P-PFC: Reducing Tail Latency with Predictive PFC in Lossless Data Center Networks

Chen Tian, Bo Li, Liulan Qin, Jiaqi Zheng, Jie Yang, Wei Wang, Guihai Chen, and Wanchun Dou

Abstract—Remote Direct Memory Access(RDMA) technology rapidly changes the landscape of nowadays datacenter applications. Congestion control for RDMA networking is a critical challenge. As an end-to-end layer 3 congestion control mechanism, Datacenter QCN (DCQCN) alleviates the unfairness and head-of-the-line blocking problems of Priority-based Flow Control (PFC). However, a lossless network does not guarantee low latency even with DCQCN enabled. When network congestion happens, switch queues still build-up due to the response latency of end-to-end solutions. In this article, we propose Predictive PFC (P-PFC) to reduce tail latency in RDMA networks. P-PFC monitors the derivative of buffer occupation, predicts the happening of PFC trigger in the future, and proactively triggers PFC pause in advance. The benefit is that buffer usage can be maintained at a low level, hence the tail latency can be controlled. Preliminary evaluation results demonstrate that P-PFC can reduce tail latency by more than half of that in standard PFC in many scenarios, without hurting the throughput and average latency. P-PFC can also protect innocent flows compared with standard PFC according to our experiments. To our best knowledge, this is the first work of using derivative to improve PFC in lossless RDMA networks.

Index Terms—Data center networks, congestion control, PFC, RDMA

1 Introduction

Remote Direct Memory Access (RDMA) technology rapidly changes the landscape of nowadays datacenter applications [1], [2], [3], [4], [5], [6]. By implementing the protocol stack on the host NIC(s), RDMA provides lower latency and higher per-connection throughput with nearly zero CPU consumption.

Congestion control is a critical challenge for RDMA networking. RDMA demands a lossless network, where there is no packet loss due to buffer overflow at the switches. RDMA over Converged Ethernet v2 (RoCEv2) [7], [8] standard supports RDMA over the converged enhanced Ethernet and IP networks. To prevent buffer overflow, its Priority-based Flow Control (PFC) mechanism can pause the upstream Ethernet port (or a particular priority class) when buffer occupancy reaches a specified threshold. By inverting the congestion level by level to the upstream node until the network terminal devices, PFC shifts the congestion from the network center to the edges. As a coarse-grained, hop-by-hop layer 2 congestion control mechanism, PFC has fundamental problems such as unfairness and head-of-the-line blocking [9], [10], [11], [12]. To solve these two problems, [Data Center Quantized Congestion Notification (DCQCN)] uses an end-to-end layer 3 approach which can perform congestion control and suppress the PFC mechanism [10]. DCQCN requires standard [Random Early Detection (RED)] [13] and [Explicit Congestion Notification (ECN)] [14] support from switches to encode congestion information into packets. At the receiver, it uses the existing Congestion Notification Packet (CNP) mechanism [15] defined in RoCEv2 to notify the senders. A sender uses explicit rate control, and decreases/increases an individual flow's rate based on the received CNP packets.

A key issue for end-to-end congestion control schemes is that the convergence process is not fast enough in certain scenarios such as incast. It leads to high buffer occupation during the converging process. The high buffer occupation further leads to increased latency, which is critical for RPC-like (Remote Procedure Call) data center applications [11]. Although RDMA technology can eliminate the large latency with bypassing the host network stack, the large latency incurred by the long queue (up to milliseconds) in switches will also hurt latency-sensetive applications. Because the FCT (Flow Completation Time) of these applications can be only several microseconds. Note that increased latency increases the control loop of end-to-end schemes which can further slow down the convergence.

To illustrate the buffer overflow problem, we perform simulations of incast [16] scenarios on an NS3 RDMA simulator [10]. Each host is connected to a 40 Gbps PFC-enabled switch with link propagation latency of 1 μs. The size of the shared buffer in the switch is 4 MB (marked by the black dotted lines in Fig. 1). We mark the queue length as red lines and the time points when senders receive a CNP packet as green short vertical lines. First, we demonstrate a 16:1 incast scenario in Fig. 1a. Please notice that in RDMA environment, each host sends packets at line rate (at 40 Gbps) rather than window-based. While DCQCN is enabled and functioning correctly in

• *The authors are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210008, China. E-mail: alexandretian@gmail.com, {245939069, liulan_q, 467287824}@qq.com, {jzheng, ww, gchen, douwc}@nju.edu.cn.*

Manuscript received 7 Oct. 2019; revised 29 Dec. 2019; accepted 21 Jan. 2020. Date of publication 23 Jan. 2020; date of current version 11 Feb. 2020. (Corresponding author: Jiaqi Zheng.)
Recommended for acceptance by J. Zhai.
Digital Object Identifier no. 10.1109/TPDS.2020.2969182

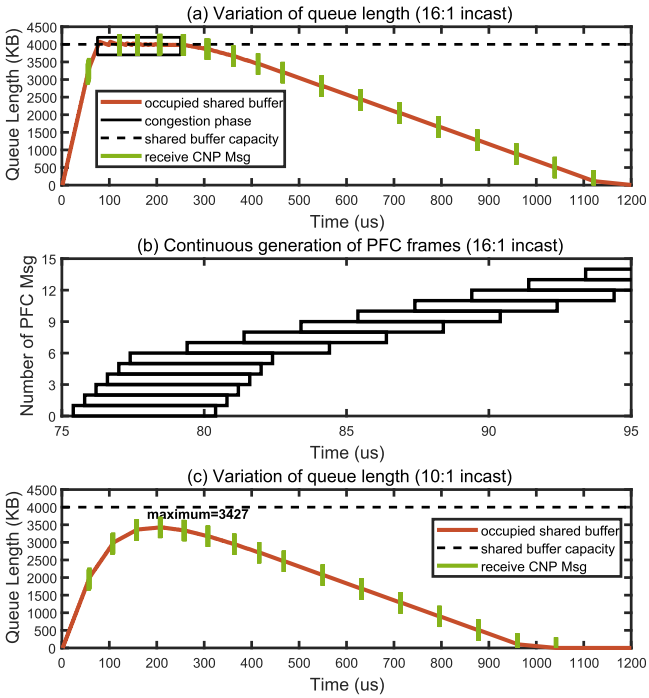


Fig. 1. Latency increases due to (a), (b) 16:1 incast with PFC generation, and (c) 10:1 incast with queue built up.

this scenario, the occupied buffer size quickly grows to the capacity of shared buffer and there is a congestion phase marked as the black square in Fig. 1a. As a result, the PFC mechanism is triggered repeatedly. After several rounds of CNP packets generation in the receiver and the DCQCN congestion control in the senders, the occupied buffer size starts decreasing. Nevertheless, any passing-through flow that shares either the in-port or the out-port with these incast flows would suffer from maximum tail latency of around 830 μs.

To understand the PFC patterns, we draw the PFC frames received by an upstream port during 75 μs and 95 μs in Fig. 1b. Each rectangular represents a received PFC packet with a fixed pause time of 5 μs. We observe that these rectangles overlap with each other which means that an upstream port receives PFC pauses continuously during the congestion phase. The upstream port is thus paused for a long time, which eventually can lead to cascading PFC pauses in upstream switches. We next consider a less severe congestion scenario, a 10:1 incast, in Fig. 1c. In this case, the maximum buffer size is about 3.4 MB, which is below the capacity of shared buffer. Although this case will not trigger the PFC mechanism, the queue length still builds up and the tail latency is 685 μs.

In this paper, we propose Predictive PFC (P-PFC) to reduce tail latency in lossless datacenter networks. Our key observation is: *the switch can predict that congestion will happen by monitoring the derivatives of buffer change.* Unlike the traditional PFC mechanism, P-PFC proactively triggers PFC pause by monitoring the derivative of the buffer occupation. Therefore, P-PFC can quickly react to the ephemeral congestion before the queue length actually builds up. This quick reaction buys time for the more accurate end-to-end schemes, such as DCQCN. The benefit is that buffer usage can be maintained at a low level during the congestion period, hence the tail latency can be controlled. Moreover, P-PFC uses a conservative prediction algorithm so that it has low false alarm rate, *i.e.*, it will not generate a pause message unless the pause message will be generated in a near future by the traditional PFC with high probability. To avoid hurting the throughput and average latency, P-PFC uses carefully designed algorithms to determine the condition for predictive PFC pauses, the upstream ports that should be paused, and the duration of each pause. Preliminary evaluation results demonstrate that P-PFC can reduce tail latency by more than half of that in standard PFC in many scenarios, without hurting the throughput and average latency. According to our experiments, P-PFC can also protect innocent flows compared with standard PFC. Compared with total reduced tail latency, the total pause time caused by P-PFC does look negligible.

The rest of paper is organized as follows. Section 2 introduces the relative background knowledge and explains the some intuitions of P-PFC. Section 3 analyzes the performance of P-PFC compared with standard PFC theoretically. Section 4 provides the detailed designs of algorithms in P-PFC. In Section 5, we evaluate the performance of P-PFC with DCQCN and TIMELY [11] in a variety of scenarios. In Section 6, we give some interesting discussions about P-PFC. Section 7 concludes the paper.

