This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Re-Architecting Buffer Management in Lossless Ethernet

Hanlin Huang, *Graduate Student Member, IEEE*, Xinle Du, Tong Li, *Member, IEEE*, Haiyang Wang, *Member, IEEE*, Ke Xu, *Fellow, IEEE, Member, ACM*, Mowei Wang, and Huichen Dai, *Member, IEEE*

***Abstract*— Converged Ethernet employs Priority-based Flow Control (PFC) to provide a lossless network. However, issues caused by PFC, including victim flow, congestion spreading, and deadlock, impede its large-scale deployment in production systems. The fine-grained experimental observations on switch buffer occupancy find that the root cause of these performance problems is a mismatch of sending rates between end-to-end congestion control and hop-by-hop flow control. Resolving this mismatch requires the switch to provide an additional buffer, which is not supported by the classic dynamic threshold (DT) policy in current shared-buffer commercial switches. In this paper, we propose Selective-PFC (SPFC), a practical buffer management scheme that handles such mismatch. Specifically, SPFC incrementally modifies DT by proactively detecting port traffic and adjusting buffer allocation accordingly to trigger PFC PAUSE frames selectively. Extensive case studies demonstrate that SPFC can reduce the number of PFC PAUSEs on non-bursty ports by up to 69.0%, and reduce the average flow completion time by up to 83.5% for large victim flows.**

***Index Terms*— Priority-based flow control, lossless network, buffer management, mismatch.**

I. Introduction

MICRO-BURST, exemplified by incast, is a typical traffic pattern in modern data center networks [1], [2], [3]. Generally, it refers to bursty traffic with a microscopic time scale that cannot be regulated by end-to-end congestion control. It is typically generated by various online data-intensive applications [4] and virtualized services [5], which appear in switches when packets of multiple concurrent flows are destined for the same output port. Packet dropping caused by micro-burst traffic might be unacceptable because micro-burst traffic is usually comprised of several delay-sensitive short flows, and the triggered timeouts always extend the flow completion time (FCT) [3], [6], [7].

In recent years, lossless networks have become the trend of data centers [8], [9], [10] and cluster computing systems [11], [12], [13], which can avoid loss retransmissions. In addition, scaling transport protocols such as Remote Direct Memory Access (RDMA) in data centers require reliable transport without congestion loss [14], [15]. In order to provide a lossless network switching fabric, Converged Ethernet employs the Priority-based Flow Control (PFC) [16] mechanism, which ceases the upstream port before the buffer overflows and resumes packet sending after queue decreasing.

The decision maker of the PFC threshold, i.e., the PFC trigger, depends on the buffer architecture and buffer management policy in a switch. Today, most commercial switches employ on-chip shared memory to reduce latency by avoiding packet reads and writes from/to external storage [3]. They share the input queue to trigger PFC by counting ingress bytes [17]. To guarantee efficiency, every port can access the shared memory pool, meaning that any single port can theoretically use as much as the entire buffer space. However, shared memory switches might suffer the fairness problem that few ports could occupy all shared buffers, starving other ports. To overcome the problem, many buffer management policies were proposed to restrict the queue length on each port.

Among them, the classic Dynamic Threshold (DT) policy [18] has been widely used by switch manufacturers [19]. In this policy, the queue length is restricted by a dynamic threshold shared by *all ports*, proportional to the current amount of free buffer space. All ports with sufficient traffic to warrant thresholding should obtain the same amount of space. Each port attempts to limit its length to some function of the unused buffer space. However, we find that the classic DT policy has limitations. It treats every switch port fairly and fails to make full use of buffers to improve performance according to the characteristics of the lossless network.

Manuscript received 23 August 2023; revised 19 January 2024 and 3 June 2024; accepted 11 July 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Park. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102301; in part by the NSFC Projects under Grant 61932016, Grant 62132011, Grant 62221003, and Grant 62202473; in part by China National Funds for Distinguished Young Scientists under Grant 62425201; and in part by the China Computer Federation (CCF)-Huawei Populus Euphratica Innovation Research Funding. *(Corresponding authors: Tong Li; Ke Xu.)*

Hanlin Huang is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: hhl21@mails.tsinghua.edu.cn).

Xinle Du and Mowei Wang are with Huawei, Shenzhen 518129, China (e-mail: duxinle1@huawei.com; mowei.wang@icloud.com).

Tong Li is with the Key Laboratory of Data Engineering and Knowledge Engineering, Information School, Renmin University of China, Beijing 100872, China (e-mail: tong.li@ruc.edu.cn).

Haiyang Wang is with the Department of Computer Science, University of Minnesota Duluth, Duluth, MN 55803 USA (e-mail: haiyang@d.umn.edu).

Ke Xu is with Beijing National Research Center for Information Science and Technology (BNRist) and the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with Zhongguancun Laboratory, Beijing 100094, China (e-mail: xuke@tsinghua.edu.cn).

Huichen Dai is with ByteDance, Beijing 100089, China (e-mail: dhconly@gmail.com).

Digital Object Identifier 10.1109/TNET.2024.3430989

1558-2566 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

To explain this more clearly, we have conducted empirical and experimental investigations to attain an in-depth understanding of buffer management architecture in lossless Ethernet. We can find that burst traffic causes throughput loss on non-burst traffic ports, resulting in the ***victim traffic phenomenon***. The root cause of this performance issues is due to the ***mismatching*** between the fast hop-by-hop PFC flow control and the relatively slow end-to-end congestion control. This mismatch may occur during the interaction between burst flows and non-burst flows. And this can be further attributed to the PFC DT policy that all the ingress ports have an identical PFC threshold. For burst flows, the rate regulation through congestion control is difficult, and the burst traffic ports can only be suspended through messages called PFC PAUSE to stop sending. For non-burst flows, end-to-end congestion control needs buffers (time) to adjust to the target sending rate [20], [21]. Hence ports with non-burst traffic also trigger PFC PAUSE, causing the victim traffic phenomenon.

In light of these insights, we re-architect buffer management in lossless Ethernet. We aim to propose a buffer management policy that can buffer non-burst mismatch traffic to alleviate the victim traffic phenomenon problem. Designing such a policy can be challenging because it requires switch ports to proactively detect port traffic and adjust buffer allocation accordingly. More specifically, a switch port must determine the type of traffic it transmits on time with limited port-level information, such as queue length, packet count, and input/output rate.

In this paper, we propose Selective-PFC (SPFC), a buffer management policy that controls the buffer allocation of shared memory switches based on real-time detection of port traffic status. The primary objective of SPFC is to fully utilize the shared buffer to absorb mismatch traffic, avoid a larger performance penalty by sacrificing the burst port's buffer for a short time strategically, and be friendly to applications in data centers that emphasize tail latency.

In particular, SPFC uses a set of port-wise control states to differentiate the status of different ports and impose different thresholds on individual ports accordingly. The transition between different port control states is determined by a state decision module that detects port traffic in real-time. The state decision module of SPFC is only composed of several counters and triggers. When port traffic changes, SPFC proactively raises or lowers the port threshold according to the port control state to allocate buffer among different ports dynamically.

We evaluate SPFC with ns-3 [22] simulation and DPDK [23] testbed, comparing SPFC with existing buffer management policies. Extensive results demonstrate that SPFC can reduce the number of PFC PAUSEs on victim ports in burst tolerance scenarios by up to 69.0%. In real workload, SPFC reduces the average FCT by up to 83.5% for large victim flows, compared to the classic schemes DT and ST (Static Threshold). When there is a deadlock topology, SPFC can also provide better performance. In summary, we make the following contributions:

- SPFC is the first PFC threshold adjustment policy, to the best of our knowledge, that reduces unnecessary PFC triggerings.
- A detailed design of threshold determination is provided based on ingress port states and the state decision module.
- Solid testbed experiments and large-scale simulations with real workloads have verified SPFC's performance.

The rest of the paper is organized as follows. We introduce the background in Section II. Section III reveals inadequate elements in the existing buffer management architecture for lossless Ethernet and gives the motivation for our solution. The detailed design of SPFC is demonstrated in Section IV. Then we discuss the implementation and design details in sections V and VI. In Section VII, we evaluate SPFC. Section VIII summarizes related work, and Section IX concludes the paper.

II. Background

A. Traffic Workload in Data Centers is Bursty

The traffic demands of the various applications in data centers are complex in terms of distributions and patterns [24], [25], [26]. For example, WebSearch [27] service usually generates short and burst flows, while distributed data storage applications introduce a few long-lived flows to transfer the bulk of data.

Many surveys of operation data centers show a wide distribution of traffic patterns [28]. Short flows can be tiny (0.05 KB), whereas long flows can be huge (over 100 MB). Short flows contributed the most traffic connections, while long flows contributed the most bytes of traffic. Due to the high bandwidth-delay product (BDP) in the data center (e.g., 100 Gbps $\times$ 20 us $=$ 250 KB), most short flows cannot be adjusted by an end-to-end congestion control scheme finishes sending all packets before receiving any ACKs.

Besides, many measurements [1], [24], [28], [29] show that the occurrence of short flow is frequent and bursty. The frequent burst of short flows will shock the switch buffer, resulting in RTT instability and buffer overflow [3], [6], [28]. Although these flows do not react to congestion control schemes, they seriously interfere with the regular congestion management of switching fabric in data centers.

