Computer Science and Information Systems 13(2):639–658 DOI: 10.2298/CSIS160301020H

A Mechanism Achieving Low Latency for Wireless Datacenter Applications

Tao Huang[1,2], Jiao Zhang[1], and Yunjie Liu[2]

[1] State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing, P. R. China
{htao, jiaozhang}@bupt.edu.cn
[2] Beijing Advanced Innovation Center for Future Internet Technology
Beijing 100124, China
htao@bupt.edu.cn, liuyj@chinaunicom.cn

Abstract. Recently, several wireless/optical datacenter architectures are designed to overcome the drawbacks of wired datacenter topologies, such as expensive high-end switches, high cabling complexity, congestion caused by a few hot nodes. Compared with wired switches, current commodity wireless switches usually suffer lower throughput as well as higher packet loss ratio and latency. However, todays datacenter applications, such as web search, require quite low latency to improve user experience. In this paper, a mechanism, named pECN, is proposed to achieve low latency for delay-sensitive flows. pECN makes use of Explicit Congestion Notification (ECN) to maintain a small queue length and a priority-based scheduling mechanism to provide delay differentiated services. The priority-based scheduling mechanism works at the packet level. When transmitting, the packet with the highest priority will be sent. Since the queue length is controlled within a small value by the ECN mechanism, the scheduling mechanism does not incur large overhead. Also, since it works at the packet level, it can support unlimited number of services. Besides, pECN does not need to maintain any flow state at switches. We implemented the proposed mechanism and evaluated the performance on the ns-2 platform. The simulation results show that pECN works better compared with TCP, DCTCP and D2TCP in terms of the flow completion time and the number of flows that can be supported without compromising deadlines.

Keywords: datacenter network, latency, ECN, priority based scheduling

1. Introduction

Datacenters has been growing with the increase of cloud services. However, typical wired datacenter topologies [9, 22] face a lot of challenges. For example, Google has got over 900000 servers in its 30 data centers [5]. The cabling in such large datacenters incur significant initial effort and may cause high energy cost [23]. Besides, unbalanced traffic distributions easily cause congestion hotspots [18].

To overcome the drawbacks of wired datacenters, much attempt has been done to design hybrid datacenters [23, 24, 40], that is, some wireless switches are combined with existing wired datacenters. Wireless switches could avoid complex cabling and can be set up on-demand. However, most commodity wireless switches support lower data rate and suffer higher packet loss ratio and link propagation delay.

Latency is a quite critical metric for cloud computing services. High latency will impact the user experience and decrease the profit of cloud service providers. The typical latency requirement of web applications is 200-300 milliseconds [45]. Today's web applications are very complex. For example, a Facebook page makes an average of 130 internal sequential requests, and Amazon reported similar results [36]. All of these sequential requests have to be done before the applications' deadlines. Therefore, the flow-level deadline could be as low as only several milliseconds [42]. To satisfy the latency requirements of applications in wireless datacenters, we need to design deadline-aware mechanisms to guarantee that latency-sensitive flows finish within expected time.

The researchers in different fields, including the storage [36], the operating system [39], and the networking, are making attempts to reduce latency in datacenters. In the networking field, reducing flow completion time has attracted a lot of attention recently [12, 13, 25, 42, 45].

There are two main reasons for higher flow latency. First, large queue buildup results in long queuing delay [13, 41]. The Round Trip Time (RTT) of a datacenter network is very short, only about 200 microseconds. However, the queuing delay could be up to more than 10 milliseconds [13]. For example, if the switch buffer size is 512KB and the output link capacity is 1Gbps, then the queuing delay achieves as high as about 4 milliseconds when the queue is full. A packet typically needs to pass five switches in a three-layered datacenter. If all of them are congested, then the total queuing delay can be up to 20 milliseconds. Second, many lost packets cause long recovery time. If a packet is lost, retransmission will be triggered by three duplicate ACKs or a retransmission timer. The time required by a sender to receive three duplicate ACKs after transmitting the lost packet is at least one RTT. Thus, retransmission triggered by fast recovery needs at least one RTT. Besides, in some situations, the sender fails to receive enough ACKs. For example, if a query flow or its responses only consist(s) of no more than three packets [13], then when any one of them is lost, the retransmission has to be triggered after a timeout period. Also, some special communication pattern, such as incast [17, 43, 46], causes the sender unable to get more packets from the application layer to send before the lost packet is recovered. Therefore, the retransmission will be done after the retransmission timer fires. The timeout period in TCP, which is 200 milliseconds in default, is quite large compared with the delay requirements of the flows. Thus, if a delay-sensitive flow, especially short flow, loses some packets, the flow completion time will be possible to increase greatly.

To reduce the flow latency, zero queue length is the ideal network state. Zero queue length does not incur any queuing delay. Furthermore, zero queue length means that packets loss ratio will be reduced due to buffer being overwhelmed. Unfortunately, the traffic in data centers is dynamic and has many short bursts [14]. It is very difficult to maintain a zero queue length all the time with quite complex mechanisms. The credit-based flow control [29] designed for ATM could achieve this goal. However, a separate queue is required for each connection. There are possibly several thousand concurrent connections at a switch in datacenters. Maintaining so many queues is not practical. HULL [11] tries to maintain zero queue for delay-sensitive flows by employing shadow queue. However, it has to sacrifice some bandwidth.

Queue-length-based AQM mechanisms, such as RED [21] and its variants [32, 35, 44], make a tradeoff between the large queue buildup and zero queue. These mechanisms maintain a limited queue length by sending congestion information to the sender in ad-

vance, such as after the queue length exceeds a fixed threshold. Through maintaining a short queue length, the queuing delay could be largely reduced. Besides, since the rate control at the sender is not driven by packet losses, the number of dropped packets is dramatically reduced. Thus, the time wasted in recovering lost packets can be greatly decreased. DCTCP [13] employs ECN to limit the queue length and thus reduce the flow latency [13].

However, even if the queue length can be controlled within a relatively small value by an AQM mechanism, it is still a little large compared with the small RTT in datacenters. For example, if the stable queue length is 30KB, then the queuing delay at one buffer switch can be up to 30KB/1Gbps = 240 microseconds. If there are five bottlenecks along the path, the queuing delay could be up to $240 \times 5 = 1.2$ milliseconds. Furthermore, the AQM mechanisms fail to provide differentiated services for latency-sensitive flows and background flows without delay requirements.