2 Background and Intuition

2.1 PFC Mechanism

PFC is a hop-by-hop layer 2 congestion control mechanism. Downstream switches can send PFC pause packets to upstream switches to control the traffic. Once an upstream port receives a pause message, it should stop data transmission for a given priority class based on the priority and pause duration specified in the pause message. After the pause duration expires (or reception of a new pause message with a pause duration of zero), the upstream port resumes data transmission. According to the standard IEEE 802.3x Pause and PFC frame format [17], each priority has its own independent pause duration, which can be set to a numeric value. PFC is triggered by the usage of the switch ingress buffer, which is actually a counter. All packets are buffered at the central memory. Each packet is counted in both ingress and egress ports. Shared buffer switch divides buffer into two logic parts, *i.e.*, the *reserved buffer* and the *shared buffer*. A reserved buffer is exclusively owned by the ingress queue of a port (or a priority). Take the Mellanox switch SN2700 as an example. It divides shared buffer into several pools and each priority of a port can be mapped to a specific pool. A pool can be configured in the dynamic mode, where it uses a configurable parameter α to define the maximal buffer space for a priority, which is less than $\alpha * free\_pool\_space$. Each ingress port first uses the shared buffer, which is usually controlled by the α value. When the shared buffer is going to be used up, the ingress port begins to use its own reserved buffer. When a small threshold value (*e.g.*, 17 KB) of the port reserved buffer is exceeded, it generates pause to the given upstream port/priority. The rest of the port reserved buffer is used to absorb on-flight packets on the wire before a pause message reaches the upstream port and takes effect.

PFC Drawbacks. [Suppose two flows come in a switch from the same in-port. One flow's out-port is severely

congested. Its packets accumulate then the in-port sends PFC pause frames to block the source port of the link. Now the other flow, although targets a different out-port, is also blocked. This is the Head-Of-Line (HOL) problem caused by PFC. PFC also leads to unfairness problem [10]. When multiple flows from different in-ports cause congestion at an out-port, all ports can be suspended regardless of their fair share of bandwidth.]

2.2 Incast is a Tough Nut in RDMA Network

[Incast is a many-to-one communication pattern [18] that typically occurs in high-performance data centers, especially for distributed storage (*e.g.*, Ceph [19]) and computing applications (*e.g.*, Key-value store [1]). When a client machine sends query requests to a set of servers, the servers may respond at almost the same time. The large amount of traffic generated by these nodes accumulate at the switch's out-port to the client machine. When the egress buffer is full, PFC generation is inevitable.]

Incast control has been studied for many years in TCP networks [20]. The solutions can be generally divided into two categories: window-based solutions [18], [21] and recovery-based solutions [16], [22]. While both of two kind of these solutions have some important drawbacks and it's hard to apply them to RDMA network directly. First, the incast problem only worsens with shallow buffered commodity switches that only provide 100 KB of packet buffer per 10 Gbps port [23] as well as in high-speed network (e.g., 100 Gbps) where switch buffer per port per Gbps decreases as link speeds go up. Second, even if congestion control algorithms like DCQCN and TIMELY estimate each flows fair-share correctly, bursty flow arrival [21], [24] still causes unbounded queue build-up and tail latency, this is deadly enough for many applications like online business, storage, games, videos, etc. According to Fig. 1a, we can find the deluge of packets in high-bandwidth network can fill shallow buffer of switch when the first signal packet like CNP just comes back. PFC is a hop-by-hop congestion control mechanism and it can react fast enough to deal with incast congestion although it has so many notorious problems. What P-PFC wants to do is to fully leverage the quick reaction of PFC and avoid its drawbacks as far as possible at the same time by carefully designed algorithm. We believe P-PFC is a good start to face and improve the traditional PFC mechanism instead of escaping from it.

2.3 Buffer Control With Predictive PFC

A key idea of P-PFC is to send the PFC message in advance that the traditional PFC will eventually send with high probability. To illustrate this and how predictive PFC pauses can help control the buffer size, consider the example shown in Fig. 2a. The switch has three data input ports $P1$, $P2$ and $P3$ attached with sender hosts X, Y and Z respectively. They all forward data packets to output port $P4$ which is attached with a receiver host O. The propagation delay is 1 time unit for all physical links. Suppose that X and Y each sends one unit of data per time unit to O and Z periodically sends packets to O each with a negligible size. Also suppose that O can only receive 1 unit of data per unit time. As the total sending rate of $P1$ and $P2$ is higher

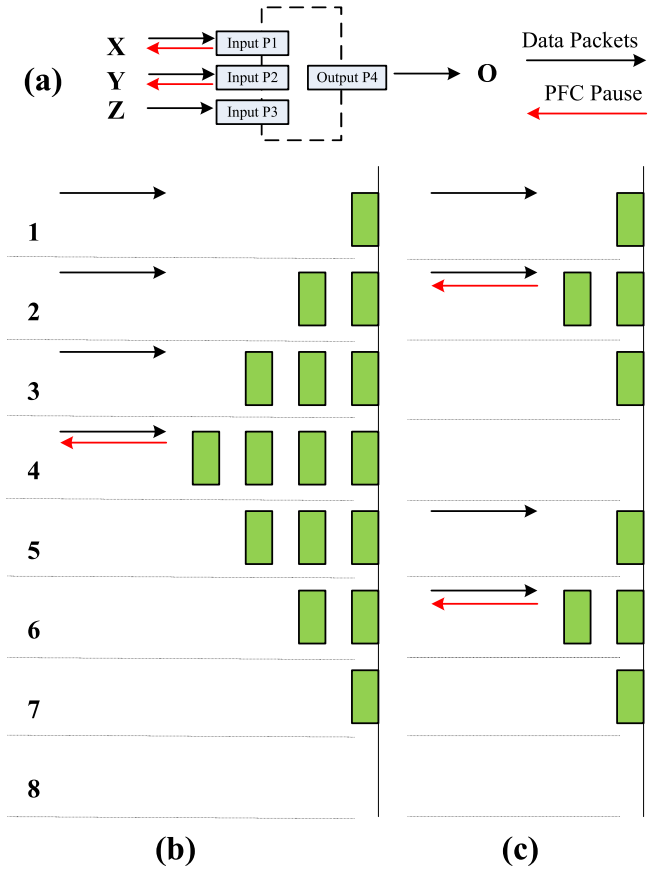


Fig. 2. PFC-based buffer control (a) Topology, (b) Fixed PFC threshold, and (c) Predictive PFC generation

than the receiving rate of $P4$, the queue in the switch builds up with a speed of 1 unit per time unit. Let's assume that shared buffer has a size of 3 data units, reserved buffer is 1 data unit, and the pause duration is 4 time units. [In Figs. 2b and 2c, the green squares represent the accumulated packets in the buffer. The black arrows represent data packets sent from hosts X, Y and Z. The red arrows represent PFC pause frames sent to hosts X, Y and Z.] Therefore, standard PFC mechanism may use up all shared buffer as shown in Fig. 2b. It waits until the start of time slot 4 to send PFC pauses to X and Y respectively. With one time unit for the message to be received by X/Y, the buffer size reaches its peak of 4 data units at the end of time slot 4. While the buffer gradually drains from 4 units to 0 during the following slots, the high buffer occupancy leads to higher latency for packets from Z. From Z's point of view, the average latency is $(1+2+3+4+3+2+1)/8 = 2$ time units, and the tail latency is 4 time units.

Now suppose that we use P-PFC to predict the buffer occupation before sending PFC. As the switch finds that its shared buffer is increasing at a speed of 1 units per time slot, it can predict that the buffer will be used up in the next few slots. Therefore, it can proactively send out PFC pause at the end of time slot 1. The pause duration can be calculated by the expected buffer draining rate when X/Y are paused. In this case, the pause is set to be 2 time units as shown in Fig. 2c. At the end of time slot 2, the buffer occupancy reaches its peak of 2 data units and the pause frame reaches X/Y. By repeating this process, the predictive PFC

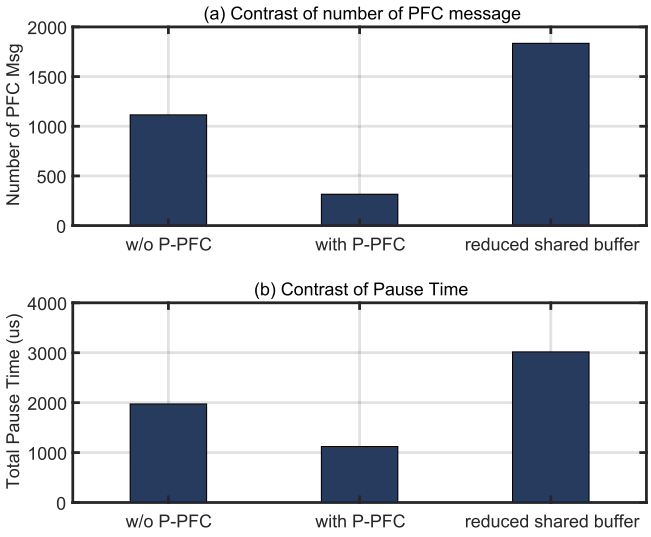


Fig. 3. Number of PFC messages and pause time.

can control the buffer size before the end-to-end congestion control takes effect. Note that $P4$ is still sending at its full rate during the whole process so that the throughput is not harmed. Compared with the traditional approach, the average latency is only $(1+2+1)*2/8=1$ time units, and the tail latency is 2 time units. Both the average and tail latency are reduced by half. In this way, P-PFC provides a quick reaction mechanism to ephemeral congestions.