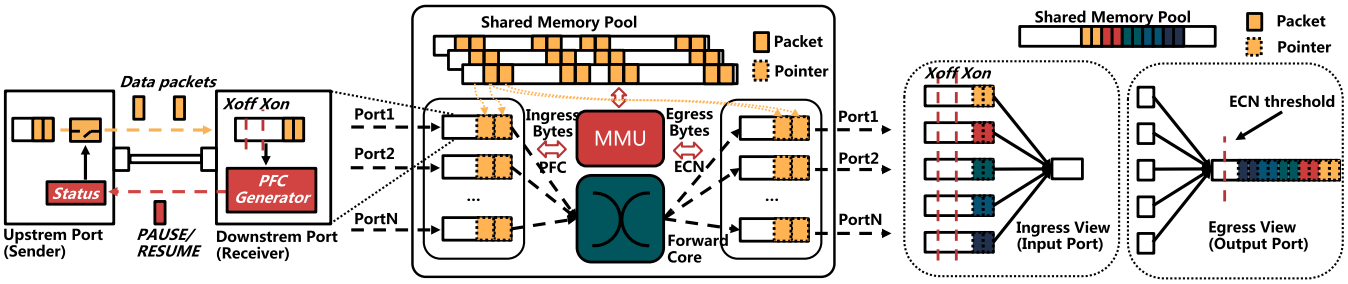
B. Priority-Based Flow Control in Lossless Networks is Inevitable

Many cloud providers deploy RDMA due to ultra-low latency, high throughput, and low CPU overhead. To deploy RDMA in Ethernet networks, PFC is needed to enable a lossless Ethernet fabric. PFC prevents buffer overflow on switches and NICs [30], [31]. As shown in Fig. 1(a), the receiver pauses the sender when the ingress queue length exceeds X_{off}. The receiver will resume the sender When the ingress queue length decreases below X_{on}. PFC specifies up to eight priority classes. PAUSE/RESUME packets specify the priority classes to which they are applied.

Although PFC can guarantee zero packet loss due to network congestion, it also causes some well-known performance issues such as head-of-line blocking (HLB), victim traffic phenomenon and even deadlock [30], [32], [33], [34]. Because PFC is a coarse-grained mechanism, it operates at the port (or port plus priority) level and does not distinguish between

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Flow controls for zero packet loss. (b) The architecture of a shared memory switch. (c) The Ingress view and Egress view in MMU.

Fig. 1. The model of a shared memory switch.

flows. The solution for these performance issues is to eliminate persistent congestion by end-to-end congestion control schemes such that PFC is not triggered incessantly [34], [35], [36]. The RDMA needs PFC to prevent packet loss due to transient congestion of uncontrollable bursts, and RDMA also needs end-to-end congestion control to eliminate persistent congestion. In this case, the end-to-end congestion control and hop-by-hop lossless flow control are complement each other.

C. Two Perspectives of PFC-Enabled Shared Memory Switch

Fig.1(b) shows the simplified architecture of a typical PFC-enabled shared memory switch. The architecture can be divided into three components: forwarding core, memory management unit (MMU), and shared memory pool. Each ingress port maintains an $IngressBytes$ counter which is the bytes entered from this ingress port, and the packets are still queued in the switch egress. Indeed, $IngressBytes$ is the input queue size. When the packet enters a port, the MMU checks whether the $IngressBytes$ of the current port exceeds the X_{off} threshold for triggering the PFC PAUSE, and then updates the $IngressBytes$ counter and the $EgressBytes$ counter of the corresponding output port. Once the port triggers PFC PAUSE, the port sends a RESUME message only when the $IngressBytes$ counter is reduced to X_{on}. It is worth noting that the ingress bytes are counted for the PFC trigger, while the egress bytes are counted for the ECN trigger. Next, we will further illustrate the principle of PFC-enabled switches.

Fig.1(c) shows a 5-to-1 incast traffic pattern with two views in the MMU. Five flows (each flow has two packets) enter the switch through 5 ports and leave the switch through one port. Physically, these ten packets are all stored in the shared memory pool. But from the MMU's perspective, different views have different functions:

(a) PFC trigger works on the ingress view. From the ingress view, the packets of each flow are queued at the corresponding input port in virtual. When the queue length of an input port ($IngressBytes$) reaches the PFC threshold, PFC PAUSE is triggered. The $IngressBytes$ counters and DT algorithm determine whether PFC should be triggered.

(b) ECN trigger works on the egress view. From the egress view, the packets of each flow are all queued at the output port. When the output queue length reaches the threshold, each packet will be labeled as specified by ECN policy [3], [37].

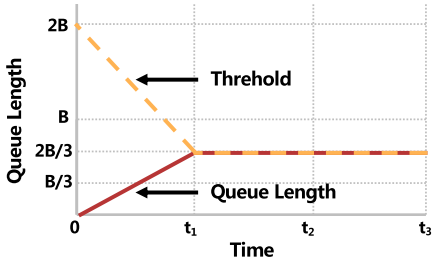


Fig. 2. Queue length and threshold evolutions.

Each port has its own reserved buffer and headroom [38], which are used to absorb packets after sending PFC PAUSE until the upstream is actually paused. To allocate buffer for ports, shared memory switches rely on a specific buffer management policy deployed on MMU, which we will discuss next.

D. Classic Buffer Management Policies Treat Each Port Fairly

Classic dynamic threshold scheme (DT) [18] is the most widely used buffer management policy by switch vendors [39], which determines whether the ingress port triggers PFC. DT sets its threshold, which is shared by all ports, proportional to the current amount of unoccupied buffer space. More specifically, the threshold at time t can be calculated by

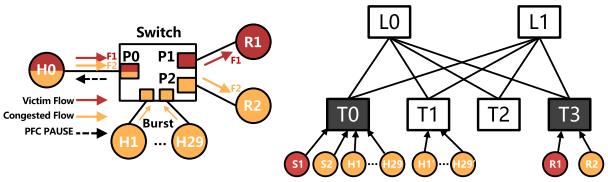
$$T(t) = \alpha \cdot \left(B - \sum_i Q_i(t) \right) \tag{1}$$

Where $T(t)$ is the threshold at time t, B is the total buffer size, $Q_i(t)$ is the queue length of port i at time t, and α is a control function normally set to a constant value for simplicity. To avoid unfairness when traffic changes, DT reserves a certain amount of buffer in a "stable state" (i.e., when queue length is equal to the port threshold).

To understand DT, consider the following scenario. Assuming that the switch buffer is empty and only one priority is used. The port k will be overloaded at time $t = 0$, then $\sum_i Q_i(t) = Q_k(t)$ when $t = 0^+$. Let $\alpha = 2$, then $T(t) = 2(B - \sum_i Q_i(t))$. At time $t = 0$, $Q_k(0) = 0$ and $T(0) = 2B$, thus $Q_k(0) < T(0)$. Packets are allowed to enter into the buffer, and $Q_k(t)$ will increase until $Q_k(t) = T(t) = 2/3B$, as illustrated in Fig. 2. Once $T = Q_k$, the port cannot occupy

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

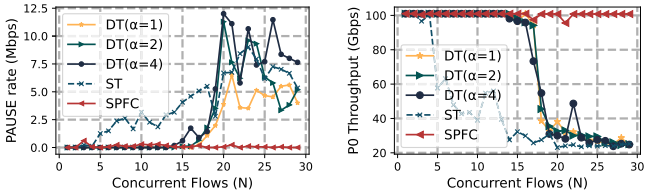
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Compact and typical network scenario. (experiment settings in Section III-A)

(b) Victim flow in Spine-Leaf.

Fig. 3. Victim phenomenon.



(a) PAUSE rate in Port P0.

(b) Throughput in Port P0.

Fig. 4. The pre-experiment results.

an additional buffer and the queue length will no longer increase. The reserved buffer size, in this case, is $B/3$. The α represents the extent to which the activated port occupies the buffer. The larger the α, the larger buffer the activated port occupies and the smaller the remaining buffer.

III. Motivation

In this section, we expose our experimental observations and insights. First, we conduct a case study to explain the victim traffic phenomenon. Then we give our insights into why this phenomenon happens.

A. Victim Traffic Harms Performance

Congestion control algorithms (e.g., DCQCN [34], TIMELY [35]) have been proven to control long flows well. However, when a large number of bursty flows flood into the network unpredictably, the interaction between the flow control mechanism (PFC) and congestion control may cause unexpected problems. The existing traffic in the network may not adjust the sending rate as quickly as possible, resulting in buffer overflow and triggering PFC PAUSE. As a result, those flows will still be affected by the side effects related to PFC, such as congestion spread, victim traffic phenomenon, and even deadlock. Subsequently, we first conduct a careful, fine-grained, and multivariable observation and then infer the root cause of particular symptoms and problems.

We use a topology as illustrated in Fig. 3(a). There are 30 senders (H0-H29), and two receivers (R1-R2) are connected by a 32-port switch. According to the literature [40], it is a compact unit in Fat-Tree [27] or Leaf-Spine [41] like Fig. 3, where S1 sends to R1, S2 sends to R2 and H* sends to R2. In Fig. 3(a), we simulate the end-to-end transmission process in data centers. The links, each with a speed of 100 Gbps, exhibit a propagation delay of 9 μs for the link from H*(0-29) to the switch, and one hop delay of 1 μs for the link from the switch to R*(1, 2). The switch buffer size is 16 MB. The flows are a mixture of long flows and concurrent burst short flows. In detail, we assume H0 starts two long-lived flows to R1 and R2, called F1 and F2, respectively.[1] When the two long-lived flows are stable, each sender of H1-H29 generates a short flow (varies from 1 to 29) to R2 at line rate (i.e., 100 Gbps) simultaneously, and the size of each flow is 64 KB. Since each short flow only lasts for 5.5 μs (<1 RTT), it is uncontrollable by the end-to-end congestion control mechanisms. These uncontrolled burst flows begin at 1 ms and last for about 11 ms in total. We conduct simulations via ns-3 to investigate various buffer management schemes, including DT and Static Threshold (ST). We configure $\alpha = 1, 2, 4$ to test DT, and the PFC threshold for ST is $B/32$ (32 is the port number). For our method, we raise the threshold to the entire buffer size. The host uses DCQCN as the congestion control. All parameters are set to the default values recommended by the related standards, and literature [14], [34], [42], and more details are given in Section VII.

Fig. 4 shows the simulation results. We measure the PAUSE rate (i.e., the rate of transmitting PAUSE messages) and the throughput of port P0 during the burst. As shown in Fig. 4, the PAUSE rate increases with the increase of concurrent burst traffic, while the throughput of port P0 rapidly decreases by up to 80%. F2 is the traffic that actually causes congestion, while F1 is paused by the PFC. In this paper, we define it as the victim traffic phenomenon when a flow (e.g., F1) is paused because of congestion caused by other flows (e.g., F2 and concurrent burst flows). The victim phenomenon significantly reduces the forwarding efficiency of the switch and affects the victim traffic performance. When raising the PFC threshold for P0, it can eliminate the victim phenomenon (SPFC in figures, we simply consider a large dynamic PFC threshold as SPFC to be explored in the pre-experiment.). This is because F2 has enough buffer (time) to converge to the normal rate without affecting F1.