In datacenters, the traffic is mixed of delay-sensitive short flows and background flows. Although the number of back- ground flows is quite small, about one percent [7], most of packets in the network belong to the background traffic [13]. That is to say, most of the packets accommodated by the switch buffer belong to the background flows without delay requirements. A packet of a delay-sensitive flow is likely to wait for a relatively long time behind some packets of background flows before being scheduled. To provide differentiated services for flows with different latency requirements, D2TCP [42] extends the congestion window evolution of DCTCP. It suggests that the near-deadline flows should reduce congestion window slowly than the far-deadline flows. Therefore, the near-deadline flows can get larger congestion window than the far-deadline flows. Intuitively, it works well. Higher congestion window means the flows can be finished using fewer RTTs. However, it ignores an important traffic feature in datacenters that many delay-sensitive flows are quite short. The queries and responses of web search applications take only 1.6 to 2KB [13]. The maximum segment size (MSS) of TCP protocol can be up to 1,460Bytes. Thus, a query or response only consists of two packets. If the slow start window size of TCP is two, the flow can be finished using only one RTT. Even if TCP flow starts from one packet, the latency-aware congestion window evolution after receiving the marked information will not take effect. Therefore, the latency reduction for the near-deadline flows through increasing the flow rate at the end host responses too slowly.

Scheduling at the switches is an alternative to provide more timely differentiated services. Priority-based scheduling mechanism is well known and used in traditional Internet, such as DiffServ [8].

If the packets of the lowest latency flow are given the highest priority to be scheduled, the flow will suffer zero queuing latency. However, if only scheduling mechanisms at the switches are employed, packet dropping could not be avoided. The latency caused by fast recovery or timeout still exists. Besides, if the queue length is quite large, the prioritized scheduling mechanism will possibly cause a large overhead.

Considering the features of AQM and scheduling mechanisms, a hybrid mechanism is required to reduce flow latency and provide differentiated services without sacrificing bandwidth. The challenge is that as various cloud services are developed, the latency requirements of flows are possibly quite different. We could not use a limited number of priority queues to provide differentiated services for all the flows.

In this paper, a mechanism, named pECN, is proposed, which consists of ECN and a priority-based scheduling mechanism. pECN takes advantage of the ECN mechanism to limit the queue length within a fixed small number, and makes use of a priority-based scheduling mechanism to provide unlimited differentiated services. The sender at the end host puts the priority value in the header of each packet. The priority could be a function of deadline or remaining flow size. The priority-based scheduling mechanism at the switches works at the packet level. The packets are sent according to the priority value in their headers. When transmitting, the packet with the highest priority will be sent. In this way, we could provide unlimited differentiated services. Since the queue length is limited within a small value, the searching overhead is acceptable. Besides, limited queue length means that the commodity switches could have smaller buffer size and thus reduce the switch cost. When a packet arrives, if the queue length exceeds the marking threshold, there are two marking policies. One is marking the new packet as suggested in ECN. However, the new arrival packet may have higher priority, and intuitively the packet with the lowest priority should be marked. The other one is priority-based marking (pMark). We will evaluate the performance of both of them in the simulation section.

We implemented pECN and pECN with pMark on the ns2 platform and compared their performance with TCP, DCTCP, and D2TCP. Two kinds of traffic patterns are generated to evaluate the performance of the mechanisms. One is burst traffic pattern, that is, many senders transmit packets to the same receiver at the same time. The other is realistic traffic generated based on the traffic measurement [13]. Series of simulations with these two kinds of traffic patterns have been conducted. The simulation results show that pECN reduces the flow completion time for the latency- sensitive flows and can decrease the number of missed deadline flows.

The remainder of the paper is organized as follows: Section 2 summarizes the related work; Section 3 describes the motivation of this work. In Section 4, the details of pECN are presented. The performance evaluation of pECN is shown in Section 5. Finally, the paper is concluded in Section 6.

2. Related Work

Active Queue Management (AQM). There are a series of work on active queue management mechanisms, which can be classified into two main classes in terms of the monitored metric. One is queue-length-based AQM, such as RED [21] and its variants [32, 35, 44].This kind of mechanism aims to control the queue length and avoid unfair packet losses caused by the drop tail queue management mechanism. However, it is quite difficult to tune the parameters of RED. Therefore, it is not widely employed in industry. The other one is rate-based AQM, such as BLUE [20], AVQ [30, 31], and SFED [27]. This kind of mechanism control congestion is based on the difference of the input and the output aggregated flow rates. However, the parameters in most of the mechanisms are required to be tuned with different number of sources, link capacities and feedback delay. Besides, both of RED and AVQ mechanisms are prone to instability as the link capacities increase [28, 33].

In this work, we only need the AQM mechanism to limit the queue length so that timeouts caused by lost packets can be largely reduced and the packet-level prioritized

scheduling will not incur large searching overhead. Therefore, the ECN marking policy, which is already available in many modern switches, is employed.

Latency-Aware Transport Control Protocol. In datacenters, some work has been done to reduce flow latency. DCTCP [13] examines that short flows and long flows coexist in datacenters. Short flows suffer large latency due to the large queuing delay in datacenters. Thus, ECN mechanism is employed to decrease the queue length. However, this method does not provide differentiated delay services and also does not guarantee a specific delay requirement. Afterwards, some work has been done to reduce the latency from different perspectives.

First, setting different flow rates at the end hosts. The typical mechanisms include D3 [45] and D2TCP [42]. D3 [45] aims to satisfy the delay requirements of flows. All the delay- sensitive flows compute their desired rates based on the flow size and deadline, and send them to the switches. The switches allocate bandwidth according to the collected rates. The remaining bandwidth is fairly allocated to all the flows. If the bandwidth is not large enough to satisfy all the delay- sensitive flows, D3 greedily tries to satisfy the rate requests of as many deadline flows as possible. All the remaining flows are assigned a base rate to allow them to send a header-only rate request per RTT. D3 proposes the problem of guaranteeing the delay requirements of flows. However, its greedy approach based on first-come-first-served may allocate bandwidth to far-deadline requests arriving slightly ahead of near-deadline requests [42].

D2TCP [42] modifies the congestion control algorithm of DCTCP to provide differentiated delay services for different flows. The rates of all the flows in DCTCP are decreased in proportional to the marking proportion, while the congestion window of the near- deadline flows in D2TCP decreases less than the far-deadline flows. Thus, D2TCP reduces the number of missed deadlines. Intuitively, D2TCP is a good method to provide delay differentiated services. However, it fails to perform as well as expected in datacenters with special traffic characteristic. In datacenters, the latency-sensitive flows, such as search queries, only have one or two packets [11, 13]. Generally, TCP starts from slow start window of one or two packets, while the congestion window adjusted by D2TCP will not take effect until after at least one RTT. Thus, the latency suffered by the quite short flows is mainly determined by the queuing delay during the paths instead of the transmitting rate at the end host.