Note that P-PFC is different from QCN [7]. QCN only reacts to the derivation of egress queues, while P-PFC considers both ingress and egress buffer usage. Besides ingress queues, CaPFC [25] monitors the length of egress queues. CaPFC sets a fixed threshold of egress queue length to trigger PFC proactively. As a comparison, P-PFC takes the consumption rate of the ingress buffer into account.

2.4 Simply Reduce Shared Buffer?

An interesting question is: *Is P-PFC equivalent to simply reducing the allowed usage of shared buffer?* For example, if we simply reduce the shared buffer to 1 data unit in Fig. 2, we can control the buffer size similar to P-PFC. However, there is a fundamental difference between P-PFC and the straight forward approach of reducing shared buffer. The prediction made by P-PFC is based on both the current buffer occupation and the derivative of the buffer occupation. By predicting the future buffer size, P-PFC only reacts to sudden buffer surges that cannot be timely handled by end-to-end schemes. If the buffer occupation increases slowly, P-PFC would not over-react by triggering the PFC at an early stage. Instead, P-PFC would allow the buffer to increase to a moderate size and let end-to-end schemes to handle the congestion with a fine-grained way.

To compare P-PFC with reduced shared buffer, we perform experiments on a 16:1 incast scenario. We first measure the peak buffer occupation of P-PFC and then reduce the shared buffer of a standard PFC algorithm to the same value (*i.e.*, around 1.1 MB). In this way, the peak buffer occupation for P-PFC and reduced shared buffer approach are the same. We also run the same simulation without modifying the PFC threshold and purely use DCQCN to control the traffic. From Fig. 3, we observe that P-PFC generates much less PFC messages than both the other two approaches. If reduce shared buffer, the system generates a large amount of PFC due to the false alarming when the buffer occupation exceeds the reduced value. For the unmodified approach, the system generates more PFC messages because it needs to use short pauses to repeatedly control the buffer size before DCQCN takes effect. The total upstream pause time for the reduced shared buffer approach is also much longer than the others by over 1000 μs. This might hurt the upstream flows as well as the throughput. The P-PFC approach has similar total pause time as the unmodified case, showing that it is mostly harmless to the throughput.

TABLE 1
Key Notations in This Paper

n	The number of the senders in our model.
s_i	The ith sender. There are n senders in our model.
$v_i(t)$	The sending rate of sender s_i at time t.
r	The receiver r. There are only one receiver in our model.
$v_r(t)$	The receiving rate of receiver r at time t.
c_i	The link capacity corresponds to the sender i. $v_i(t) \leq c_i$
d_i	Delay between the sender s_i and the buffer.
d	Delay between the buffer and the receiver r.
$q(t)$	The current queue length at t.
Q	The queue size.
K_{min}	The minimum PFC threshold. $K_{min} < Q$
K_{max}	The maximum PFC threshold. $K_{max} < Q$
T_{thres}	The threshold for the remaining time, which is used for P-PFC

3 Theoretical Analysis

In this section, we theoretically analyze the performance of P-PFC compared with standard PFC. We first present our network model. Without loss of generality, our model captures the $n:1$ incast network scenario, where n sources s_i communicate with one destination r simultaneously via a single switch. The sending rate of each source is $v_i(t)$ and the receiving rate of the destination is $v_r(t)$. The queue in the switch begins to build up when $\sum_{i=1}^{n} v_i(t) - v_r(t) > 0$. The delay between the sender s_i and the switch is d_i. The queue size is assumed to be Q, beyond which the packets will be dropped. We denote the current queue length by $q(t)$, which varies with the time t. For traditional PFC mechanism, the switch starts to send PFC pause frame once the queue length is larger than the pre-defined threshold K_{max}, and resume the transmission when the queue length decreases to K_{min}. For convenience, we summarize important notations in Table 1.

Let us introduce three related notations first.

Definition 3.1 The queue length varying rate. *The queue length varying rate $v(t)$ at t can be defined as the difference between the total sending rate and the receiving rate: $v(t) = \sum_{i=1}^{n} v_i(t) - v_r(t)$.*

Definition 3.2 The changes of queue length. *The changes of queue length during the time intervals Δt can be defined as $\int_0^{\Delta t} v(t)\,dt$.*

From the definition above, we can conclude that the current queue length $q(t)$ begins to increase when $v(t) > 0$,

begins to decrease when $v(t) < 0$ and keeps the same when $v(t) = 0$.

Definition 3.3 The remaining time. *The remaining time T can be defined as the division between the vacant queue size and the varying rate of queue length: $T = \frac{Q-q(t^*)}{v(t^*)}$*

The remaining time captures that how fast the total queue space will be used up. It's a reasonable metric to determine when to send PFC pause frame. Compared with the static threshold K_{max} for PFC, P-PFC dynamically send PFC pause frame based on the remaining time and can flexibly adjust the sending time instant.

Theorem 3.1. *The tail latency of P-PFC is less than or equal to that of PFC if the following condition holds.*

$$T_{thres} > \frac{Q - K_{max}}{v(t)}. \tag{1}$$

Proof. Assume the initial time is t_0 in which the switch queue is empty. The queue length increases to K_{max} at $t_0 + \Delta t$, and the switch begins to send PFC pause frame to s_i. We have the following:

$$K_{max} = \sum_{i=1}^{n} \int_{t_0}^{t_0+\Delta t} v_i(t)\, dt - \int_{t_0}^{t_0+\Delta t} v_r(t)\, dt.$$

Due to the transmission delay of PFC pause frame, the sending rate $v_i(t)$ for s_i cannot be reduced to zero immediately and the queue length will go on increasing for a short time. The maximum queue length L_{pfc} for PFC can be given by

$$L_{pfc} = K_{max} + \sum_{i=1}^{n} \int_{t_0+\Delta t}^{t_0+\Delta t+d^*} v_i(t)\, dt - \int_{t_0+\Delta t}^{t_0+\Delta t+d^*} v_r(t)\, dt.$$

where $d^* = \arg\min_i d_i$

For P-PFC, let t^* denotes the time point in which the equation below holds.

$$\frac{Q - q(t^*)}{v(t^*)} = T_{thres}. \tag{2}$$

Combining (1) and (2), we obtain,

$$t_0 \le t^* \le t_0 + \Delta t \tag{3}$$

The current queue length $q(t^*)$ can be formulated as

$$q(t^*) = \sum_{i=1}^{n} \int_{t_0}^{t_0+t^*} v_i(t)\, dt - \int_{t_0}^{t_0+t^*} v_r(t)\, dt.$$

P-PFC proactively sends PFC pause frame at t^*. Taking the transmission delay into account, the maximum queue length L_{PPFC} for P-PFC can be given by

$$\begin{aligned} L_{ppfc} &= q(t^*) + \sum_{i=1}^{n} \int_{t_0+t^*}^{t_0+t^*+d^*} v_i(t)\, dt - \int_{t_0+t^*}^{t_0+t^*+d^*} v_r(t)\, dt \\ &= \sum_{i=1}^{n} \int_{t_0}^{t_0+t^*+d^*} v_i(t)\, dt - \int_{t_0}^{t_0+t^*+d^*} v_r(t)\, dt. \end{aligned} \tag{4}$$

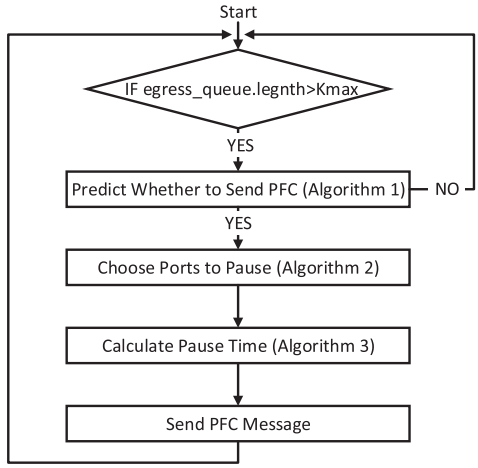


Fig. 4. [P-PFC procedure.]

From (3) and (4), we have,

$$L_{ppfc} \le \sum_{i=1}^{n} \int_{t_0}^{t_0+\Delta t+d^*} v_i(t)\, dt - \int_{t_0}^{t_0+\Delta t+d^*} v_r(t)\, dt = L_{pfc}.$$

The inequation $L_{ppfc} \le L_{pfc}$ concludes the proof. □

Theorem 3.2. *P-PFC cannot result in the throughput loss if the pausing time satisfies the following condition.*

$$T_p < \frac{q(t^*) + v(t^*) \cdot d^*}{v_r(t^*)}$$

where t^* satisfies the Equation (2).