In this case study, we have set F1 to be a long-lived flow for brevity. We note that H0 might also send a mixture of long flows and short flows. As a result, the occurrence of the victim traffic phenomenon further enlarges the tail FCT, which significantly impacts the performance of delay-sensitive short flows (see detailed results in Section VII).

B. Hop-by-Hop Flow Control Contradicts End-to-End Congestion Control

The victim traffic phenomenon results in unexpected throughput loss and enlarged tail FCT. But the root cause of the victim traffic phenomenon requires further understanding of the PFC interaction process. To further understand the PFC interaction process, we now analyze the dynamic behavior of flows in detail, revealing the reasons for the long duration of the congestion tree and the unexpectedly large throughput loss.

[1]F1 can also include delay-sensitive short flows. For brevity, we set F1 to be a long-lived flow in this case study.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

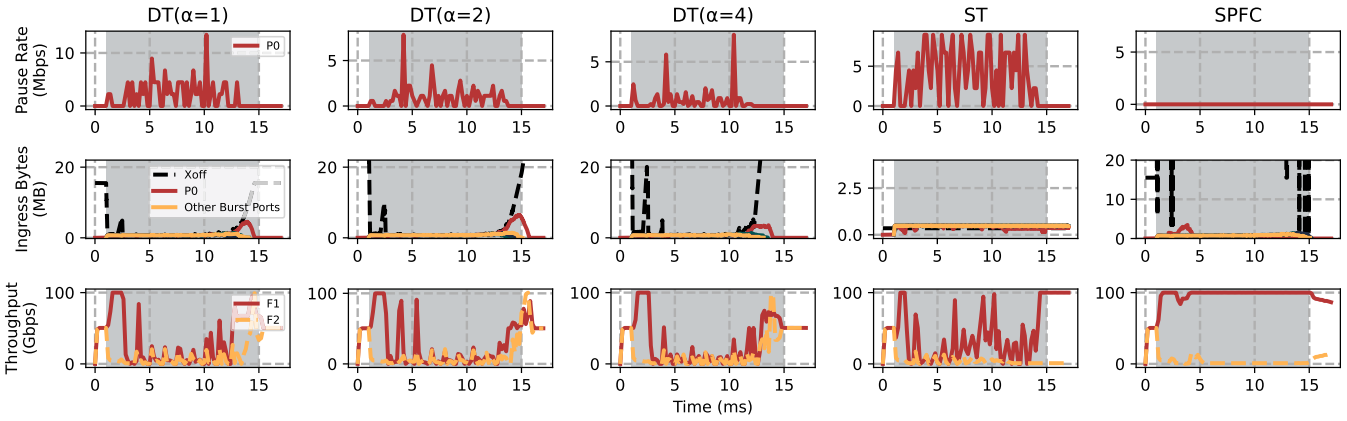


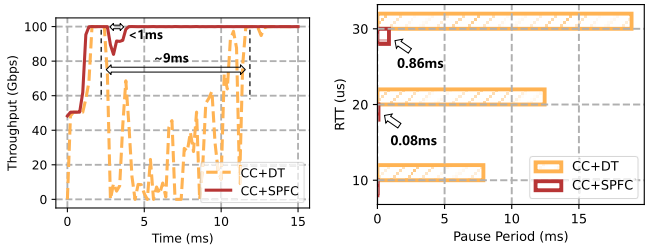
Fig. 5. Interactions between PFC and congestion control (grey areas are during burst).

Fig. 5 shows the sequence diagram where N = 24 (N is the number of burst ports in Fig. 3(a)). In the input port, we measure the PAUSE rate (i.e., the rate of transmitting PAUSE messages), ingress bytes, and the throughput. As shown in the third row of Fig. 5, when PFC is employed, the input port P0 pauses its upstream port H0 to avoid packet drops, and concurrent burst flows congest the port P2. During this process, both the congested flow F2 and non-congested flow F1 face significant throughput loss, whether they are the direct cause of congestion at port P2. Conversely, giving more buffer to P0 ensures keeping the throughput of F1 stable.

During the first 1 ms, the sending rate of F1 and F2 reaches the fair share, i.e., $R_{F1} = R_{F2} = 50$ Gbps. At 1 ms, 24 uncontrolled burst flows started at line rate. The ideal state is that F2, H1-H24 share the bandwidth of receiver R2 on average, which is $R_{F2} = R_{H1} = \ldots = R_{H24} = 100/25 = 4$ Gbps. It is the port P2 rather than its upstream port P0 that is the bottleneck of F2. As a result, F1 can send data at a higher rate, i.e., $R_{F1} = 100 - R_{F2} = 96$ Gbps. However, congestion control requires feedback information and multiple RTTs to adjust the sending rate (one RTT slows down by up to 1/2 the current speed). Excess data will need to be stored in the switch buffer.

As shown in the second line of Fig. 5, the DT's X_{off} threshold begins to decrease as the other burst ports and P0 begin to accumulate buffers. These ports start sending PAUSEs when the X_{off} equals the queue length. When port P0 leaves no buffer to store, P0 will send PFC PAUSE to H0. As a result, H0 becomes the bottleneck, and F1 turns out to be the victim. That is, although F1 does not cause congestion, its sending rate is far below the ideal rate (e.g., 96 Gbps), shown in the third line of Fig. 5. The PFC's congestion tree will continue to pass. If H0 is also a switch at this point, it may continue to send PAUSEs to its upstream nodes, which further results in congestion spreading. What's more, from the third line of Fig. 5, DT algorithm treats each port equally, so burst ports are also allocated the same amount of buffer. DT algorithm degrades into ST algorithm during burst. And there is no difference in DT between different α.

Based on these observations, we infer that the end-to-end congestion control needs buffer/time to adjust to the target sending rate when a burst occurs. Still, the buffers are overrun with burst traffic, causing avoidable PFC PAUSEs. In other words, the fast operations of hop-by-hop PFC flow control mismatch the evolution-based rate adjustment of end-to-end congestion control. This mismatch fundamentally results in the victim traffic phenomenon.



(a) The time needed for recovering F1. (b) Pause duration under different end-to-end RTTs.

Fig. 6. Mismatch between port control and congestion control (CC for short).

Let's discuss more about the "mismatch problem". Assuming that the time for F2 to be completely controlled by the end-to-end congestion control from the arrival of the burst is denoted as t. During this period, the accumulated F2's packets at port P0 can be approximated as $S' = (FlowRate - C/n) * t$ ($FlowRate$ is decreasing). The PFC PAUSE threshold provided by DT algorithm is T. Due to the mismatch between instantaneous port rate control and the long time required for congestion control, F2's packets accumulate rapidly, resulting in $S' > T$. Consequently, the upstream port of P0 is paused, affecting the throughput of the victim flow F1. But if we increase the PFC threshold for port P0 to T', ensuring that $S' < T'$, the port can be kept from triggering PAUSE or trigger as few PAUSEs as possible. Then the victim traffic would not be unfairly treated.

To explain this more clearly, we further extend the duration of the burst and repeat the above experiment with N = 20. From Fig. 6(a), it can be seen that when congestion control (DCQCN as default) with DT ($\alpha = 2$)[2] is used, the throughput of F1 takes nearly 9 ms to recover. However, if more buffer

[2]If not specified, DT defaults to $\alpha = 2$ setting afterwards.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

is allocated to port P0 (taken by SPFC), F1 can recover in less than 1 ms and its throughput is not significantly affected during this period. As the end-to-end RTT increases, flows controlled by congestion control would take longer to recover and the port length easily reaches the PFC PAUSE threshold. Due to insufficient buffer to accommodate the accumulated packets, the duration for pausing the upstream port also increases, as shown in Fig. 6(b). Therefore, the victim port needs sufficient buffer (time) to match the reaction time of congestion control, which reduces PFC PAUSE and maintains the performance of victim flows.

Solving this problem requires a buffer management policy to determine whether a switch port transmits long-lived CC-controlled flows or short bursty uncontrolled flows. Consequently, rethinking the way of buffer management that eliminates the mismatch between congestion control and flow control would be a relevant contribution. We will elaborate the design next.

IV. Design of SPFC

In this section, we developed a new buffer management approach called Selective-PFC (SPFC), a simple but effective buffering strategy at each input port, easily implemented in switches. The critical factor that inspires the new buffer management approach is that if we could give more buffer for CC-controlled traffic to allow them to adjust their sending rate, we would reduce unnecessary PFC PAUSEs (see SPFC's performance preview in the first row of Fig. 5). The goal of SPFC is to eliminate the victim traffic phenomenon, congestion spreading, and even deadlock. Therefore, SPFC needs to identify ports in different states and then configure different buffer management policies according to the states.

A. An Overview of SPFC

The victim traffic includes not only long flows but also short flows destined for other output ports. According to the previous analysis, they should not be paused for transmission. We refer to the input port through which the victim traffic passes as the "victim port". On the one hand, we need to provide sufficient buffer for this type of port. On the other hand, we cannot indefinitely squeeze the resources of other ports. Therefore, when the victim traffic phenomenon occurs, we allocate as much buffer as possible to the victim port of the switch, while the other ports continue to use the DT algorithm.

This is not a permanent state. In general, they share the buffer and use the same DT buffer management. Moreover, the duration of the victim phenomenon is brief. Once the congestion ends, the victim port will return to normal status.

We still take Fig. 4(a) as an example. P0 acts as the victim port and may contain not only F1 but also other short latency-sensitive flows. When the victim phenomenon occurs, the PFC pause threshold of port P0 is raised, making the victim flow less likely to be paused. This provides controlled time for the normal flow F2, thereby reducing the impact on the queuing of port P0. Although allocating more buffer can eliminate this phenomenon, it can be difficult for switches if there are numerous ports similar to P0, due to limited memory. Fortunately, in data centers, there are relatively few uncontrollable long-lived flows (like F2 in Fig. 3(a)) [28], [43], so switches have the ability to provide more buffer to a limited number of victim ports. Ultimately, P0 and other burst ports will all revert to the same state and adopt the DT buffer strategy.