Second, reducing latency at the switches. Most of this kind of work employs prioritized scheduling mechanisms. The packets with higher priority are scheduled first and thus the latency on the path can be decreased. The challenge is that the number of priority classes may become quite large as various applications are developed in datacenters. PDQ [25] designs a distributed scheduling mechanism to approximate shortest job first and earlier deadline first policies in datacenters. PDQ borrows ideas from centralized scheduling disciplines and implements them in a fully distributed manner, making it scalable to today's datacenters. In PDQ, each switch needs to maintain the state about the most critical 2k flows on each link, where k is the number of sending flows (i.e., flows with sending rate larger than zero). DeTail [48] is a cross-layer mechanism. In routing layers, it employs the congestion-aware packet-level routing mechanism to fully utilize the multiple redundant bandwidths in datacenters. In the link layer, it utilizes the Priority-based Flow Control (PFC) mechanism that defined in IEEE 802.1Qbb to schedule the

packets with higher priority first. However, the PFC mechanism is designed to enable the Ethernet to be a unified fabric of Storage Area Network, Local Area Network and High Performance Computation. There are only eight queues totally. Even if some of them can be used to provide differentiated services for the communication traffic, it could not provide finer-grained delay differentiated services. In pFabric [12], the end hosts put a priority in each packet header. The switch always sends the packet with the highest priority and drops the packet with the lowest priority. To avoid large searching overhead, pFabric suggests that the switch buffer size should approximately equal the product of bandwidth and round trip time. However, it is quite hard to modify the hardware for a specific protocol. Besides, currently TCP traffic takes about 99.91% [13]. TCP needs larger switch buffer to accommodate traffic bursts. If the buffer size is too small, TCP will lost a great number of packets and thus reduce the link utilization. Therefore, decreasing the switch buffer size to a quite small value is a little aggressive.

HULL [11] does not employ a scheduling mechanism, but trades a little bandwidth to reduce the latency of short flows. It leverages Phantom Queues to get congestion information in advance. The end hosts make use of DCTCP mechanism to evolve the congestion window. Besides, packet pacing is employed at the end hosts to smooth bursts induced by hardware offload mechanisms like LSO. Through maintaining near-zero queue length, HULL achieves low latency. However, it sacrifices some bandwidth.SDN employed in datacenters is attracting an increasing attention of academics, and multiple approaches are put forward, like Hedera [10], MicroTE [15], and Planck [37]. However, these methods use centralized traffic engineering to reroute traffic based on network conditions. The communication between switches and controllers causes an overhead that is quite considerable comparing with the completion time of short flows. Hence, the distributed approaches we discuss in this paper are more practical and are ready to deploy.

3. Motivation

3.1. Quite Low Latency Requirement

To achieve horizontal scalability, scale-out approaches are widely used in today's datacenters. Scale-out means that multiple servers share the computing load of an application. A typical example is the web search applications. The task of responding each user's query is portioned into small tasks, which are assigned to different workers. All the responses from the workers are aggregated to generate the final result. This pattern is named as Partition/Aggregation [13, 42, 48].

The parallel work reduces the computing time at the servers since the search task assigned to each server is smaller. However, the communication between servers becomes more frequent. To guarantee that the response can be generated within a specific time, the communication time between two servers has to be very small. Currently, the response time of a user query is required to be not larger than 200-300 milliseconds. If the query is partitioned twice, then possibly some flows have to be finished in less than ten milliseconds. For example, Figure 1 shows a partition/aggregation workflow pattern. The deadline of one user query is 300 milliseconds. Each worker needs 40ms to finish the sub-task and each aggregator needs 100ms to aggregate the responses from its children. The time of transmitting a response message is twice as much as that of transmitting a query message. Then a query flow deadline is only 10 milliseconds.

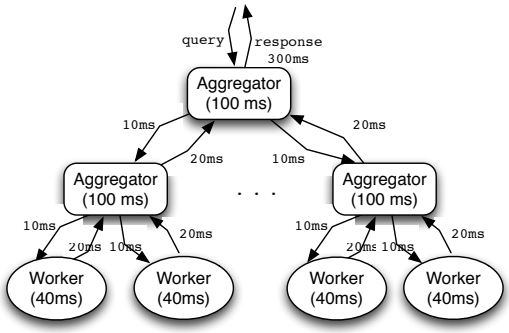

Fig. 1. A Partition/Aggregation workflow pattern. A user query task with 300 ms deadline is completed by multiple workers.

3.2. Cost of Latency is High

Cloud service providers aim to provide online services to users. The response time largely impacts the users' experiences. Large latency brings great negative impact on the profits of the service providers. For example, Google researchers run an experiment where the number of search results is increased from 10 to 30 one page. The latency correspondingly increased from 0.4 seconds to 0.9 seconds. As a result, the half a second delay caused a 20% drop in traffic [1]. Amazon found every 100ms of latency cost them 1% in sales [4]. If a broker's electronic trading platform is 5 milliseconds behind the completion, the broker could lose at least 1% of its trade, which is $4 million in revenue per millisecond. Up to 10 milliseconds of latency could result in a 10% drop in revenue [3]. Therefore, in order to provide better cloud services and thus win more users, reducing latency is quite critical.

3.3. Varied Latency Requirements

Cloud users have different expectations for the service delay. For example, users expect web search engine could return the search results instantly, while they can accept larger latency of an email service or a cloud storage service. As the cloud computing technology develops, more and more cloud services will come up. Cloud service providers need to guarantee different deadlines for them according to the service function and the system capability. Therefore, the latency requirements at the service level are different.

Besides, even if all the services have the same latency requirement, the flows still possibly have varied deadlines. Taking the web search as an example, each worker takes charge of different parts of keys, possibly runs different search algorithms, and returns different number of search items. All of these may lead to different flow deadlines [42,45].

Therefore, flows in datacenters could have varied deadlines. Using a limited number of priorities to provide differentiated services is not a scalable method.

3.4. Quite Short Latency-Sensitive Flows

Query traffic in datacenters follows the partition/aggregation pattern. The query flows are generally quite short and have strict latency requirements. The size of the query flows

is generally 1.6 to 2KB [13]. The latency of an online application is generally 200-300 milliseconds. Since the applications usually need to be done across thousands of servers based on the partition/aggregation pattern, the latency of one flow will be decreased to several or dozens of milliseconds as shown in subsection III-A.