Proof. The PFC pause frame is triggered by the switch at t^*, the current queue length is $q(t^*)$. When the PFC pause frame arrives each source s_i, the queue length becomes $q(t^*) + v(t^*) \cdot d^*$. Since the sending rate has already reduced to zero, the queue will become empty after $\frac{q(t^*)+v(t^*)\cdot d^*}{v_r(t)}$. Thus the necessary and sufficient condition without throughput loss for P-PFC is $T_p < \frac{q(t^*)+v(t^*)\cdot d^*}{v_r(t^*)}$. □

4 System Design

4.1 Design Overview

P-PFC works on switches (congestion point) without changes to end hosts. Fig. 4 shows P-PFC generation procedure. P-PFC first checks whether the length of egress queue exceeds the ECN threshold of K_{max} and only performs prediction when there is congestion. This is to allow the end-to-end congestion schemes to get enough congestion signals like ECN tagged packets while P-PFC is working. We will explain why to use egress queue in detail later. Under congestion, P-PFC performs the following these steps repeatedly with an interval of 1 μs: (1) predicts whether PFC should be proactively triggered, if it is true then do (2) chooses the ports to send the PFC pause, and (3) calculates the pause time.

4.2 Prediction

To predict whether we should send the PFC pause or not, P-PFC first monitors the consumption speed of the shared

buffer. This is done by counting the buffer increase for each ingress queue with an interval of 1 μs. [$increment[port][priority]$ represents the buffer increase for each ingress queue and each priority.] We then sum up the counts for individual ports to get $Increment_{total}$, the increase of the whole switch memory. With the size of the free space in the shared buffer, $switch\_space_{left}$, we can calculate the time required for the shared buffer to be used up. If the remaining time is smaller than a given threshold of T, P-PFC will consider sending a predictive PFC pause message, as shown in Algorithm 1. Note that the predictive PFC is based on the prediction of future buffer size, rather than the current buffer size. This allows P-PFC to react for a predefined time interval of T before the standard PFC. In the meanwhile, if it still takes some time to use up the shared buffer, P-PFC would wait to see if other congestion control mechanisms would help.

Algorithm 1. Prediction

Require: Increment of each port and priority
Ensure: Whether to send PFC message
for each $port$ and $priority$ **do**
  Get $increment[port][priority]$
  $Increment_{total}$ += $increment[port][priority]$
end for
$remainingtime = \frac{switch\_space_{left}}{Increment_{total}}$
if $remainingtime < T$ **then**
  Run Algorithm 2 and Algorithm 3
else
  exit
end if

Algorithm 2. Choosing the Ports to Pause

Require: Composition of the egress queue, Increment of each port and priority
Ensure: A group of ports that should be paused
while $fraction_{total} < R$ **do**
  Choose ports randomly according to their fraction in egress queue
  if $increment[port][priority] > 1$ **then**
    Mark the port
    $fraction_{total}$ += $fraction[port][priority]$
  end if
end while

4.3 Choosing the Ports to Pause

After P-PFC determines that PFC message should be sent, the next step is to determine the ports that we should temporally pause. When selecting the ports to pause, we should not hurt the innocent flows and find out the ringleader of the congestion.

To protect innocent flows, we first rule out ports that have a decreasing buffer occupation. This can be done by finding out ingress queues with negative buffer increase speeds. Second, we choose ports with a probability that is proportional to its buffer occupation in a similar way as in RED. Considering the example shown in Fig. 5, the port A accounts for 80 percent of the egress queue, port B and port C accounts for 10 percent respectively. In this example, port A will be paused with a probability of 80 percent. Third, we will only pause a fraction of candidate ports. As shown in Algorithm 2, we iteratively select ports to be paused until the group of paused ports account for a fraction of buffer occupation larger than a given threshold of R.

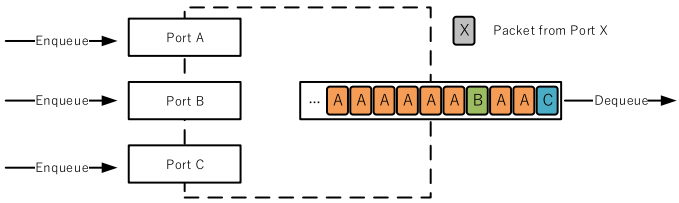


Fig. 5. [Example of packets in an egress queue.]

4.4 Calculating the Pause Time

Now we need to decide how long to pause the upstream is appropriate. According to IEEE 802.3x PFC frame format, numeric values can be used directly to describe the requested duration of PAUSE for each CoS. The PAUSE duration for each CoS is a 2-byte value that expresses time as a number of quanta, where each represents the time needed to transmit 512 bits at the current network speed. While typical implementations will not try to set a specific duration for PAUSE, instead relying on the X-ON and X-OFF style behavior that can be obtained by setting PAUSE for a large number of quanta and then explicitly resuming traffic when appropriate. But we think leveraging this duration wisely rather than using simple X-ON and X-OFF style is more meaningful. The algorithm of how to calculate the pause duration of each port is shown in Algorithm 3. [$packets[port][priority]$ denotes packets number in the egress queue and each priority.] According to Algorithm 3, the pause time contains two parts which correspond to $\frac{packets[port][priority]}{Transmission_{speed}}$ and $\frac{increment[port][priority]}{Transmission_{speed}}$ respectively. The first part of the pause time is used to empty the packets buffered in the egress queue and the second part is used to absorb the packets in flight.

Algorithm 3. Calculate Pause Time

Require: A group of ports that should be paused, Composition of the egress queue, Increment of each port and priority
Ensure: The time that the port should be paused
for each $port$ and $priority$ that should be paused **do**
  Get $packets[port][priority]$ in the egress queue
  $duration[port][priority] = \frac{packets[port][priority]+increment[port][priority]}{Transmission_{speed}}$
end for

Reconsider the example in Fig. 5, port A should be paused longer as it has more packets than the others in the egress queue. We also take the increment of the port into consideration. If the increment of the port is more than other ports, it should be also paused longer and this can help mitigate the upcoming congestion.

5 Evaluation

In this section, we evaluate the performance of P-PFC with DCQCN and TIMELY in a variety of settings using NS-3. At

TABLE 2
NS-3 Simulation Parameters

Parameter	Value	Description
T^*	100 μs	See algorithm 1
R^*	80%	See algorithm 2
K_{max}	200 KB	Parameters of ECN
K_{min}	40 KB	Parameters of ECN
P_{max}	1	Parameters of ECN
g	1/256	Parameter for rate decrease
R_{AI}	40 Mbps	Rate increase step
Timer	55 μs	TIMER for rate increase
Byte Counter	10 MB	Byte Counter for rate increase
CNP Interval	50 μs	Interval between sending CNP

the same time, P-PFC is compared with Congestion Aware Priority Flow Control (CaPFC) proposed in [25] to compare the influence on tail [latency]. We set the speed of each link to 40 Gbps, the delay of each link to 1 μs, the size of shared buffer to 4 MB and the total size of the buffer to 9 MB, the size of each packet to 1 KB and the throughput of each flow to 40 Gbps in all the experiments. To induce incast scenarios, we create the burst of flows that similar to the flows generated by files copy in distributed storage systems like Ceph [26].

5.1 P-PFC Benefits DCQCN and TIMELY

We use the 6:1 and 16:1 incast scenarios to study the performance of P-PFC in terms of the switch queue length and tail latency. Simulation parameters for DCQCN are listed in Table 2. T and R (marked with star) are new parameters introduced in P-PFC. We will discuss them in section 6.1. The value of all other parameters are borrowed from DCQCN and TIMELY paper [11] directly.

P-PFC can reduce the maximum queue length and the tail latency by about 4x when used along with DCQCN and TIMELY. Fig. 6a shows the buffer size of both P-PFC and the standard PFC mechanism. We observe that the maximum switch queue length of P-PFC is 1002 KB, while standard PFC uses up the whole switch buffer space. Due to

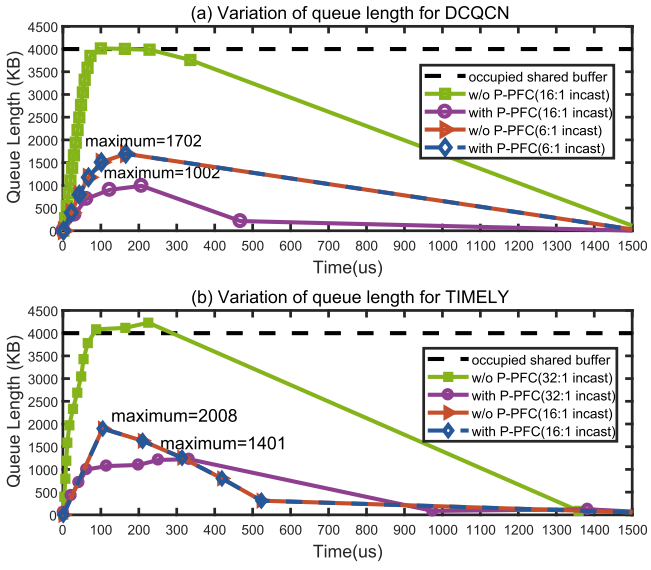


Fig. 6. Performance of DCQCN and TIMELY with/without P-PFC in different incast scenario.