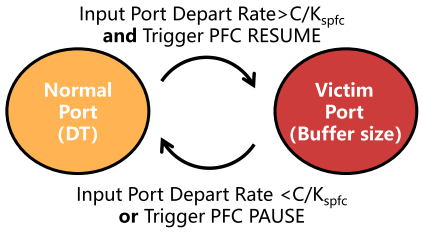


Fig. 7. State transition diagram of SPFC in each port.

The key to our solution is how to identify the victim port, assuming that it contains long-lived controlled flows, to adopt the correct buffer strategy. Next, we'll detail our definitions of the two port states, the different actions, and the transitions between the two states.

B. Port States and Actions

1) Two Different States: As mentioned in Section II-C, PFC-enabled shared memory switches share buffer across input ports, so SPFC works on each input port. Based on our observations in Section III, we define two states of switch input ports in lossless networks as follows:

•**Victim state**: This port has potential victim traffic. Once some of these flows share a bottleneck with the burst flows, it can cause that port to pause the upstream, affecting other victim traffic. For instance, in Fig 3(a), port P0 is the victim port, F1 is the victim flow and F2 shares the P2 with the burst flows.

•**Normal state**: There are two cases of this state. The first is that this port has no victim traffic. The traffic on this port contributes to congestion, and of course, this port should send PFC PAUSE. In Fig 3(a), ports H1-H29 are the normal port, and all the burst flows contribute to the congestion on P2. The second is that this port was previously a victim port, but because it doesn't need more buffer or it can't take up too much buffer, it is just like any other normal port. The second case will be discussed in more detail in the next section.

2) Two Different Actions: The SPFC needs to give more buffer to the CC-controlled traffic, and then when rate mismatches occur, congestion control can be converged to avoid unnecessary PFC. But on the switch, it is difficult to do per-flow operations, and it is difficult to identify whether there is controlled traffic in a port. So, the SPFC assumes that all victim ports may have long-lived controlled traffic and the PFC threshold of the victim port is set to a large value. Once the congested traffic occupies too much of the buffer and triggers the PFC PAUSEs, the SPFC turns this port to the normal port state and returns the port's threshold to the normal value.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Based on the above analysis, SPFC performs the following actions according to different states:

•**Victim Port Action**: The PFC threshold for this port is set to the buffer size. This means that the victim ports can use all buffers. But when the shared buffer is full, the victim ports still trigger the PFC PAUSE. When there is no burst flow, raising the PFC threshold for the victim ports has no effect. Long-lived flows using congestion control can make the queue smaller than the normal DT threshold. PFC will not be triggered because the queue length is far below the threshold. A victim port occupies more buffer if and only if the rate mismatch occurs.

•**Normal Port Action**: Normal Port uses the normal DT algorithm. When the victim port occupies more buffer, according to the DT algorithm (1), the corresponding PFC threshold of normal ports becomes smaller.

In this way, SPFC achieved what we expected, providing more opportunities to occupy buffer for the long-lived flow and reducing other port buffer usage.

3) State Transitions: To distinguish between these two different input port states, SPFC needs to use some input port information. Assume the input port departure rate is the output speed of the input port, which is:

$$R_{depart} = R_{input} - \triangle Q_{input} \tag{2}$$

It is the rate at which the input port processes packets and is also the throughput of each flow that passes through the input port. Our key finding is that a victim port has a high departure rate because the port has other victim flows in addition to the congested traffic. A normal port has no additional flow, so it has a low departure rate. Therefore, we can rely on a selected rate C/K_{spfc} (assume that the port line rate is C) to distinguish between the victim port and the normal port. The K_{spfc} is a key value used to distinguish victim port states. We will discuss the details later.

Fig 7 illustrates state transitions among two states in switch ports:

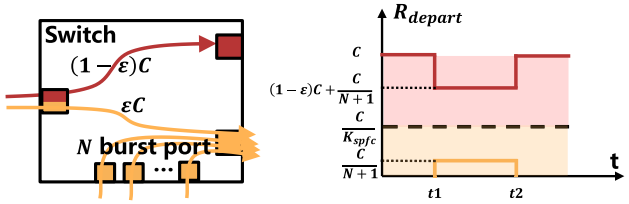
•**When $R_{depart} > C/K_{spfc}$ and this port triggers PFC RESUME,** the port is the victim port and the PFC threshold is the entire buffer size.

•**When $R_{depart} < C/K_{spfc}$ or this port triggers PFC PAUSE,** the port is the normal port and the PFC threshold is controlled by DT.

For example, as shown in Fig. 3(a), when the burst does not occur, the departure rate of port P0 is 100 Gbps. This indicates that the port is sending at a very high rate, and the possibility of the victim port is very high. When a burst occurs, the output speed of H1-H29 ports is only about 4 Gbps, so H1-H29 is experiencing a burst.

It's worth noting that a port's state is shaped not just by its rate but also by the length of its input queue. Suppose a previously victim port returns to normal state due to PFC PAUSE, its queue length would gradually decrease. After the length falls below X_{on}, the port triggers PFC RESUME and then reverts to the victim state when its R_{depart} remains high.

The above analysis shows that the foundation of SPFC is the K_{spfc}. It distinguishes whether the port deserves a larger PFC threshold. The crucial problem is to determine the value of K_{spfc}. In the next subsection, we will give an example of how to use a flow model to select the value of K_{spfc}. We also analyzed which cases were misidentified in the port state and how to remedy them.



(a) Conceptual flow distribution in the switch. (b) Flow model of R_{depar}

Fig. 8. SPFC flow model.

TABLE I
PARAMETERS OF THE FLOW MODEL

Paramter	Definition
C	Link capacity.
ϵ	Ratio of congested traffic to all the victim port traffic.
N	Number of burst ports.
K_{spfc}	Threshold of the SPFC port for identifying the victim port.
R_{depart}^{burst}	Departure rate of the burst port.
R_{victim}^{burst}	Departure rate of the victim port.

C. Flow Model

We build a conceptual flow model to describe K_{spfc} in lossless networks. Fig. 8(a) shows traffic distribution. The traffic of the victim port (left) is divided into two parts: the victim traffic in red with a traffic rate of $(1-\epsilon)C$, and the congestion traffic in yellow, with a traffic rate of ϵC. At time $t1$, N burst ports send uncontrolled traffic to congested ports and end at time $t2$. It is assumed that all traffic does not change the sending rate for a short period of time, and that the switch's buffer is large enough not to trigger PFC. Key parameters are listed in Table I.

We have built a fluid model to analyze K_{spfc}, seen in Appendix A. From it, K_{spfc} is determined by N and ϵ. We conclude that N is independent of ϵ because of the uncertainty of the flow distribution, and K_{spfc} is unbounded. However, a reasonable K_{spfc} can adapt to most cases, which is enough for switches to distinguish between the victim port and the normal port. Fig. 9 demonstrates the values of K_{spfc}. We take the 32-port switch as an example in this paper. The value of ϵ ranges from 0 to 1. The oblique plane at the top is $f(N,\epsilon) = N+1$, and the surface at the bottom is $f(N,\epsilon) = (N+1)/((1-\epsilon)(N+1)+1)$. The z-value of the flat plane is $K_{spfc} = 5$. We empirically recommend K_{spfc} is 5 because it can already cover most solutions.

There are two parts that SPFC can't cover. First, when $N < 4$, SPFC identifies the burst port as the victim port. In this case, the burst was so small that a mismatch would require a small buffer that would not affect the original victim traffic. Even when burst ports start to trigger PFC, they will return to normal

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

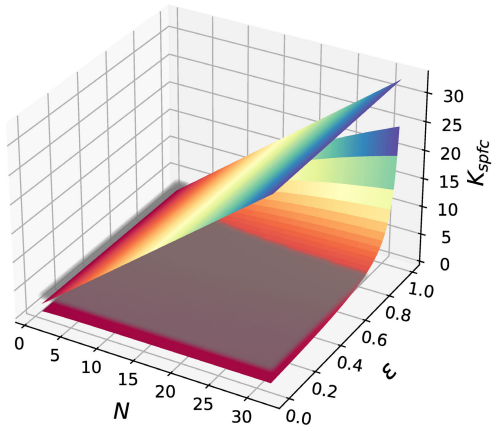


Fig. 9. The relationship between burst port number N, ϵ and K_{spfc}. The z-value of the flat plane is $K_{spfc} = 5$.

without affecting judgment. Second, when $\epsilon > 0.8{+}1/(N{+}1)$, SPFC identifies the victim port as the burst port. Note that as ϵ increases, the ratio of victim traffic to all the victim port traffic decreases. In this case, it is reasonable to identify the port as a normal port. And we proved through simulation that SPFC can also slightly outperform DT in the case of victim traffic. The recommended setting is also verified through simulations in Section VII-A.

D. Analysis of SPFC

PFC threshold and queue length: Although we significantly increase the PFC threshold when the victim phenomenon occurs, the impact on normal ports is temporary. Assume that the switch buffer size is B and there are n victimized ports and m normal ports at a given moment. The queue lengths of the victim ports are $Q_1, Q_2, \ldots, Q_n$, and the queue lengths of the normal ports are $q_1, q_2, \ldots, q_m$. Then the total buffer occupied by the victim and normal ports are:

$$Q_{victim} = \sum_{i=1}^{n} Q_i \qquad q_{normal} = \sum_{i=j}^{m} q_j \tag{3}$$

So the PFC threshold for normal ports is:

$$T_{normal} = \alpha * (B - Q_{victim} - q_{normal}) \tag{4}$$

Typically, the victim port requires one RTT to control the flow competing with the burst, like F2 in Fig. 3(a). As a result, the victim port length Q_i will decrease and the normal port length q_i will be raised. When reaching the final state, they all have the same PFC threshold complying with Eqn (4).