D2TCP works in one RTT interval. After receiving unmarked ACKs, it works as TCP. After receiving marked ACKs, the flows with different deadlines adjust their congestion windows based on their remaining time and their current congestion window. This method works well in long-term since the near-deadline flows will get more bandwidth than the far-deadline flows. However, it is possible to be ineffective for the latency-critical short flows in datacenters.

For example, the maximum TCP packet length is 1,460KB. Then a query flow only consists of 2 packets. The slow start window of today's TCP generally equals 2 if delayed ACK is employed. Therefore, a query flow can finish using only one RTT. Even if D2TCP is used, it does not take effect for the quite short flows.

To further validate the above thought, we implemented D2TCP on the ns2 platform based on the DCTCP ns2 code [2].We compared the performance of DCTCP and D2TCP in a simple scenario to validate whether D2TCP works well for the short latency-sensitive flows. Six servers connect with a switch. The buffer size of each port is 128KB and the link capacity is 1Gbps. Four servers established four long flows at time 1.0 second to make the network full of packets. The flow size was 100MB. At 1.5 second, a short flow with 0.2 millisecond deadline was established between the other sender and the receiver. The flow size equals the size of queries [13], that is, 2KB. The completion time of flows DCTCP and D2TCP is shown in Figure 2. We can see that the performance of D2TCP is quite close to DCTCP. To validate D2TCP takes effects for long latency-sensitive flows, we did the previous simulation again. The difference is that at 1.5 second, a flow with 10MB length and 200 millisecond deadline was established. Figure 3 shows the flow completion time of the five flows. We can see that D2TCP indeed makes the latency-sensitive flows finish quicker than DCTCP and meets the deadline. This is because the

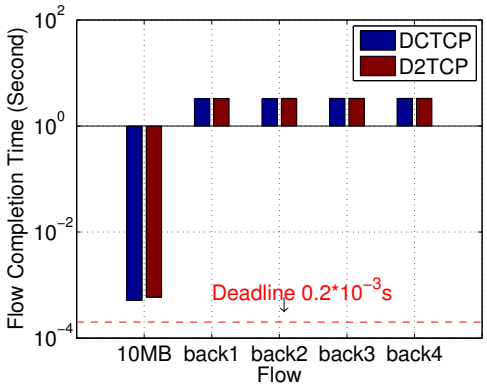

Fig. 2. Flow completion time of the five flows. D2TCP does not take effect for the short latency-critical flows.

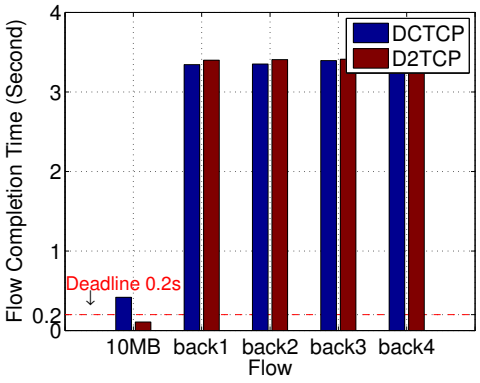

Fig. 3. Flow completion time of the five flows. In D2TCP, the latency-sensitive flow meets its deadline while DCTCP can not guarantee the deadline.

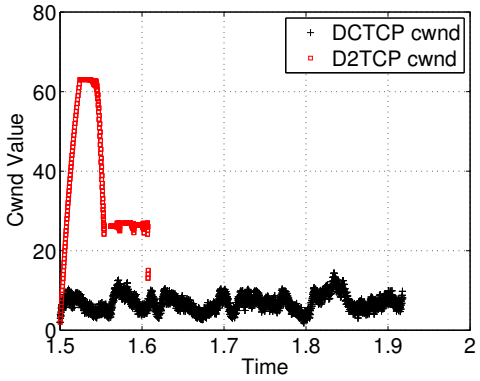

Fig. 4. Cwnd variation of the delay-sensitve flow. In D2TCP, it can get larger cwnd.

large delay-sensitive flow needs a relatively long time to finish. The congestion window of it in D2TCP is larger than that in DCTCP. Therefore, it can meet the deadline.

Figure 4 confirms the above conjecture. The variation of congestion window in DCTCP and D2TCP of the deadline flow is depicted. We can see that the window continually increases in D2TCP at first. This is because the expected deadline is shorter than the computed deadline during the first half time. Thus, it will be scheduled first. The sender can quickly receive ACKs the sender of the delay-sensitive flow decreases the congestion window by a quite small number even if it receives marked ACKs. After about 1.53 seconds, the congestion window is already much large. The expected deadline becomes larger than the computed deadline. Therefore, the congestion window begins to decrease faster. At about 1.61 seconds, the flow finishes in D2TCP, while the flow finishes at about 1.92 seconds in DCTCP.

We can see that rate-based mechanisms at the end hosts response too slowly for the quite short latency-critical flows. Mechanisms at the switch can response more quickly. Therefore, we designed a mechanism, which works at the switches to achieve low latency and provide unlimited differentiated services.

4. Details of pECN

In this section, we will describe the details of the proposed mechanism, pECN (prioritized ECN). pECN aims to reduce flow latency and provide unlimited number of differentiated services. The basic idea of pECN is to employ ECN technology to maintain a short queue length and utilize a prioritized scheduling mechanism to provide differentiated services for different flows. pECN can quickly respond to the nearest-deadline flow and adapt to varied deadline requirements.

4.1. Marking Policy

The ECN mechanism [6] will mark the arrived packets by setting the Congestion Experienced (CE) code point when the queue length exceeds a limited value, and the receivers will send the marked information back to the senders by ACKs. After receiving marked ACKs, the senders will reduce congestion window according to a congestion control algorithm. The marking policy includes two parts. One is determining the marking threshold. The other one is determining which packet should be marked. The marking threshold K is given in [13] as

$$K > \frac{C \times RTT}{7} \tag{1}$$

Note that it is a quite small number in datacenters. For example, if the link capacity is 10Gbps and RTT is 200 microseconds, then $\frac{C \times RTT}{7}$ is only about 35 KBytes.

In terms of the selection of the marked packets, generally the just arrived packet will be marked. However, if some packets of a near-deadline flow are unfortunately marked, then the sending rate of the flow will be reduced, which may increase the flow completion time. It is better to mark a packet of far-deadline flows. Therefore, there is another kind of marking policy. When a packet arrives after the queue length exceeds the threshold, the packet with the lowest priority will be found and marked. We named this marking policy as priority-based marking (pMark). This method will incur some search overhead.

In the evaluation part, both of the two marking policies will be evaluated. We found that the priority-based marking policy does not play an important role in reducing the latency. The main reason is that even if the packet with the highest priority is marked, it will be scheduled first. The sender can quickly receive ACKs and send new packets.