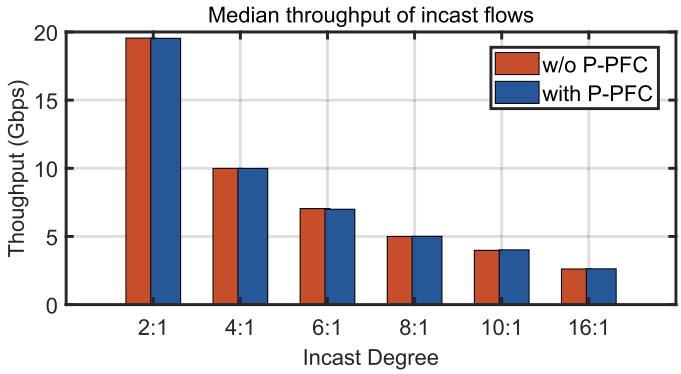


Fig. 7. Throughput under different incast degrees.

lower queue length, we find that the tail latency can also reduce by over 2x. We observe that P-PFC reacts much earlier than the DCQCN and TIMELY due to the prediction based PFC mechanism. For 16:1 incast, the DCQCN convergence time for P-PFC is reduced by 50 percent compared to the standard PFC mechanism. This is because smaller queue length also reduces the control loop length for end-to-end control schemes. We can get similar results from Fig. 6b that means P-PFC can also work well with TIMELY.

5.2 P-PFC Does Not Hurt DCQCN or TIMELY

P-PFC is robust and would not be triggered under moderate congestion. To see whether P-PFC would overreact under moderate congestion, we perform 6:1 and 16:1 incast experiments with/without P-PFC while enabling DCQCN and TIMELY. From Fig. 6a, we observe that the maximum queue length for both case is 1,702 KB and two curves coincide, which means almost no PFC pause messages are sent in both cases. Similar findings have been made from Fig. 6b for TIMELY. In such less congestion scenarios, P-PFC behaves like standard PFC and does not send PFC pauses. Since DCQCN is more common in RDMA networks, we focus on the evaluation of P-PFC+DCQCN below.

P-PFC would not over-set the pause time and hurt the user throughput. We also test the throughput of user traffic under different incast degrees with P-PFC and without P-PFC for DCQCN. The result is shown in Fig. 7, the throughput for both cases are almost the same under different incast degrees, which proves that P-PFC does no harm to throughput or stability at all. Each sever can still be assigned correct throughput by DCQCN with P-PFC.

P-PFC only ignites when length of egress queue exceeds the ECN threshold of K_{max}, in other words P-PFC only starts after DCQCN finishing its work for tagging ECN flags. That means P-PFC will not affect the stability and convergence of DCQCN that has been discussed in the fluid model in [10].

5.3 P-PFC Could Control the False Alarm Rate

In large-scale RDMA network, PFC message is common but dangerous cause too much PFC message could bring all kinds of trouble [9]. So how P-PFC control the false alarm rate to avoid sending too much unnecessary PFC message becomes an important problem. At present P-PFC mainly uses parameter T to guarantee low false alarm rate. For example, the congestion degree of 4:1 and 6:1 incast scenario

TABLE 3
Number of PFC Message with Different T

scenarios	$T = 50\ \mu s$	$T = 100\ \mu s$	$T = 150\ \mu s$
4:1 incast	0	0	0
6:1 incast	0	5	43
16:1 incast	277	316	406

are low so they should not incur any PFC message from standard PFC or P-PFC. As shown in Table 3, by using suitable value of T, P-PFC would send no PFC message. We will discuss how to set T in Section 6.1.

5.4 P-PFC Try to Protect Innocent Flows

P-PFC does consider protecting innocent flows in [many] ways. As shown is Fig. 4, P-PFC is different from traditional PFC, it will only be triggered when the length of egress queues exceeds the threshold instead of ingress queues. That means P-PFC only pauses those flows that really cause the congestion of egress port. We conduct some simple experiments to illustrate this. The topology of this experiment is shown in Fig. 8, all hosts are connected by one switch, host group 1 has 16 hosts, host group 2 has 4 hosts and host group 3 has just one host. All hosts, links and switches used in these testbeds are the same as before. Each host group sends packets to one receiver at 40 Gbps and the path is colored differently. We find only host group 1 receives a lot of PFC messages and neither group 2 nor group 3 receives. Host group 3 receives no PFC message because its flows don't go through the congested egress port, they must not be paused by P-PFC. However, even though there are multiple hosts in host group 2 sending packets to one receiver, it also receives no PFC message because the K_{max} threshold in Fig. 4 plays a role, the threshold can let P-PFC bear transient slight congestion at egress port. This makes P-PFC will not react aggressively, it won't consider sending PFC message until the egress queue length exceeds a threshold. Therefore, P-PFC won't interfere the working of ECN tagging and can leave DCQCN largest space to take effect.

According to Algorithm 2, even for the flows that pass through the same congested egress port, P-PFC will not simply pause them all. P-PFC chooses ports to pause with a probability that is proportional to its buffer occupation. In other words, P-PFC always wants to find out which ports deserve much of the blame for the congestion and then to pause them. We also use a 16:1 incast scenario to illustrate this, but we let hosts send packets at different rates shown in Fig. 9a. 3 hosts use 1 Gbps throughput, 5 hosts use 10 Gbps throughput, 4 hosts use 20 Gbps throughput and the other 4 hosts use 40 Gbps throughput. As shown in Fig. 9b, we snapshoot the occupation of different hosts in egress buffer. Then we can observe

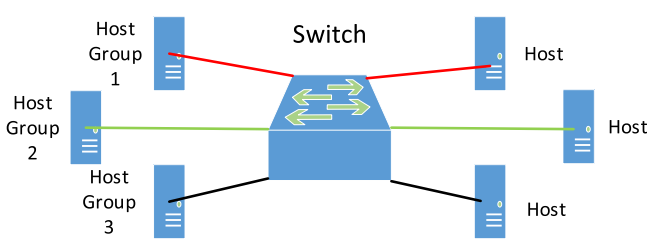


Fig. 8. Topology of the experiment for innocent flows.

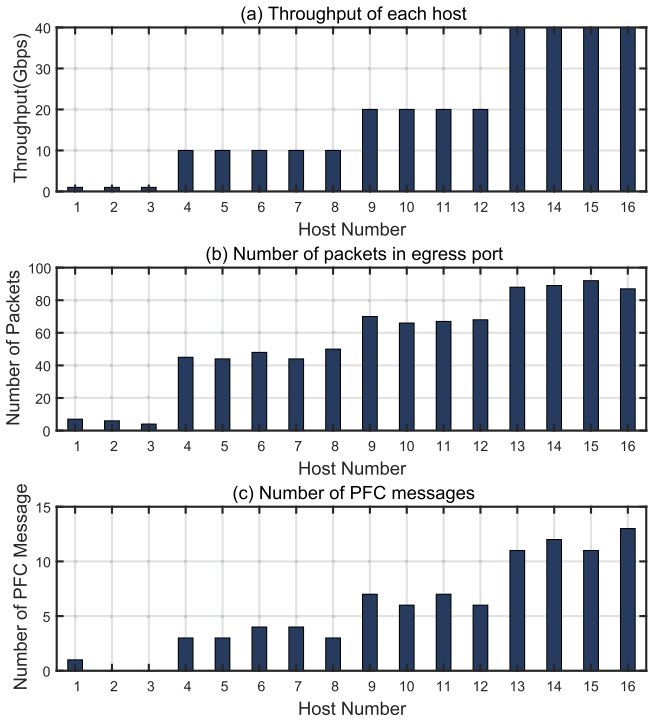


Fig. 9. (a) is the throughput of each host; (b) is the occupation of each host in egress port; (c) is the number of PFC message each host receives.

that the occupation in egress buffer of each host coincides with their throughput in Fig. 9a. From Fig. 9c, we finally find out the occupation in egress buffer of each host results in the number of PFC messages they received directly. Like Algorithm 2 designed, if a host sends packets at lower rate, it will accumulate fewer packets in egress port and P-PFC will send less PFC messages to it.