Long-lived flow start after burst: In the preceding analysis, our findings and the flow model are based on the assumption that a long-lived flow already enters the stable state when burst flows start. It is not clear what happens when a long-lived flow starts after there are already a large number of burst flows in the network. Essentially, the order in which long flows and bursts arrive does not bring any additional negative impact. In the latter scenario, there are no victim ports as the traffic on these ports does not compete with the burst traffic. The port still uses the DT algorithm at this point. When a long-lived flow arrives and competes for the same output port as the burst, it may cause other traffic on the input port to become victims. In this case, we need to increase the PFC threshold of the input port to protect the victim traffic. Therefore, the order in which long-lived flows arrive does not conflict with our previous analysis.

Different victim patterns: The victim flows can be either small flows or large flows. When they are small flows, the victim traffic phenomenon significantly impacts delay-sensitive small flows. When the victim flow is a large one, a transient burst has a relatively small impact on it. However, bursts occur frequently in data centers [2], [44] and the victim's long flow would suffer from long-term unfairness, leading to performance degradation. To thoroughly demonstrate the effectiveness of our proposed method, we generate a mixed traffic pattern at port P0, comprising both large and small victim flows with various sizes (as described in Section VII-C). Fig. 13 presents the robustness of SPFC under different traffic patterns.

E. Putting It All Together

To demonstrate the advantages of SPFC, we enable SPFC and repeat the simulations in section III, and the results are also inserted into Fig. 4 and Fig. 5. The results in the last column of Fig. 5 tell that PAUSEs in P0 are completely avoided. The results also confirm that SPFC can help the uncongested flows grab idle bandwidth quickly and regulate the congested flows to proper rates correctly and promptly. Next, we will explain how SPFC works.

At first, every port is in the normal state, and there is no traffic on the switch at this time. When H0 starts sending data, the P0 port detects a speed of 100 Gbps and changes the port into the victim state. Port P0 does not have queues building up before burst because there is no bottleneck in its transmission. When burst comes, the queues of P0 and H1-H29 begin to grow until the entire buffer is full. When the buffer is nearly full, the PFC threshold for H1-H29 is triggered, and PFC PAUSE is initiated to notify upstream ports to suspend transmission. Since port P0 is in the victim state, its PFC threshold is larger than the other ports, so it will continue to buffer packets. According to the DT, all PFC thresholds of H1-H29 are lowered due to the P0 queue increase. All the buffer saved by H1-H29 was reserved for P0 to absorb the mismatch traffic. When the congestion control rate of F2 is adjusted correctly, the queue length of P0 can be controlled to a smaller queue length than the normal DT threshold, and the queue length of H1-H29 ports also returns to the normal DT threshold. During this process, there is no PFC PAUSE in port P0, so F1 also quickly adjusts the speed to the correct value through congestion control.

V. Implementation

In this section, we present a possible implementation of traffic-aware buffer management policies, Selective-PFC (SPFC). Based on the classic dynamic threshold (DT) [18], SPFC controls port thresholds by assigning a traffic state indicator to each port and detecting traffic changes.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

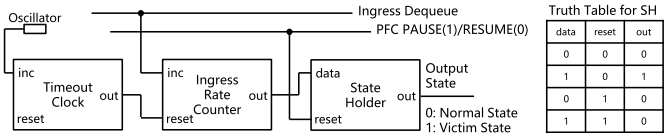


Fig. 10. Circuit diagram.

A. State Decision

An important objective is to design a buffer scheme that is simple to implement. Fig. 10 illustrates the circuit diagram of the SPFC added to each input port. Only a few counters and triggers need to be maintained to monitor the state of the ports. Inputs of this diagram are the ingress dequeue signal and PFC PAUSE signal generated by each logic input queue of the port. A pulse is generated on them whenever a packet is enqueued or dequeued, and a PFC PAUSE frame is triggered. The output in this diagram determines whether SPFC is in the victim port state. Timeout Clock (TC) is a countdown timer, and they begin to count down from their default values once they are enabled. Ingress Rate Counter (IRC) increases its value for every input pulse. State Holder (SH) holds its state until reset by reset pulse. Next, we'll show the details of these timers and counters one by one.

Timeout Clock (TC) is used to fix a period of time to calculate the departure rate of each input port. It begins to count down when Oscillator triggers a pulse signal and stops when its value reaches 0. When TC is counting down, its output pin is set to 0. And when it reaches 0, its output pin is set to 1 to signal IRC to reset its counter to calculate the next period rate. The default value of the TC should be long enough to keep the rate calculation stable and prevent the rate from fluctuating too much. But it can't be too long, so the rate calculation is not timely.

Ingress Rate Counter (IRC) is used for identifying that the input port becomes the victim state. It increases for each pulse on the ingress dequeue signal. Therefore, in combination with the TC, its value represents the packet number over a period of time. When it does not exceed its counting number, its' output is 0, indicating that the current rate is insufficient and SPFC is in the normal state. SPFC will change into the victim state when it reaches its counting number, and its output turns to 1. The counting number influences the sensitivity of identifying the victim state.

State Holder (SH) is used for holding the status of IRC temporarily. Because every time a TC is re-counted, the IRC will be cleared to zero, and the output will be 0. Therefore, SH is to hold the state in the last time interval, making the change of state stable. In addition, it has the very important function of penalizing ports that have given sufficient buffers but still trigger PFC PAUSE. When PFC PAUSE is triggered, the reset button is activated, making the port normal and running DT algorithm. During this interval, the port output remains 0 regardless of the IRC's output. The IRC output becomes effective only after receiving PFC RESUME and the reset becomes 0. The truth table in Fig. 10 concisely illustrates the logic, consistent with the description in Section IV-B.

B. Convenient Parameter Setting

In SPFC, only three parameters (TC, IRC, and α) need tuning before deployment. We assume that the line rate C of all ports is equal in the same switch for simplicity. The switch has a buffer size of B.

TC and IRC together make victim state decisions; thus, their thresholds should be set together. TC controls the time interval for measuring the rate; thus, a smaller TC threshold means SPFC is more sensitive to rate changes, and a larger TC would mean otherwise. In practice, $TC \geq 4RTT$ is good enough for most cases. IRC is the key to determining state. We set the IRC threshold as $(C \times TC)/K_{spfc}$. $(C \times TC)$ is the maximum number of packets that can be exported from the port in the interval of TC. The K_{spfc} setting has been discussed in detail in Section IV-C. The other parameter is α, which is the parameter of the DT algorithm under the normal port. It's a tradeoff, just like a normal DT algorithm. α value is large, which means that the active port can occupy more buffer. Small α means otherwise.

VI. Discussion

Implementation requirement: SPFC can be implemented directly on the data plane, which needs a little switch modification. Compared to the traditional DT method in switches, the SPFC only requires one more counter per port and several more lines of logic. The space and computing complexities of modification are both O(1).

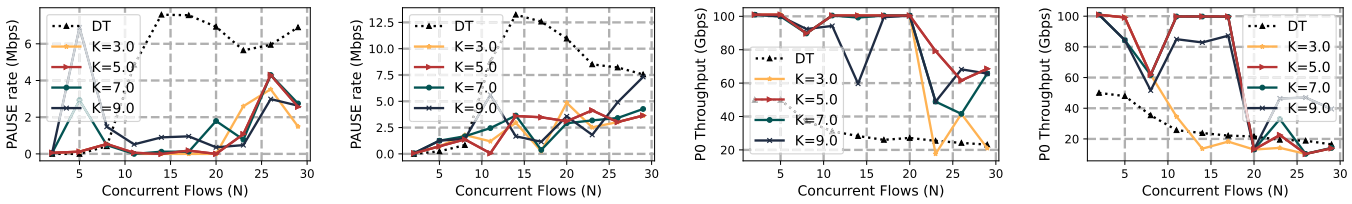
Temporary unfairness: SPFC essentially uses the buffers of burst ports to provide more buffers for non-burst ports. In this case, non-burst ports can be adjusted to the correct speed by congestion control without causing performance penalty problems. This creates unfairness in the port, but Fig. 5 shows that the unfairness is temporary and will soon be recovered. By the way, more emphasis is placed on the tail FCT of services in the data center, and SPFC can greatly reduce the tail FCT by controlling the victim traffic phenomena.

Low latency requirements: SPFC makes the burst port use the buffer less, in which case the PFC-induced port latency might cause some flows to miss their deadline. First, the victim traffic phenomenon that SPFC wants to address doesn't happen very often. SPFC is only effective when the victim traffic phenomenon occurs, causing temporary unfairness. Second, suppose it is necessary to ensure low latency for important flows strictly [45], [46]. In that case, we recommend raising the priority of these flows so that they are not affected by SPFC regulation.

Performance across diverse traffic scenarios: Based on the flow model in Section IV-C, we have inferred that SPFC can cover most cases under $K_{spfc} = 5$. In a few scenarios, SPFC may also make incorrect judgments regarding the port state (i.e., when $N < 4$ or $\epsilon > 0.8 + 1/(N+1)$). We have illustrated that the impact of such misunderstandings is negligible. In addition, we have also conducted experiments to

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) PAUSE rate when $\epsilon = 0.75$. (b) PAUSE rate when $\epsilon = 0.875$. (c) Throughput when $\epsilon = 0.75$ (d) Throughput when $\epsilon = 0.875$

Fig. 11. SPFC with $K_{spfc} = 5$ can provide better robustness when meeting different number of burst flows. [topology in Fig. 3(a)].

investigate the performance of SPFC when facing different combinations of traffic. Both victim and burst ports may contain complex traffic patterns, like a mixture of large and small flows. We generated test traffic based on the WebSearch workload [3] for these two types of ports and Fig. 13 demonstrates the robustness of SPFC under $K_{spfc} = 5$. In all-to-all communication, we have also done testing under the Hadoop workload [28]. For more traffic scenarios, further exploration is required and we leave it as future work.

VII. Evaluation

In this section, we compare the performances of DT, ST and SPFC on ns-3 platform [22] and DPDK [23] testbed.

We evaluate three critical aspects of SPFC:

(1) We verify the parameter selection by simulation and verify the correctness of simulation by testbed experiment.

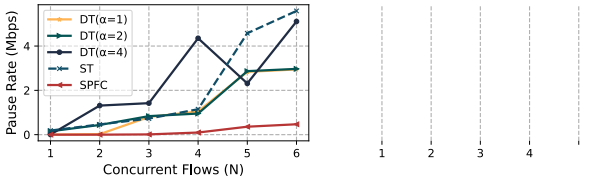
(2) We evaluated the flow scalability of SPFC in the Burst Tolerance scenario, including PAUSE rate, FCT of different traffic sizes, and PFC of different ports.