4.2. Priority-Based Scheduling

Priority queue is widely used to provide differentiated services, such as DiffServ [8] in Internet and PFC in datacenters. However, this kind of method can only provide limited number of services. It fails to provide differentiated services for more various flows.

If more priority queues can be maintained, then more different kinds of services can be provided. At the worst case, if each packet belongs to a kind of service, then the number

of priority queues required is same to the number of the packets. In traditional Internet, the number of packets at a switch buffer is quite large. It is impractical to provide the priority-based scheduling at the packet level.

However, since datacenters have quite short RTT, the queue length could be quite short as well if an AQM mechanism is used to limit the queue length. Therefore, it is possible to implement priority-based scheduling at the packet level. When a packet needs to be transmitted, the packet with the highest priority will be found and then sent out.

The priority values are set at the end hosts. They can be determined according to the flow deadline. Then prioritized scheduling becomes Early Deadline First. The priority can also be positively proportional to the flow size. Then the prioritized scheduling acts as Shortest Job First scheduling mechanism.

4.3. Congestion Control Mechanism

Since the CE information is a single bit in the ECN mechanism, the congestion control mechanism of standard TCP is that the sender cuts the congestion window by half after receiving a CE mark. In DCTCP, a multi-bits feedback is derived from the single bit mark. And the congestion control mechanism at the sender side is modified to cutting the congestion window in proportion to the fraction of the marked packets.

In D2TCP, the congestion window is reduced based on not only the fraction of the marked packets, but also the remaining time, the remaining bytes and the current congestion window of the flows. Near-deadline flows cut fewer congestion windows than far-deadline flows.

In pECN, we use the congestion control mechanism in DCTCP. On one hand, the rate control at the end hosts plays little role in reducing the latency of quote short flows. On the other hand, once the prioritized scheduling mechanism is used, the rate of the flows with higher priority will increase faster than the other flows since the packets of the flows with higher priority encounter shorter round trip time. Faster ACK feedback will lead to faster sending rate.

4.4. Benefits of pECN

Providing unlimited differentiated services. The traditional priority-based scheduling mechanisms based on the priority queues with limited number, generally 8, could only support at most 8 kinds of different services. While in pECN, when a switch's output port could send out a packet, it will select the packet with the highest priority value to transmit, that is, no matter how many kinds of packets with different priority values exist, pECN will always transmit packets in the order of their priority values. Thus, pECN could support unlimited differentiated services.

No need to maintain any flow state. When making forwarding decisions, pECN could only need to check the priority values of the accommodated packets in the buffer, that is, it does not need to rely on any historical flow information. Therefore, pECN does not need to maintain any flow state at the switches. This is a quite important benefit in datacenter networks. Since most of flows in datacenter networks have quite small flow size, they only last for a quite short time. A mechanism that requires to record flow information will incur substantial overhead.

Simple Implementation. Since pECN does not need to maintain historial flow information, the implementation of it is simple. As for the pMark function, we will evaluate its effectiveness in the simulation part. The results show that it impacts a little on the performance of pECN. Thus, the pMark function could be ignored in real implementation of pECN. If the pMark policy is not employed, only the dequeue function at the switches is required to be modified.

5. Evaluation

We will implement pECN on the ns2 platform and evaluate its performance in this section. The topology is shown in Figure 5. There are totally 18 ToR switches. Each rack has 20 servers. The capacity of the link between each server and its ToR is 1Gbps, and the link capacity between each ToR and the core switch is 10Gbps. Therefore, the oversubscription is 2. The propagation delay of each link is 30 microseconds. The buffer size of each port is 256KB.

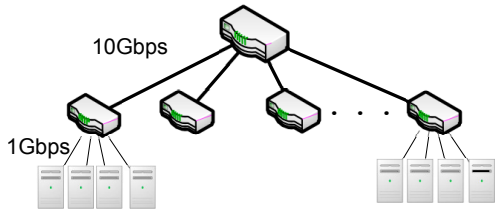

Fig. 5. Simulation topology. Each top of rack contains 20 servers.

The performance of pECN is compared with TCP, DCTCP [13], D2TCP [42], and pECN with pMark. TCP uses Drop-Tail queue management policy, while DCTCP, D2TCP and pECN use RED. The threshold of marking is 30Kbytes. Some important parameters on the ns2 platform need to be modified. `Window_`, which controls the advertised window at the receiver side, should be increased from 20 to a large enough value so that the receiver will not become the bottleneck. In our simulations, we set it to 1,000. Delayed ACK is disabled. In TCP, the timer of the delayed ACK is 40 milliseconds. In our simulations, we found that if delayed ACK is enabled, the latency of the flows with odd packets will be quite large. We can either disable delayed ACK or reduce the timer duration of the delayed ACK. Considering that a query flow will possibly be finished in a quite short time, it is difficult to set a proper delayed ACK value for all the flows. Therefore, in our implementation, the delayed ACK is disabled. The timeout is 200ms as suggested in TCP.

Two kinds of traffic patterns are generated to test the performance of the protocols.

A typical kind of traffic pattern in data- centers is multiple to one burst, that is, multiple senders transmit packets to the same receiver at the same time. This kind of traffic widely exists in datacenters, such as in the Data- Intensive Scalable Computing (DISC) systems [16], including MapReduce [19], Dryad [26], Spark [47], CIEL [34], TritonSort [38], in the partition/aggregation workflow. Since this kind of traffic is quite popular

in today's datacenters, the designed protocol should perform well with it. In terms of the performance metrics, we mainly use flow completion time, which is a widely used metric to compare the transport protocols in datacenters [13,25,42,45], 99th tail flow completion time, and the number of senders that can be supported without compromising deadlines.

In this section, we will investigate the performance of the mechanisms with burst traffic. This kind of traffic possibly exists in two scenarios. One is intra-rack burst traffic. The other is inter-rack burst traffic. Application designers would like most of the communication between servers to happen inside a rack, which is called locality. Locality reduces the traffic across different subdomains. The bandwidth pressure at the top layer of the datacenter topology can be alleviated. Therefore, intra-rack burst traffic is more desirable than inter-rack burst traffic. However, if a task is so big that it requires hundreds of servers to cooperate, then inter-rack burst traffic is also needed. Hence, we will investigate the performance of pECN with both of these two kinds of scenarios.

We will firstly investigate the performance of the mechanisms in the scenario of burst traffic without background flows. Then some background traffic is added to study whether pECN works well in the scenario of burst traffic with background flows.