P-PFC also tries to guarantee fairness by calculating pause time for each host independently and dynamically. With Algorithm 3, we calculate the pause time primarily according to the occupation of each host in egress buffer. The key idea is similar with Algorithm 2. If a host has more packets in egress port then we consider it should take more responsibilities for the congestion and P-PFC will naturally pause it longer. This ensures even a host with low-speed is paused by P-PFC with a small probability, it will not be paused for a long time. That is why P-PFC use a dynamical pause time instead of a fixed value. As shown in Fig. 10, we compare using dynamical pause time and several fixed pause time in

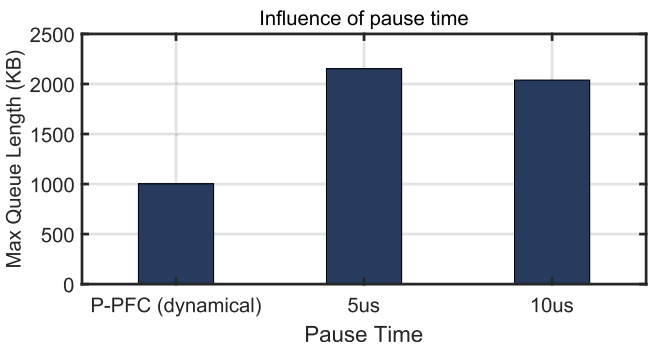


Fig. 10. Influence of different pause time.

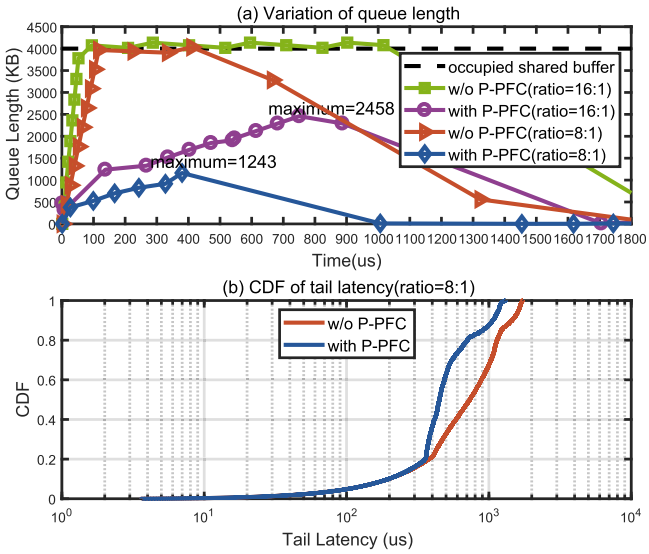


Fig. 11. Performance of DCQCN with/without P-PFC in dumbbell topology.

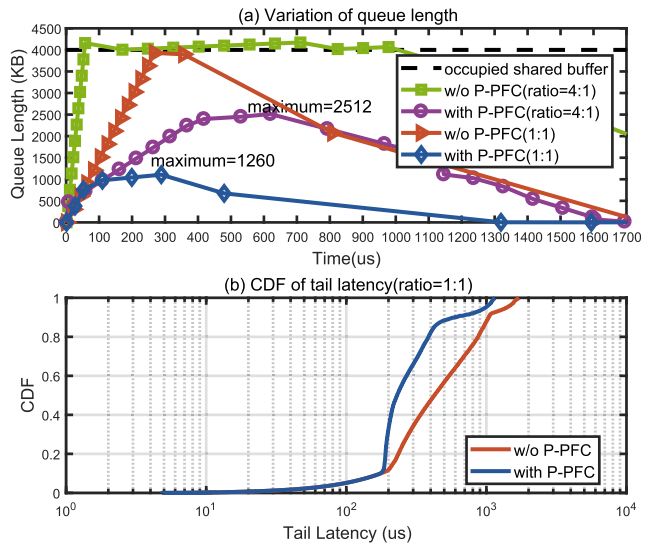


Fig. 12. Performance of DCQCN with/without P-PFC in spine-leaf network topology.

P-PFC. The max queue length can be reduced about half when using dynamical pause time. This is because using fixed pause time can not guarantee no congestion in core network. The queue of switch will still build up if use small fixed pause time. Although use large fixed pause time can avoid this, hosts are also paused too long and throughput are hurt seriously.

5.5 Performance of P-PFC in Different Topologies

For answering this question we build two other testbeds, dumbbell topology and spine-leaf topology. [As shown in Fig. 13,] the dumbbell topology has 2 rack switches connected to each other, and each is connected with 32 servers or 64 servers for testing different oversubscription ratio. [As shown in Fig. 14,] the leaf-spine topology has 6 rack switches connected by 4 core switches, and each is connected with 32 servers or 64 servers for testing different

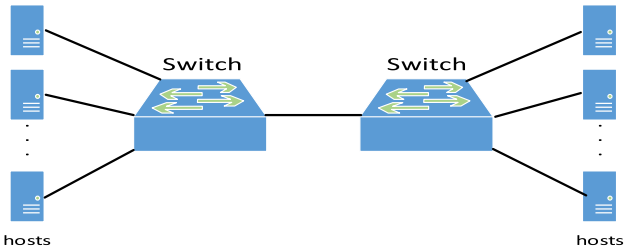


Fig. 13. [Dumbbell topology of testbeds.]

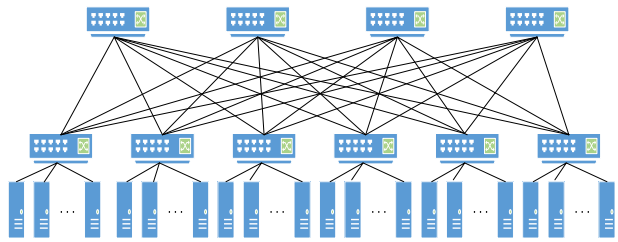


Fig. 14. [Spine-leaf network topology of testbeds.]

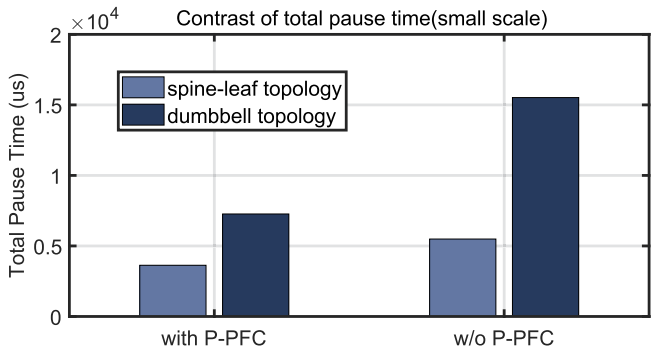


Fig. 15. Total pause time with P-PFC.

oversubscribed ratios. For both topologies, each server is connected via a 10 Gbps link, and the oversubscription ratio are 8:1 and 16:1 respectively.

As shown in Figs. 11 and 12, P-PFC can reduce the maximum queue length and the tail latency by about 4x and 2x under high and low pressure respectively. Notice that we only use the data in the first 1700 microseconds in Figs. 11b and 12b in order to be consistent with Figs. 11a and 12a. According to the CDF in Figs. 11 and 12 we can also find P-PFC will not influence DCQCN during stable phase. The convergence process is also reduced by half when using P-PFC. We also measure the total pause time of P-PFC and tradition PFC (primordial DCQCN) in the these two testbeds. According to Fig. 15, P-PFC can reduce the total pause time than tradition PFC in these two testbeds which means it can also reduce the collateral damages at the same time.

5.6 Influence on Tail Latency

A congestion awareness PFC (CaPFC) has been proposed in [25]. CaPFC is designed to address the deficiency of PFC without QCN. CaPFC also monitors egress queues to increase congestion control awareness. CaPFC introduces a WARN threshold at each egress buffer to measure the congestion. In the 16:1 incast scenario, the influence of P-PFC

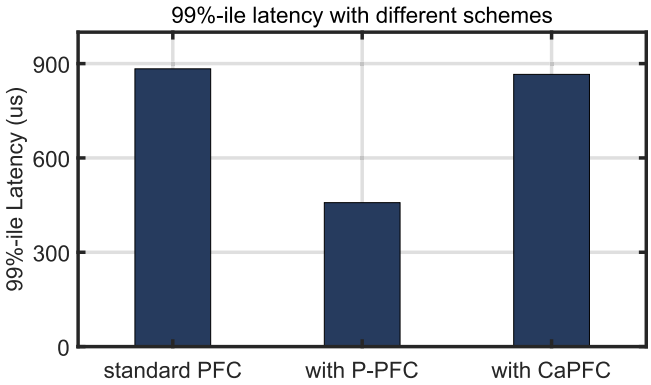


Fig. 16. Tail latency with different schemes.

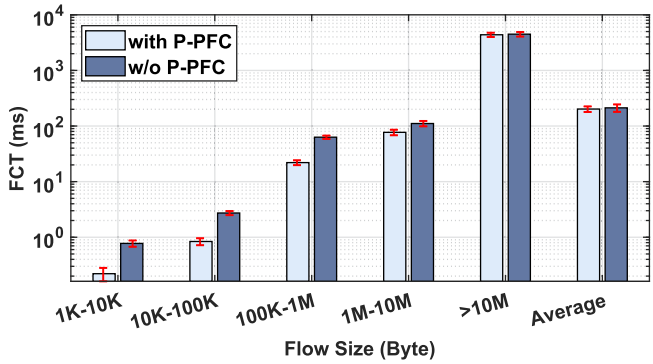


Fig. 18. Flow completion time with different schemes.

and CaPFC on tail latency (99%ile) is shown in Fig. 16. P-PFC can reduce tail latency almost by 2x compare with CaPFC. Although CaPFC can effectively reduce FCT without DCQCN, the reduction of tail latency is not obvious when DCQCN is enabled.