(3) We measured the performance of SPFC in large-scale all-to-all scenarios and investigated its utility in mitigating deadlock.

A. Parameter Validation of K_{spfc}

The first step is to verify that the selection of K_{spfc}, the most important parameter of SPFC, is appropriate. As mentioned earlier, the phenomenon of victim flows often occurs due to bursty traffic. In order to create victim flows more effectively, we have introduced competition between normal flows and bursts by increasing their intensity. We use the topology shown in Fig. 3(a). H0 sends 1 long flow to R1, and 3 ($\epsilon = 0.75$) and 7 ($\epsilon = 0.875$) long flows to R2, respectively. To validate the theoretical analysis presented in IV-C, we conducted tests on the PFC pause rate and throughput under different values of K_{spfc}. The classical buffer scheme DT is used for comparison. Fig 11 shows the experimental results.

From the result, a larger K_{spfc} (e.g., 7 and 9) will result in a larger pause rate when the number of concurrent requests is small. This is because when the concurrent traffic is few, the phenomenon of victim flows is not obvious, and normal ports may be mistaken for victim ports. As a result, more victim ports compete for buffer resources, which can easily trigger PFC. In contrast, a smaller K_{spfc} (e.g., 3) will not guarantee a good throughput when the number of concurrent requests is large. So $K_{spfc} = 5$ is recommended as a good choice.



(a) PAUSE rate in Port P0.

Fig. 12. Result of Testbed Validation.

In addition, in section IV-C, we analyzed that when $\epsilon > 0.8 + 1/(N+1)$, SPFC identifies the victim port as the busrt port. However, fig 11(b) and (d) show that when $K_{spfc} = 5$, SPFC still performs better than DT. Even if SPFC misidentifies the state of a port, it is only temporary because N will dynamically change with the burstiness of traffic.

B. Testbed Validation

In this subsection, we verify the basic function of SPFC using simple synthetic microbenchmarks and repeat the experimental results of Section III-A on the testbed.

Testbed setup: We deploy SPFC on a testbed built upon DELL servers with two Intel 82599 NICs and four Intel 850nm SFP+ laser ports for optical fiber connections, acting as a PFC-enable switch. Each NIC has two 10Gbps Ethernet ports, and the server is equipped with Intel i7-9700K CPUs (8 cores, 3.60GHz). We configure 8GB huge page memory for DPDK (4GB/NUMA Node). We bind 5 threads on 5 physical cores for analyzing traffic and adjusting port state. Due to the lack of physical ports, we virtualize 5 network ports in the DPDK kernel. Thus, the server can work as a nine-port switch. We implement PFC according to IEEE 802.1Qbb [16]. Both PFC and SPFC are based on the reference test-pipeline project [47]. Four hosts are equipped with Mellanox RDMA 10Gbps network adapters, two with CX5 [48] and two with CX6 [49]. They are all connected to the PFC-enable switch. PFC-support service is enabled to process received PFC frames.

Scenario: The network topology is similar to Fig. 3(a), where the hosts linked to the four physical ports are victim port H0, receiver port R1, R2, and burst port H1, respectively. The DPDK switch will make 5 copies of the traffic from the H1 port to virtualize additional ports. Packets are forwarded to the corresponding egress ports based on their destination addresses. The total buffer of the switch is 2.5MB and the

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

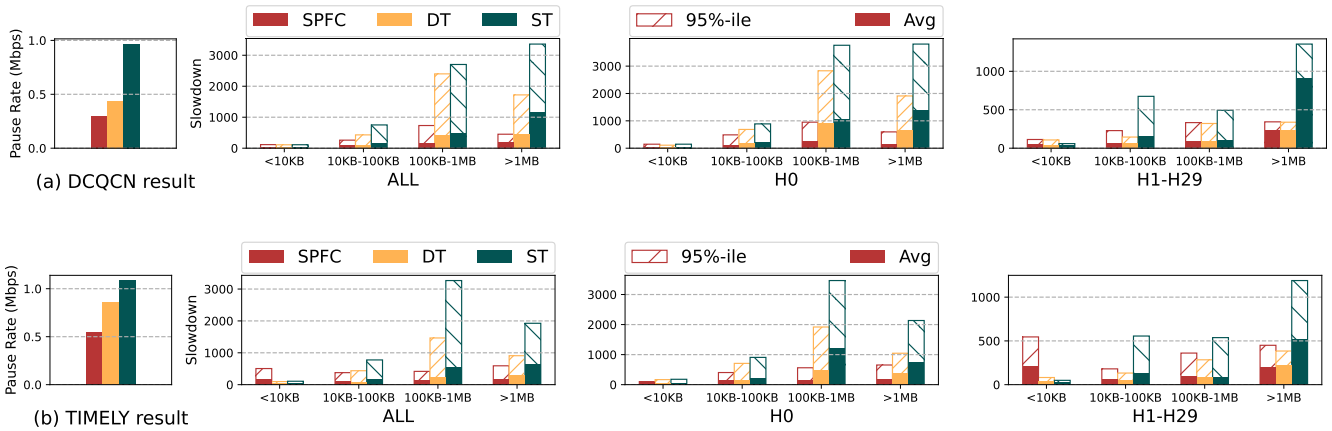


Fig. 13. Compared with DT and ST, SPFC can also perform well in PAUSE rate and FCT even under a mixture of traffic patterns. [topology in Fig. 3(a)].

K_{spfc} is set to 5. We used DT and ST as the comparison schemes. At the end host, DCQCN is used as the congestion control with default parameters. We use Perftest [50] to generate traffic.

The measurement results are shown in Fig 12. The testbed experiment well verifies the simulation results. With the increase of burst ports, SPFC does not cause the victim port to trigger too many PFC packets, and the throughput of the victim port does not decrease. In contrast, ST performs the worst because it cannot dynamically allocate resources among different ports.

C. Burst Tolerance

One advantage of SPFC is its robustness against PFC triggers caused by concurrent burst flows. Next, we also use the basic scenario in Fig. 3(a) to evaluate SPFC in the typical victim traffic scenario. The bandwidth and latency of the links remain consistent with the specifications outlined in Section III-A. Hosts H0-H29 generate flows according to the heavy-tailed WebSearch workload [3] with exponentially distributed inter-arrival time. Especially, the workload generators at hosts H1-H29 are set to be synchronous to simulate concurrent bursts. The target load at the bottleneck links is set to 0.6 (0.4 for H0 and 0.2 for burst). The switch memory is 16 MB, and the α for DT and SPFC is set to 1. DCQCN and TIMELY are set to the default values recommended by the related literature [34], [35]. We measure the pause rate and flow completion time (FCT) of SPFC and compare them with DT and ST.

Performance Results: The left subgraph in Fig. 13 shows the generating rate of PFC PAUSEs. SFPC triggers the fewest PAUSEs. It can reduce 31.6% and 69.0% PFC PAUSEs compared to DT and ST with DCQCN, 36.5%, and 49.7% with TIMELY, respectively. And the average and 95th percentile FCT slowdowns from different hosts are drawn in the right subgraph in Fig. 13, where the slowdown is the ratio of the actual time required to complete an echo RPC divided by the best possible time for an RPC of that size on an unloaded network. A lower slowdown value indicates better performance. A slowdown of 1 is ideal. Slowdowns for small flows and large flows are independent and cannot be compared with each other. The solid bar at the bottom indicates the average FCT slowdown, and the upper stripe bar shows the 95th percentile value. Clearly, SPFC performs better than DT for all kinds of flows in general.

TABLE II
FLOW SIZE DISTRIBUTION OF REALISTIC WORKLOADS

Flow Size	% of number		% of traffic	
	W1	W2	W1	W2
0KB-10KB	15.0	70.26	0.04	0.47
10KB-100KB	39.16	18.24	0.98	7.4
100KB-1MB	15.83	9.0	4.01	23.45
1MB-	27.0	2.5	94.97	68.68

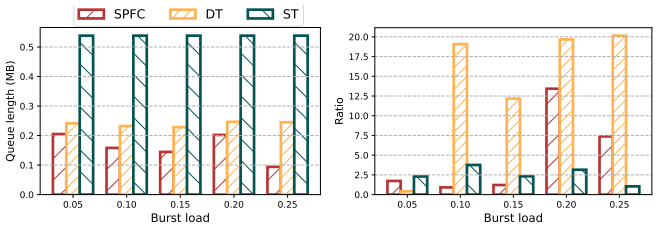
W1: WebSearch at Microsoft [3].
W2: Hadoop at Facebook [28].

As mentioned earlier, the SPFC actually uses the H1-H29 buffer for the P0 buffer during mismatch, so the FCT slowdown of H1-H29 is increased a little in some cases, but the FCT slowdown of H0 is reduced clearly compared with DT and ST. For all ports, SPFC achieves lower 95th FCT slowdown compared to DT and ST. Specifically, when adopting DCQCN, for small flows (<10KB), SPFC provides similar FCT slowdown with DT and ST. But for flows (10KB-1MB), SPFC outperforms the other two schemes. For large flows (>1MB), SPFC reduces the average slowdown by 57.9% compared to DT and by 83.5% compared to ST. Furthermore, under the TIMELY's control, the performance improvement of SPFC is similar with DCQCN's, except for small flows (<10KB). The reason is that incast flows from H1-H29 ports could not obtain enough buffer, prolonging the overall latency for small ones. While for large flows, SPFC reduces the average slowdown by 39.4% compared to DT and by 70.9% compared to ST. For port H1-H29, SPFC only has obvious FCT increase on small flows (<10KB) due to inadequate buffer. But its impact is limited, and we can improve the overall performance of other kinds of flows.

Fined-grained Analysis: We measured the average length of burst ports when PFC pause frames are triggered. Fig. 14(a) demonstrates the queue lengths of ports under different burst load. In SPFC switches, burst ports pioneer in triggering PFC, which keep the queue length at a lower level. Although DT also reduces the PFC threshold, it provides longer queues,

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Burst queue length. (b) Ratio of buffer occupation.

