Lab. for Novel Software Technology. His research interests include cloud computing and mobile networks. His publications include those appeared in IEEE TMC, TPDS, TC, ICDCS, INFOCOM, and ACM MobiHoc. He received the Best Paper Runner-Up Award from IEEE MASS 2012. He is an IEEE member.

Zhuzhong Qian received the PhD degree from Nanjing University in 2007. He is an associate professor in the Department of Computer Science and Technology, Nanjing University. His current research interests include distributed systems and data center networking. He has published more than 40 papers in referred journals and conferences, including TPDS, INFOCOM, IPDPS. He is a member of the IEEE.

Hao Wu received his BS degree from Nanjing University in 2012. He is now a master student in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. His research interests include data center flow scheduling.

Sanglu Lu received her BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas. She is a member of IEEE.