5.7 Benefits of Using P-PFC

We can find the benefits brought by P-PFC outweigh its disadvantages according to Figs. 12 and 15. At the same time, we calculate the total reduced latency of packets during congestion phase in different topologies as shown in Fig. 17. We can find the reduced latency will increase as the topology becomes more complex. This mainly because DCQCN spends more time dealing with congestion when distance between senders and receivers become longer. That makes the degree of congestion be more serious. Notice that the results in Fig. 17 is only for a one-time congestion process. If there is more congestion of workload, the reduced latency will also be more. Compared with total reduced tail latency, the total pause time caused by P-PFC does look negligible.

To be more intuitive, we measure the FCT with different schemes. We use data mining trace whose cumulative distribution functions and descriptions can be found in [27]. As shown in Fig. 18, mice flows (<1 M) can benefit from P-PFC obviously. For example, FCT with different schemes for flows (100 K-1 M) is 22 ms, 38 ms and 63 ms, respectively. This is because P-PFC can prevent the length of queues in switches from growing. Since P-PFC will not hurt the throughput, FCT of elephant flows (>10 M) with different schemes almost remain the same. We also find smaller running interval can bring better performance. Overall, P-PFC allows small flows to benefit without hurting big flows.

P-PFC not only has less flow control, but also has better overall performance. We use the 16:1 incast scenario to study the performance of P-PFC in terms of the average latency when used along with DCQCN. Notice that we only use the data in the first 1500 microseconds in Fig. 6a. [Fig. 19 shows the average latency and 50%ile latency of the standard PFC mechanism, P-PFC and CaPFC. P-PFC can reduce the average latency up to 83.8 percent and reduce the 50%ile latency up to 90 percent compared to CaPFC.] Although CaPFC can reduce tail latency, it does not help average latency.

6 Discussions

In this section, we discuss some interesting and significant questions of P-PFC. First we give a detailed explanation of some innovations in P-PFC. Then we discuss how to implement P-PFC.

6.1 How to Set Parameters in P-PFC?

T and R are new parameters introduced in P-PFC. R decides the number of ports to pause each time P-PFC works, which is used to protect innocent flows. Actually it is not very important to the performance of P-PFC and we recommand 80 percent is good enough in most cases.

T is significant for P-PFC. Different T could make P-PFC more aggressive or more conservative and make a tradeoff between performance and stability. Assume the buffer size of switch is B, the number of ports of switch is P and the rate of each port is V. Then we can get the minimal time T_{min} to fill the switch buffer :

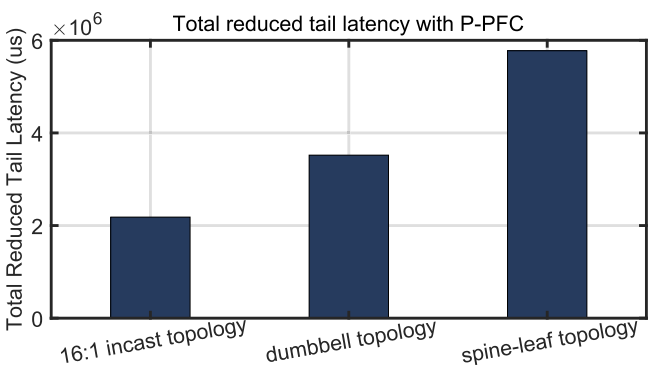


Fig. 17. Total reduced tail latency with P-PFC.

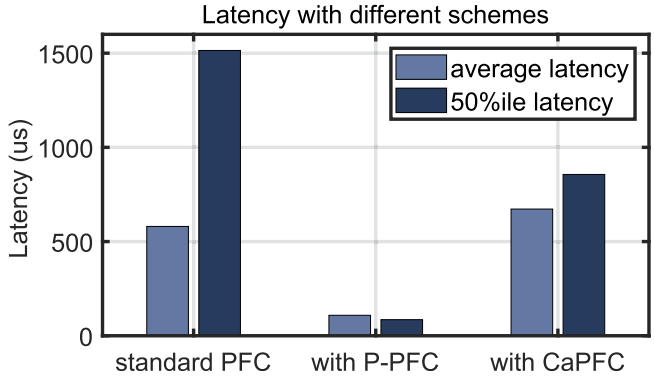


Fig. 19. [Latency with different schemes.]

$$T_{min} = \frac{B}{P \cdot V}. \quad (5)$$

In our experiments, B is 9 MB, P is 16 and V is 40 Gbps so T_{min} is 112.5 μs. T_{min} corresponds to the most serious congestion. Intuitively, P-PFC should launch when $remainingtime$ in Algorithm 1 is close to T_{min}, in our experiments we set T to 100 μs. Users can adjust T according the specific flow pattern. Actually find the optimal T with different topologies and flow patterns is challenging and we leave it in our future work.

6.2 [In Addition to Monitor Ingress Ports, Why Also Monitor Egress Ports in P-PFC?]

As is well-known, traditional PFC is notorious in various ways like HOL blocking and collateral damage. But it still has some advantages such as fast reaction to congestion compared with end-to-end congestion control. DCQCN is a typical end-to-end congestion control protocol, it uses egress queue length as a congestion signal. P-PFC also reacts only when egress queue length exceeds a threshold, it will not disturb DCQCN unless the congestion is very urgent. If P-PFC reacts according to ingress queue length that may be too aggressive, then less packets or even no packet can be tagged with ECN flags, finally delay the process of convergency. While if P-PFC pause hosts too later, the queue length will build up rapidly. Monitor egress queues can make P-PFC cooperate with DCQCN well and choose an appropriate time to send PFC messages. Except this, monitor egress queue can let us find the hosts who should be responsible for congestion intuitively. Actually, how to coordinate hop-by-hop congestion control protocol with end-to-end congestion control protocol is a complicated and interesting work. We leave this problem in our future work.

6.3 Is P-PFC Easy to be Implemented in Commodity Switch?

P-PFC only changes the part of sending algorithm of PFC PAUSE messages, and it is easy to be integrated within the existing PFC implementation. To enable P-PFC, switch needs a counter for each port and priority to get the increment of the ingress queue length in the past 1 μs. Algorithm 1 is executed every 1 μs, in each execution we can get the remaining time by making the left buffer of switch divided by the sum of all the increment of all ports and priorities. If remaining time exceeds threshold T, then Algorithm 1 sends signal to Algorithm 2. To choose which ports to pause, Algorithm 2 randomly selects ports at the egress inside and if the total increment of the selected port and priority exceeds the threshold, then the port and priority will be marked and written into a FIFO. Algorithm 3 gets the port and priority that needs to be paused from the FIFO and calculates the pause time, sends them to the switch PFC PAUSE engine finally. As we mentioned before, in IEEE 802.3x PFC frame format, numeric values can be used directly to describe the requested duration of PAUSE. To avoid sending PFC PAUSE messages repeatedly in a short time, we can set up a register bit for each port and priority. The bit is set after sending PFC messages and will be automatically unset after a while. PFC messages sending is only allowed when the bit is unset.

6.4 Novelty of P-PFC

Actually the idea of P-PFC that uses buffer occupancy as a signal to do congestion control has been studied before [28], [29], [30], [31]. But we find that this approach is rarely used in data centers. Cause the bandwidth of links in data center network are usually above 10 Gbps, the transmission of packets and change of queue lengths in switch are very fast. P-PFC combines this approach and traditional PFC mechanism naturally. PFC is a reactive protocol that works in the switch. It can monitor the length of queues in real time. P-PFC takes advantage of this and uses it to predict the future condition and makes some countermeasures in advance. Actually there are many other works [32] that try to discard PFC mechanism directly to avoid the collateral damages that PFC brings. According to our tests P-PFC can reduce the collateral damages, it tries to improve PFC instead of discarding it.

Traditional traffic shaping technology works on egress queues and PFC works on ingress queues. Both of them just consider the local knowledge. While P-PFC considers both ingress and egress queues to do the congestion control. To our best of knowledge, P-PFC is the first protocol that combines buffer occupancy technology and PFC mechanism to solve severe congestion control problem in data center networks.

7 Related Work

[There exists a number of end-2-end congestion control proposals (*e.g.*, DCQCN [10], TIMELY [11], HPCC [33] and Blitz [34]) in layer 3 recently. PFC is a per-hop flow control in layer 2 which guarantees lossless of packets. P-PFC is an enhanced version of PFC. They are orthogonal to those layer 3 congestion control researches. Both layer 2 per-hop mechanisms and layer 3 end-2-end congestion control are necessary to provide high performance RDMA in Ethernet.]

[*Congestion control in RDMA.* DCQCN [10] takes advantage of Early Congestion Notification (ECN) marks to infer imminent congestion. TIMELY [11] uses round-trip times (RTT) as a congestion signal for rate control. DCQCN+ [35] improves performance for large-scale incasts in RDMA networks. HPCC [33] uses in-network-telemetry (INT) to improve end-host based congestion control. Blitz [34] is a Receiver-oriented Congestion Control (RCC) mechanism which employs a divide-and-specialize approach to speeds up the convergence.]