Fig. 14. (a) The queue length of burst ports is lowest under SPFC. (b) The ratio of buffer occupancy, between victim ports and normal ports, demonstrates that the victim port do not excessively preempt resource under SPFC when the burst load is lower.

which increases packet queuing delay. As mentioned in section IV-D, victim ports do not excessively occupy the resources, even though they are provided with a higher threshold. Fig. 14(b) shows the ratio of the average cache occupancy of victim ports to that of normal ports when triggering PFC. The logic for calculating this ratio is as follows: assume that the victim port triggers PFC x times, and the buffer occupancy of the port is $V_1, V_2, \ldots, V_x$ each time PFC is triggered. The normal port triggers PFC y times, and the buffer occupancy is $N_1, N_2, \ldots, N_y$. Then the ratio is $\frac{\sum_{i=1}^{x} V_i}{\sum_{i=1}^{y} N_i}$. The SPFC-enabled switch is significantly lower than DT's. Overall, it maintains this value within a lower range under low burst load. Although the buffer occupancy of victim ports under SPFC control increases rapidly under high burst load, it remains lower than that of DT. ST sets the same static threshold for each port, so the victim port do not excessively occupy the buffer. Through the analysis of queue lengths, we explain the reason for low latency and demonstrate the protective for buffer of normal ports.

D. Performance in Large-Scale Networks

In this subsection, we evaluate the performance of SPFC in multi-hop datacenters to demonstrate the universality of SPFC. We still use DCQCN and TIMELY as the congestion control at the end host with the default parameter settings. **Normal Pods:** We consider an 8-pod clos network. Each pod consists of 2 Leafs, 4 ToRs, and 64 hosts (16 in each rack), and communicates with other pods through 4 spines. The link capability is 100Gbps below ToRs and 400Gbps above them. All links have a 2μs propagation delay, which gives a 24μs maximum base RTT. The switch memory is 32 MB, and the α for SPFC is set to 2. To support multi-path capability, the Equal Cost Multi Path (ECMP) scheme is used. We use widely accepted and public available data center traffic traces, WebSearch (W1) [3] and FB Hadoop (W2) [28]. We generate flows with exponentially distributed inter-arrival time and the target loads for ToR down-links range from 10% to 90%. The source and destination of each flow are arbitrarily selected.

Fig. 15 and Fig. 16 present the FCT results for all-to-all traffic under W1 workload. SPFC performs better than DT and ST for DCQCN and TIMELY. We obtained similar results in W2, and do not repeat the presentation.

Large flows: When the load is low, there is not much difference between SPFC and the other two schemes. This is because fewer flows result in fewer occurrences of victim flows. However, when the load exceeds 0.5, the advantages of SPFC start to become prominent. In terms of average latency, under DCQCN's control, SPFC achieves an improvement of up to 46.1% compared to DT and 53.8% compared to ST (139.8 for SPFC, 259.4 for DT, and 302.9 for ST). Under TIMELY's control, SPFC achieves an improvement of up to 32.3% compared to DT and 38.9% compared to ST (176.8 for SPFC, 261.3 for DT, and 289.3 for ST). As the load increases, incast occurrences become more frequent, which leads to more large victim flows.

TABLE III
AVG THROUGHPUT OF LARGE FLOWS (UNIT: GBPS)

	SPFC	DT1	DT2	DT4	ST
W1/DCQCN	33.92	26.28	26.49	27.44	26.80
W1/TIMELY	26.63	22.63	22.81	23.01	22.85
W2/DCQCN	27.65	18.61	19.91	21.35	12.50
W2/TIMELY	23.69	13.14	13.99	14.19	10.66

Small flows and Overall: Under DCQCN's control, SPFC provides similar performance to DT and ST in terms of average latency. Only under higher loads does SPFC exhibit a slight advantage. Because many small flows passing through victim ports obtain more buffer. Under TIMELY's control, the 99th percentile FCT slowdown for some flows is lower. It indicates that SPFC has the capability to ensure high throughput.

From the above results, SPFC can timely alleviate the problem of congestion control mismatch by applying more buffer, which reduces the impact on victim flows from the same port. ST has the worst performance because it does not use the buffer well. Although DT can dynamically allocate buffer among different ports, it fails to protect victim ports, causing innocent traffic to suffer PFC pause.

We have already demonstrated that SPFC provides good performance through latency analysis. In data centers, throughput of large flows is also an important performance metric. We tested the average throughput of each large flow under different buffer schemes in the above scenario under 50% load. From Table III, the results achieved by SPFC are higher than the other two schemes'. This is because ports containing large victim flows have high rate, and SPFC increases the PFC threshold to avoid innocent flows being affected. Under the same control strategy, the throughput achieved with the WebSearch workload is higher than that achieved with Hadoop. Due to more large flows contained in WebSearch, it is easier to achieve higher throughput.

Deadlock Pod: PFC is a significant cause of deadlock, and we demonstrate it through the following experiment the performance of SPFC in a deadlock scenario. We adopt the topology in Fig. 17(a), which is a single pod of the above's topology. The links L0-T3 and L1-T0 are failed, such that there is a cycle buffer dependency (CBD) as the red line draws. Other settings are kept consistent with the above's.

First, we investigated the PFC trigger frequency, which is the most direct indicator that affects deadlock. We simulate DCQCN + SPFC, DCQCN + DT, TIMELY + SPFC and TIMELY + DT with the W1 workload. The target load is 0.5 at ToR down-links. Each scheme is tested 1000 times, and

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

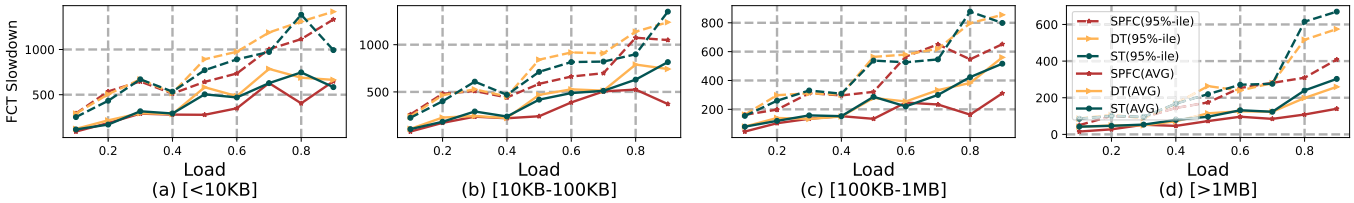


Fig. 15. FCT statistics under DCQCN. [All-to-all communication].

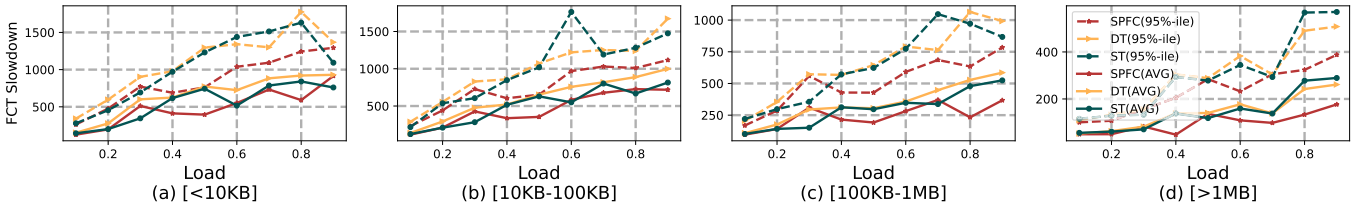
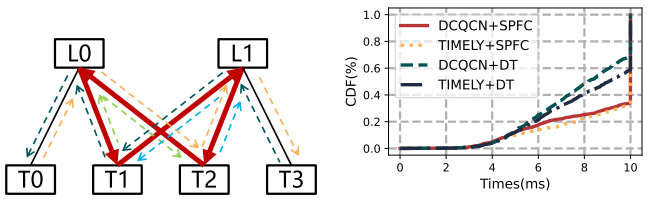


Fig. 16. FCT statistics under TIMELY. [All-to-all communication].



(a) The Deadlock Topology. (b) Frequency of deadlocks under various schemes.

Fig. 17. The deadlock test.

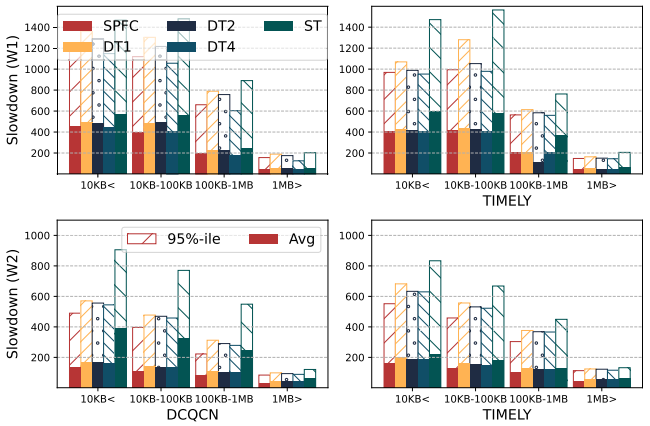


Fig. 18. When facing CBD, other schemes are more prone to triggering PFC PAUSE, resulting in longer FCT. [topology in Fig. 17(a)].

every simulation lasts for 10 ms. We record the time when a deadlock occurs. We record the time when a deadlock occurs and draw the statistical results in Fig. 17(b).

Among the 1000 simulations, SPFC encounters with deadlock for 237 and 209 times with DCQCN and TIMELY, while DT is deadlocked for 612 and 538 times with DCQCN and TIMELY, respectively. SPFC can reduce the probability of deadlock from 61% to 24%. The advantage of SPFC comes from the positive effect of mitigating PFC triggers and stopping congestion spreading.