5.1. Without Background Traffic

One server in a rack acts as the receiver, and the others are the senders. Since there are totally 20 servers in a rack, to emulate multiple servers, we let one sender generate multiple flows when the number of connections is larger than 19. The flow size is uniformly distributed across [2KB, 50KB]. The flow priority is positively proportional to the flow size. At 0.1 second, all the senders establish connections with the receiver.

Flow Completion Time. Figure 6 depicts the average flow completion time of all the flows with different number of connections. We can see that TCP performs the worst. Through investigating the log in detail, we found that TCP flows suffer many timeouts and thus the flow completion time is quite large. D2TCP performs almost the same as DCTCP except that the number of connections is quite large. This is because all the flows are very short. D2TCP can work well only for the relatively long flows. pECN performs the best. It reduces the average flow completion time by 24.5% compared with DCTCP when the number of connections equals 80. This is because pECN schedules the flow with the highest priority first. In this scenario, the priority is positively proportional to the flow size. Therefore, it actually achieves the Shortest Job First (SJF) scheduling mechanism, which minimizes the average flow completion time. The performance of pECN with pMark is almost the same as pECN, which indicates that pMark does not play a role.

To verify our conjecture about why pECN performs better than the other protocols, we plot the relationship between the flow size and the flow completion time with pECN and DCTCP mechanisms as the number of connections is 80 in Figure 7. In pECN, except from one occasional point, the smaller the flow size is, the shorter the flow completion time will be, that is, it indeed acts as a Shortest Job First scheduling mechanism. While in DCTCP, about 11 of the 80 flows finished earlier than some flows with shorter length. Thus, the average flow completion time of pECN is shorter than DCTCP. D2TCP has almost the same result with DCTCP. To make the figure clearer, we did not plot the result of D2TCP.

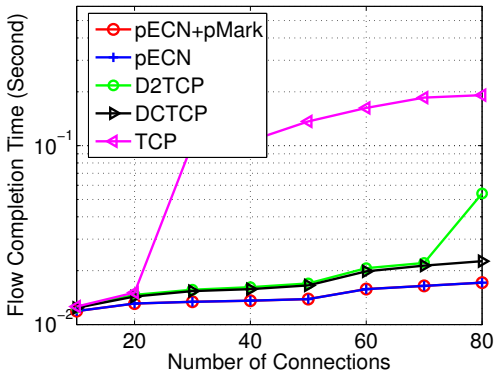

Fig. 6. Intra-rack: Average flow completion time with different number of senders. The flow size uniformly distributes across [2KB, 50KB].

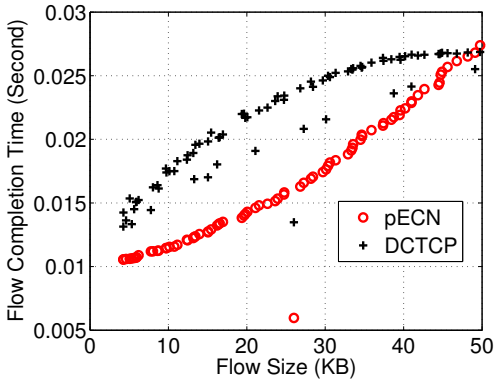

Fig. 7. Intra-rack: Average flow completion time with different flow sizes. The number of connections is 80.

It's likely that the burst traffic pattern causes congestion at the bottleneck switch buffer. To alleviate the performance degradation in the online data-intensive applications, application designers tend to use quite small responses [13]. To simulate this kind of traffic, we let the flow size be uniformly distributed across [1.6KB, 2KB] [13] and repeat the above simulations.

Figure 8 depicts the average flow completion time with different number of connections in a rack. The curves are quite different from that in Figure 6. The performances of TCP, DCTCP, and D2TCP are almost the same. This is because the flow size is too small. The congestion control mechanisms at the end hosts did not take effect. However, pECN and pECN with pMark perform better. This is because pECN acts as an SJF scheduling mechanism and it achieves the minimum average flow completion time.

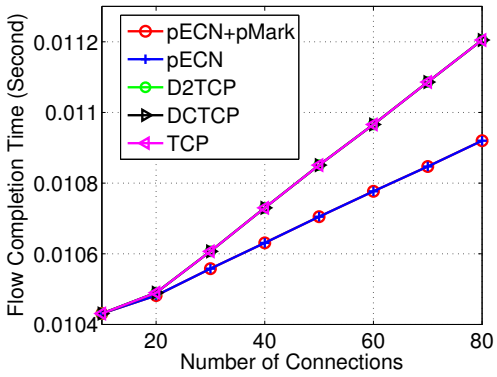

Fig. 8. Intra-rack: Average flow completion time with different number of senders. The flow size uniformly distributes across [1.6KB, 2KB]

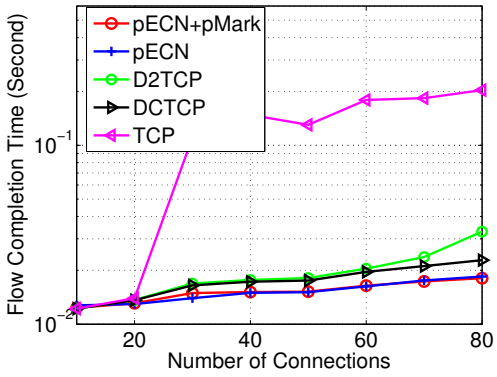

Fig. 9. Inter-rack: Average flow completion time with different number of senders. The flow size uniformly distributes across [2KB, 50KB].

As stated at the beginning of this section, except the intra-rack burst traffic, there is also inter-rack burst traffic. To investigate the performance of pECN in this kind of scenario, we selected a server that did not connect to the same ToR switch as the senders and the receiver and conducted the above simulations again.

Figure 9 shows the average flow completion time of the flows with different number of senders with inter-rack burst traffic. The results with all the mechanisms increase a little since the round trip propagation delay increases from 120 microseconds to 240 microseconds. However, relatively, the result is quite close to that with intra-rack burst traffic. pECN still has the best performance, and TCP works the worst. The results with smaller flow size are similar to Figure 8.

Missed Deadline. Figure 10 shows the number of connections that can be supported by different mechanisms without compromising the deadline. We used intra-rack burst traffic. The flow size is uniformly distributed across [2KB, 50KB]. The flow deadline is uniformly distributed across [10ms, 20ms] or [20ms, 30ms]. TCP supports the least number of connections. When the flow deadline takes a value between [10ms, 20ms], D2TCP, pECN and pECN with pMark can support the same number of connections. When the flow deadline is uniformly distributed across [20ms, 30ms], pECN with pMark can support the largest number of connections, 57, and pECN supports 54 connections, while DCTCP and D2TCP support fewer connections.