[*Flow control in RDMA.* Congestion Aware Priority Flow Control (CaPFC) in [25] also observed that PFC is not effective in many congestion cases. CaPFC monitors both ingress and egress queues, and triggers PFC with a fixed threshold value of per-input counters. Instead, P-PFC uses derivatives of buffer occupancy and has better performance in reducing tail latency.]

8 Conclusions

In this paper, we have proposed P-PFC, a predictive PFC mechanism for RDMA networking. P-PFC predicts the buffer occupation based on the derivative of queues. We

show that with the predictive PFC, we can control the queue length and reduce the tail latency of RDMA packets. Although using derivative for queue management has been used before for TCP/IP networks [36], [37], to our best knowledge, this is the first work of using derivative in lossless RDMA networks.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. This research was supported by the National Key R&D Program of China 2018YFB1003505, the National Natural Science Foundation of China under Grant Numbers 61772265, and 61802172, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

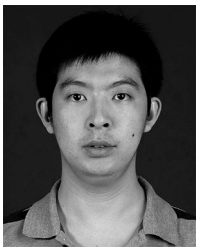
References

[1] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 103–114.
[2] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast remote memory," in *Proc. 11th USENIX Conf. Netw. Syst. Design Implementation*, 2014, pp. 401–414.
[3] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, 2014, pp. 295–306.
[4] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "MALT: Distributed data-parallelism for existing ML applications," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, Art. no. 3.
[5] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast in-memory transaction processing using RDMA and HTM," in *Proc. 25th Symp. Operating Syst. Princ.*, 2015, pp. 87–104.
[6] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and concurrent RDF queries with RDMA-based distributed graph exploration," in *Proc. 12th USENIX Symp. Operating Syst. Design Implementation*, 2016, pp. 317–332.
[7] M. Alizadeh *et al.*, "Data center transport mechanisms: Congestion control theory and IEEE standardization," in *Proc. 46th Annu. Allerton Conf. Commun. Control Comput.*, 2008, pp. 1270–1277.
[8] D. Cohen *et al.*, "Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options," in *Proc. 17th IEEE Symp*, 2009, pp. 123–130.
[9] C. Guo *et al.*, "RDMA over commodity ethernet at scale," in *Proc. Conf. ACM SIGCOMM Conf.*, 2016, pp. 202–215.
[10] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 2015, pp. 523–536.
[11] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.
[12] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "ECN or delay: Lessons learnt from analysis of DCQCN and timely," in *Proc. 12th Int. Conf. Emerg. Netw. Experiments Technol.*, 2016, pp. 313–327.
[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.
[14] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: https://rfc-editor.org/rfc/rfc3168.txt
[15] InfiniBand Trade Association, "Supplement to infiniband architecture specification volume 1, release 1.2. 1: Annex a16: RDMA over converged ethernet (RoCE)," Apr. 2010.
[16] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 303–314, 2009.
[17] Learn What You Will, "Priority flow control build reliable layer 2 infrastructure," 2015.
[18] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," *CoNEXT*, pp. 13:1–13:12, Jan. 2010.
[19] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Operating Syst. Design Implementation*, 2010, pp. 307–320.
[20] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, "PAC: Taming TCP incast congestion using proactive ACK control," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, 2014, pp. 385–396.
[21] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 63–74, 2010.
[22] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: Rapid precise packet loss notification in data center," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 17–28.
[23] A. Bechtolsheim, L. Dale, and H. Holbrook, "Why big data needs big buffer switches," 2016. [Online]. Available: https://www.arista.com/assets/data/pdf/Whitepapers/BigDataBigBuffers-WP.pdf
[24] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.
[25] S. N. Avci, Z. Li, and F. Liu, "Congestion aware priority flow control in data center networks," in *Proc. IFIP Netw. Conf.*, 2016, pp. 126–134.
[26] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, 2006, pp. 307–320.
[27] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.
[28] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," in *Proc. 20th Annu. Joint Conf. IEEE Comput. Commun. Societies*, 2001, pp. 631–640.
[29] P. P. Mishra and H. Kanakia, "A hop by hop rate-based congestion control scheme," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 4, 1992, pp. 112–123.
[30] S. Jagannathan and J. Talluri, "Predictive congestion control of ATM networks: multiple sources/single buffer scenario," *Automatica*, vol. 38, no. 5, pp. 815–820, 2002.
[31] L. Benmohamed and S. M. Meerkov, "Feedback control of congestion in packet switching networks: The case of a single congested node," *IEEE/ACM Trans. Netw.*, vol. 1, no. 6, pp. 693–708, Dec. 1993.
[32] R. Mittal *et al.*, "Revisiting network support for rdma," pp. 313–326, 2018, *arXiv: 1806.08159*.
[33] Y. Li, "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 44–58.
[34] J. Xue, M. Chaudhry, B. Vamanan, T. Vijaykumar, and M. Thottethodi, "Fast congestion control in RDMA-based datacenter networks," in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2018, pp. 24–26.
[35] Y. Gao, Y. Yang, T. Chen, J. Zheng, B. Mao, and G. Chen, "DCQCN+: Taming large-scale incast congestion in RDMA over ethernet networks," in *Proc. IEEE 26th Int. Conf. Netw. Protocols*, 2018, pp. 110–120.
[36] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proc. 20th IEEE Annu. Joint Conf. IEEE Comput. Commun. Societies*, 2001, pp. 1726–1734.
[37] J. Aweya, M. Ouellette, and D. Y. Montuno, "A control theoretic approach to active queue management," *Comput. Netw.*, vol. 36, no. 2, pp. 203–235, 2001.

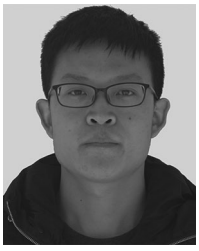
Chen Tian received the BS, MS, and PhD degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is an associate professor at the State Key Laboratory for Novel Software Technology, Nanjing University, China. He was previously an associate professor at the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming, and urban computing.

Bo Li received the BS degree from the Department of Computer Science and Engineering, Nanjing University of Science and Technology, China, in 2016. He is working toward the MS degree at Nanjing University, China. His research interest includes distributed networks and systems.

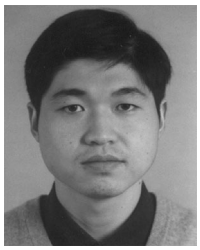
Liulan Qin received the BS degree from the Department of Computer Science and Engineering, Nanjing University of Science and Technology, China, in 2018. She is currently working toward the MS degree in the Department of Computer Science and Technology, Nanjing University, China. Her research interests include datacenter networks.



Jiaqi Zheng received the PhD degree from Nanjing University, China, in 2017. He was a research assistant with the City University of Hong Kong, in 2015, and a visiting scholar with Temple University, in 2016. He is currently a research assistant professor with the Department of Computer Science and Technology, Nanjing University. His research interests include computer networking, particularly, data center networks, SDN/NFV, and machine learning systems. He is a member of ACM. He received the Best Paper Award from IEEE ICNP 2015, the Doctoral Dissertation Award from ACM SIGCOMM China 2018, and the First Prize of the Jiangsu Science and Technology Award in 2018.



Jie Yang received the BS degree in computer science from Nanjing University, China. He is currently working toward the master's degree at Nanjing University at Computer Science Department His research interests include networking and computer architecture.



Wei Wang received the MS degree from the ESE Department, Nanjing University, in 2000, and the PhD degree from the ECE Department, National University of Singapore, in 2008. He is currently an associate professor with the CS Department, Nanjing University. His research interests include area of wireless networks, including device-free sensing, cellular network measurements, and software defined radio systems.

Guihai Chen received the BS degree in computer software from Nanjing University, in 1984, the ME degree in computer applications from Southeast University, in 1987, and the PhD degree in computer science from the University of Hong Kong, in 1997. He is a distinguished professor of Nanjing University. He had been invited as a visiting professor by Kyushu Institute of Technology in Japan, University of Queensland in Australia and Wayne State University, USA. He has a wide range of research interests with focus on parallel computing, wireless networks, data centers, peer-to-peer computing, high-performance computer architecture and data engineering. He has published more than 350 peer-reviewed papers, and more than 200 of them are in well-archived international journals such as the *IEEE Transactions on Parallel Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE/ACM Transactions on Networking* and *ACM Transactions on Sensor Networks*, and also in well-known conference proceedings such as HPCA, MOBIHOC, INFOCOM, ICNP, ICDCS, CoNext and AAAI. He has won 9 paper awards including ICNP 2015 Best Baper Award and DASFAA 2017 Best Paper Award.

Wanchun Dou received the PhD degree in mechanical and electronic engineering from the Nanjing University of Science and Technology, China, in 2001. He is currently a full professor with the State Key Laboratory for Novel Software Technology, Nanjing University. From April 2005 to June 2005 and from November 2008 to February 2009, he respectively visited the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, as a visiting scholar. Up to now, he has chaired three National Natural Science Foundation of China projects and published more than 60 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.