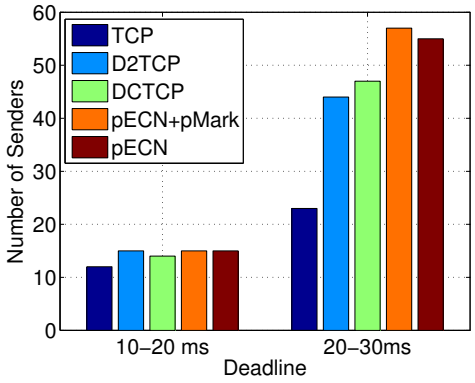

Fig. 10. The number of senders that can be supported without compromising deadlines.

5.2. With Background Traffic

To evaluate the effect of background traffic, we added two long flows as background in the intra-rack scenario. The flow size is uniformly distributed across [2KB, 50KB]. The background flows started at 0.1 second. The deadline was quite large. At 1.5 second, the queries started. Figure 11 shows the average flow completion time of the short flows. Compared with Figure 6, the average flow completion time of TCP increases a lot because of the large queue length at the bottleneck switch buffer. The other three mechanisms do not increase too much. This is because the number of short flows is much larger than the number of background flows. Therefore, the bandwidth taken by the background flows is not large. Besides, at the switches, the queue length is controlled by the ECN mechanism. Therefore, the queuing latency also does not increase too much.

The results with background traffic flows in the scenario of inter-rack burst traffic is also similar. To avoid redundancy, the figures are omitted.

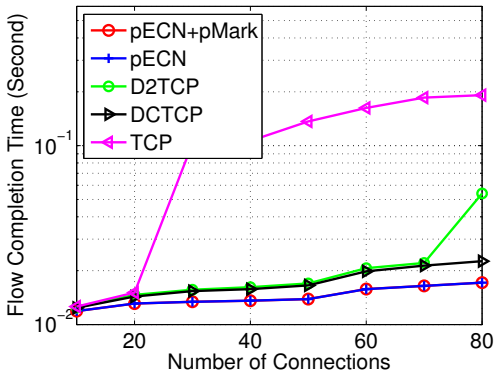

Fig. 11. Average completion time of short flows with different number of senders in the scenario of intra-rack burst traffic with background flows. The flow size uniformly distributes across [2KB, 50KB].

6. Conclusion

As datacenter applications increase, reducing flow latency is attracting more and more attention. Datacenter has special traffic characteristics. The latency-sensitive flows are usually quite short. Reducing latency through controlling rate at the end hosts responses too slowly for the short flows. In this paper, pECN is proposed to reduce latency for the delay-sensitive flows. pECN uses ECN mechanism to limit the queue length and employs the priority-based scheduling mechanism to provide differentiated services. The priority-based scheduling works at the packet level. When transmitting, the packet with the highest priority will be sent. The overhead of the scheduling mechanism is not large since the number of packets in the queue is limited within a small value by the ECN mechanism. Also, the scheduling mechanism can provide unlimited number of differentiated services without limitation by the number of priority queues. Besides, pECN does not need to maintain any flow state at the switches.

We implemented the proposed mechanism pECN on the ns-2 platform. To investigate whether priority-based marking benefits the flows with higher priority, we also implemented pECN with pMark. Burst traffic was generated to evaluate the performance of the proposed mechanism. The performances of pECN and pECN+pMark are compared with TCP, DCTCP and D2TCP in terms of the flow completion time and the number of flows that can be supported without compromising deadline. The results show that pECN has similar performance to pECN+pMark. It only works a little worse in some scenarios. And pECN has better performance for the delay-sensitive flows compared with TCP, DCTCP and D2TCP.

Acknowledgement. The work described in this paper was fully supported under the National High Technology Research and Development Program (863) of China (Project Number: 2015AA011906), and the National Natural Science Funds (Project Number: 61502049,61300184, 61302089), the National Basic Research Program (973) of China (Project Number: 2012CB315801).

References

1. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html
2. Data Center TCP. http://www.stanford.edu/ alizade/Site/DCTCP.html
3. The Value of a Millisecond: Finding The Optimal Speed of a Trading Infrastructure. http://www.tabbgroup.com/PublicationDetail.aspx?PublicationID=346 (2008)
4. Amazon found every 100ms of latency cost them 1% in sales. http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/ (Auguest, 2008)
5. Facts and Stats of Worlds largest data centers. In: https://storageservers.wordpress.com/2013/07/17/facts-and-stats-of-worlds-largest-data-centers/ (July 2013)
6. The Addition of Explicit Congestion Notification (ECN) to IP. http://www.ietf.org/rfc/rfc3168.txt (Seq 2001)
7. Abts, D., Felderman, B.: A Guided Tour through Data-center Networking. ACM Queue 10(5), 10 (2012)
8. et al., S.B.: An Architecture for Differentiated Services. In: IETF RFC 2475 (Dec 1998)
9. Al-Fares, M., Loukissas, A., Vahdat, A.: A Scalable, Commodity Data Center Network Architecture. In: ACM SIGCOMM. pp. 63–74 (2008)
10. Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A.: Hedera: Dynamic flow scheduling for data center networks. In: NSDI. vol. 10, pp. 19–19 (2010)
11. Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., Yasuda, M.: Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center. In: USENIX NSDI (2012)
12. Alizadeh, M., Yang, S., Katti, S., McKeown, N., Prabhakar, B., Shenker, S.: Deconstructing Datacenter Packet Transport. In: ACM Workshop on HotNet. pp. 133–138 (2012)
13. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J.: Data Center TCP (DCTCP) . In: ACM SIGCOMM. pp. 63–74 (2010)
14. Benson, T., Anand, A., Akella, A., Zhang, M.: Understanding Data Center Traffic Characteristics. ACM SIGCOMM Computer Communication Review 40(1), 92–99 (2010)
15. Benson, T., Anand, A., Akella, A., Zhang, M.: Microte: Fine grained traffic engineering for data centers. In: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies. p. 8. ACM (2011)
16. Bryant, R.E.: Data-Intensive Scalable Computing: Harnessing the Power of Cloud Computing (2009)
17. Chen, Y., Griffith, R., Liu, J., Katz, R., Joseph, A.: Understanding TCP Incast Throughput Collapse in Datacenter Networks. In: ACM workshop on Research on Enterprise Networking. pp. 73–82 (2009)
18. Cui, Y., Wang, H., Cheng, X.: Channel Allocation in Wireless Data Center Networks. In: IEEE INFOCOM. pp. 1395–1403. IEEE (2011)
19. Dean, J., Ghemawat, S., Inc, G.: MapReduce: Simplified Data Processing on Large Clusters. In: USENIX OSDI (2004)
20. Feng, W., Shin, K., Kandlur, D., Saha, D.: The BLUE Active Queue Management Algorithms. IEEE/ACM Transactions on Networking (TON) 10(4), 513–528 (2002)
21. Floyd, S., Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking 1(4), 397–413 (1993)
22. Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: VL2: a Scalable and Flexible Data Center Network. In: ACM SIGCOMM. ACM (2009)
23. Halperin, D., Kandula, S., Padhye, J., Bahl, P., Wetherall, D.: Augmenting Data Center Networks with Multi-gigabit Wireless Links. In: ACM SIGCOMM. pp. 38–49. ACM (2011)

24. Hamedazimi, N., Qazi, Z., Gupta, H., Sekar, V., Das, S.R., Longtin, J.P., Shah, H., Tanwer, A.: FireFly: a Reconfigurable Wireless Data Center Fabric using Free-Space Optics. In: ACM SIGCOMM. pp. 319–330. ACM (2014)
25. Hong, C., Caesar, M., Godfrey, P.: Finishing Flows Quickly with Preemptive Scheduling. In: ACM SIGCOMM (2012)
26. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In: EuroSys. pp. 59–72 (2007)
27. Kamra, A., Kapila, S., Khurana, V., Yadav, V., Shorey, R., Saran, H., Juneja, S.: SFED: A Rate Control Based Active Queue Management Discipline. IBM India Research Laboratory Research Report (2000)
28. Katabi, D., Handley, M., Rohrs, C.: Congestion Control for High Bandwidth-delay Product Networks. In: ACM SIGCOMM (2002)
29. Kung, N., Morris, R.: Credit-based Flow Control for ATM Networks. IEEE Network 9(2), 40–48 (1995)
30. Kunniyur, S., Srikant, R.: Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In: ACM SIGCOMM Computer Communication Review. vol. 31, pp. 123–134 (2001)
31. Kunniyur, S., Srikant, R.: An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. IEEE/ACM Transactions on Networking 12(2), 286–299 (2004)
32. Lin, D., Morris, R.: Dynamics of Random Early Detection. In: ACM SIGCOMM Computer Communication Review. vol. 27, pp. 127–137 (1997)
33. Low, S., Paganini, F., Wang, J., Adlakha, S., Doyle, J.: Dynamics of TCP/RED and a Scalable Control. In: IEEE INFOCOM. pp. 239–248 (2002)
34. Murray, D.G., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., Hand, S.: CIEL: A Universal Execution Engine for Distributed Data-Flow Computing. In: USENIX NSDI (2011)
35. Ott, T., Lakshman, T., Wong, L.: SRED: Stabilized Red. In: IEEE INFOCOM. pp. 1346–1355 (1999)
36. Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., et al.: The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. ACM SIGOPS Operating Systems Review 43(4), 92–105 (2010)
37. Rasley, J., Stephens, B., Dixon, C., Rozner, E., Felter, W., Agarwal, K., Carter, J., Fonseca, R.: Planck: Millisecond-scale monitoring and control for commodity networks. In: ACM SIGCOMM Computer Communication Review. vol. 44, pp. 407–418. ACM (2014)
38. Rasmussen, A., Porter, G., Conley, M., Madhyastha, H., Mysore, R., Pucher, A., Vahdat, A.: Tritonsort: A Balanced Large-scale Sorting System. In: USENIX NSDI (2011)
39. Rumble, S., Ongaro, D., Stutsman, R., Rosenblum, M., Ousterhout, J.: Its Time for Low Latency. In: USENIX Workshop on Hot Topics in Operating Systems (HOTOS) (2011)
40. Shin, J.Y., Sirer, E.G., Weatherspoon, H., Kirovski, D.: On the Feasibility of Completely Wirelesss Datacenters. Networking, IEEE/ACM Transactions on 21(5), 1666–1679 (2013)
41. Staff, C.: BufferBloat: What's Wrong with The Internet? Communications of the ACM 55(2), 40–47 (2012)
42. Vamanan, B., Hasan, J., Vijaykumar, T.: Deadline-Aware Datacenter TCP (D2TCP). In: ACM SIGCOMM (2012)
43. Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D.G., Ganger, G.R., Gibson1, G.A., Mueller, B.: Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In: ACM SIGCOMM. pp. 303–314 (Aug2009)
44. Wang, C., Li, B., Thomas Hou, Y., Sohraby, K., Lin, Y.: LRED: A Robust Active Queue Management Scheme Based on Packet Loss Ratio. In: IEEE INFOCOM (2004)
45. Wilson, C., Karagiannis, T.: Better Never than Late : Meeting Deadlines in Datacenter Networks. In: ACM SIGCOMM. pp. 50–61 (2011)

46. Wu, H., Feng, Z., Guo, C., Zhang, Y.: ICTCP: Incast Congestion Control for TCP in Data Center Networks. In: ACM CoNEXT (2010)
47. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster Computing with Working Sets. In: USENIX HotCloud (2010)
48. Zats, D., Das, T., Mohan, P., Borthakur, D., Katz, R.: DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In: ACM SIGCOMM (2012)

Tao Huang received his B.S degree in communication engineering from Nankai University, Tianjin, China, in 2002, the M.S. and Ph.D. degree in communication and information system from Beijing University of Posts and Telecommunications, Beijing, China, in 2004 and 2007 respectively. He is currently the associate professor in Beijing University of Posts and Telecommunications. His current research interests include network architecture, Routing and Forwarding, network virtualization.

Jiao Zhang is currently an assistant professor in the School of Information and Communication Engineering at Beijing University of Posts and Telecommunications (BUPT), Beijing, China. In July 2014, she received her Ph.D degree from the Department of Computer Science and Technology, Tsinghua University, China. Her supervisor is Prof. Fengyuan Ren. From August 2012 to August 2013, she was a visiting student in the networking group of ICSI, UC Berkeley. In July 2008, she obtained her Bachelors Degree from the School of Computer Science and Technology from BUPT. Her research interests include traffic management in data center networks, routing in wireless sensor networks and future Internet architecture. Until now, she has (co)-authored more than 20 international journal and conference papers.

Yunjie Liu received his B.S degree in technical physics from Peking University, beijing, China, in 1968. He is currently the academician of China Academy of Engineering, the chief of the science and technology committee of China Unicom, and the dean of the school of information and communication engineering, Beijing University of Posts and Telecommunications. His current research interests include next generation network, network architecture and management.

Received: March 1, 2016; Accepted: May 28, 2016.