Fig. 18 shows experimental results when meeting a deadlock. Whether the end-host is using DCQCN or TIMELY, the performance of SPFC is similar to DT but slightly better than ST under W1 (WebSearch workload). SPFC performs better in cases where W2 (Hadoop workload) is the majority of small flows. For small flows (<10KB), it can reduce 95th FCT by up to 45.9% compared to ST. For large flows (>1MB), it can reduce average FCT by up to 53.3% compared to ST. These benefits are due to the fact that SPFC can generate fewer PFC triggers. This shows that when there is CBD in the data center, SPFC can well alleviate the mismatching problem of the congestion control mechanism, and the network performance is greatly improved.

VIII. Related Work

Buffer management in lossy ethernet: Enhanced Dynamic Threshold (EDT) [51] improves the burst absorbing ability of DT by temporally relaxing threshold restraint to absorb microburst traffic. Traffic-aware Dynamic Threshold (TDT) [52] punishes the overwhelmed traffic by limiting the threshold value, thus reserving more buffer to improve the capacity of EDT to absorb the microburst traffic. We argue that they are not applicable to solving the mismatch problem in lossless networks. Take the Enhanced Dynamic Threshold (EDT) [51] as an example. (a) **Different goals:** First of all, EDT is designed for lossy networks. Its goal is to absorb micro-burst traffic so as to give more buffer for burst ports to prevent timeout retransmission due to packet loss. Since the lossless network ensures no packet drops, we design SPFC to give a larger buffer for non-burst traffic in order to absorb mismatch traffic. (b) **Different functions:** In addition, EDT controls the packet drops on the output port, and the control threshold is adjusted based on the egress bytes. The SPFC controls the PFC triggering on the input port, and the threshold is adjusted based on the ingress bytes.

Accurate congestion control: Recently, a large number of congestion control works have been proposed [3], [15], [34], [35], [36], [40], [53], [54]. DCQCN [34] is a congestion control algorithm that combines DCTCP [3] and QCN [42] by replacing the congestion notification mechanism with ECN in the network layer. TIMELY [35] is another congestion

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

control heuristic algorithm, which uses delay measurements to detect congestion, and then adjusts transmission rates according to RTT gradients. HPCC [36] uses switch INT (in-network telemetry) to obtain the precise switch congestion state and calculates the remaining bandwidth. PCN [40] is a receiver-driven RDMA congestion control that differentiates mismatch traffic and thus reduces the amount of PFC PAUSEs. BFC [54] utilizes new hardware and employs a hop-by-hop control mechanism to avoid end-to-end latency.

DCQCN and TIMELY are congestion control algorithms that have been deployed in data centers at present. They are slow evolution-based end-to-end congestion control and have the problem of mismatching that SPFC needs to solve. HPCC and PCN can accurately adjust the sending rate of the end hosts, so they have almost no mismatching problems. However, they need to modify both the end node and the switch at a much higher cost than SPFC, which only needs to modify the switch. Although BFC also eliminates the mismatch problem, it is difficult to scale when there are a large number of flows.

Deadlock mechanism for RDMA: There are several mechanisms available to solve the problem of deadlock in RDMA networks. In order to eliminate cycle buffer dependency (CBD), Tagger [55] placed multiple independent priority queues on each switch and increased the grouping priority hop by hop. In this way, no CBD exists under the same priority, thus eliminating the deadlock. GFC [56] is a fine-grained manipulation port rate that allows all ports to keep packets flowing, even if there is CBD, and theoretically demonstrates that GFC can eliminate deadlocks. The above work is completely orthogonal to SPFC. Tagger [55] and GFC [56] will also encounter the problem of throughput mismatch. SPFC can be well-compatible with the above work.

IX. Conclusion

In this paper, we demonstrate that mismatching of end-host congestion control leads to improper PFC PAUSEs, which can result in congestion spreading, victim traffic phenomenon, and even deadlock. We re-architect buffer management for lossless Ethernet and propose Selective-PFC (SPFC), which provides the ability to occupy more buffer on mismatched traffic ports. SPFC is a practical solution and can be easily implemented on commodity switches with incremental modifications. Extensive evaluations confirm that SPFC greatly improves performance, significantly mitigates PFC PAUSE messages, and reduces the flow completion time under realistic workloads. We believe SPFC should be considered for wide adoption in commodity switches.

Appendix

In the following, we analyze the rate changes of different types of flows based on the model developed in Section IV-C. Accordingly, we find the appropriate K to distinguish different types of ports. Moreover, we determine the conditions that needs to be satisfied to distinguish different types of ports.

When no burst occurs (i.e. $t < t1$), R_{depart} for the victim port is C and R_{depart} for the burst port is 0. When $t = t1$, N burst ports start to send packets at rate C. Since there are $N+1$ ports sending traffic to congested ports, their departure rate per port is

$$R_{depart}^{burst} = \frac{C}{N+1} \tag{5}$$

The victim port departure rate is the victim flow part and congest part, which is:

$$R_{depart}^{victim} = (1-\epsilon)C + \frac{C}{N+1} \tag{6}$$

The flow model is used to analyze the selection of K value. Fig. 8(b) shows the timing diagram. As can be seen from the figure, the $min(R_{depart}^{victim})$ and the $max(R_{depart}^{burst})$ can be distinguished by C/K_{spfc}. We can get

$$(1-\epsilon)C + \frac{C}{N+1} > \frac{C}{K_{spfc}} > \frac{C}{N+1} \tag{7}$$

And just by simplifying, we can get this

$$N+1 > K_{spfc} > \frac{N+1}{(1-\epsilon)(N+1)+1} \tag{8}$$

References

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.

[2] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA: ACM, 2012, pp. 199–210.

[3] M. Alizadeh et al., "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 63–74, Aug. 2011.

[4] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 219–230, Sep. 2013.

[5] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A scalable multi-tenant network architecture for virtualized datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 62–73, 2011.

[6] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2012, pp. 253–266.

[7] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA: ACM, 2012, pp. 139–150.

[8] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, "Practical DCB for improved data center networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 1824–1832.

[9] S. Huang, D. Dong, and W. Bai, "Congestion control in high-speed lossless data center networks: A survey," *Future Gener. Comput. Syst.*, vol. 89, pp. 360–374, Dec. 2018.

[10] C. Tian et al., "P-PFC: Reducing tail latency with predictive PFC in lossless data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1447–1459, Jun. 2020.

[11] Y. Gao et al., "When cloud storage meets RDMA," in *Proc. 18th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2021, pp. 519–533.

[12] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCS can be general and fast," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2019, pp. 1–16.

[13] E. Tasoulas, E. G. Gran, T. Skeie, and B. D. Johnsen, "Fast hybrid network reconfiguration for large-scale lossless interconnection networks," in *Proc. IEEE 15th Int. Symp. Netw. Comput. Appl. (NCA)*, Oct. 2016, pp. 101–108.

[14] I. T. Assoc. (Sep. 2014). *Rocev2*. [Online]. Available: https://cw.infinibandta.org/document/dl/7781

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[15] R. Mittal et al., "Revisiting network support for RDMA," in *Proc. Conf. ACM Special Interest Group Data Commun.* New York, NY, USA: ACM, Aug. 2018, pp. 313–326.
[16] *802.11qbb. Priority Based Flow Control*, IEEE Standard 802.11qbb, 2011.
[17] Cisco. (9300). *Cisco Nexus 9300-ex Platform Switches Architecture*. [Online]. Available: https://people.ucsc.edu/ warner/bufs/nexus-9300-whitepaper.pdf
[18] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 130–140, Apr. 1998.
[19] U. Cummings, A. Lines, P. Pelletier, and R. Southworth, "Shared-memory switch fabric architecture," U.S. Patent 7 814 280, Oct. 12, 2010.
[20] B. O. Olasoji, O. M. Olanrewaju, and I. O. Adebayo, "Transmission control protocol and congestion control: A review of TCP variants," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 3, p. 127, 2016.
[21] Z. Tafa and V. Milutinovic, "Machine learning in congestion control: A survey on selected algorithms and a new roadmap to their implementation," 2021, *arXiv:2112.15522*.
[22] (2019). *Network Simulator 3*. [Online]. Available: https://www.nsnam.org/
[23] DPDK. (2020). *Intel*. [Online]. Available: http://dpdk.org
[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.*, Nov. 2010, pp. 267–280.
[25] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Apr. 2010, pp. 446–452.
[26] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1377–1385.
[27] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.
[28] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM Conf. Special Interest Group Data Commun.* New York, NY, USA: ACM, Aug. 2015, pp. 123–137.
[29] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter, "Bullet trains: A study of NIC burst behavior at microsecond timescales," in *Proc. 9th ACM Conf. Emerg. Netw. Experiments Technol.*, Dec. 2013, pp. 133–138.
[30] C. Guo et al., "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA, Aug. 2016, pp. 202–215.
[31] G. Lee, "Ethernet fabric requirements for FCoE in the data center," in *Proc. Ethernet Technol. Summit*, 2010, pp. 1–14.
[32] S. Hu et al., "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 92–98.
[33] A. Shpiner, E. Zahavi, V. Zdornov, T. Anker, and M. Kadosh, "Unlocking credit loop deadlocks," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 85–91.
[34] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, Aug. 2015.
[35] 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.
[36] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun.* New York, NY, USA: ACM, Aug. 2019, pp. 44–58.
[37] K. Ramakrishnan, S. Floyd, and D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, document RFC 3168, 2001.
[38] Y. Zhu et al., "Packet-level telemetry in large datacenter networks," in *Proc. ACM Conf. Special Interest Group Data Commun.* New York, NY, USA: ACM, Aug. 2015, pp. 479–491.
[39] S. Das and R. Sankar, "Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications," Broadcom, Palo Alto, CA, USA, White Paper 12358325, 2012.
[40] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless Ethernet," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement.*, 2020, pp. 19–36.
[41] M. Alizadeh et al., "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.
[42] *802.11qau. Congestion Notification*, IEEE Standard 802.11qau, 2010.
[43] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA: ACM, Aug. 2011, pp. 350–361.
[44] E. Ghabashneh, Y. Zhao, C. Lumezanu, N. Spring, S. Sundaresan, and S. Rao, "A microscopic view of bursts, buffer contention, and loss in data centers," in *Proc. 22nd ACM Internet Meas. Conf.*, Oct. 2022, pp. 567–580.
[45] H. M. Wassel et al., "SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip," *ACM SIGARCH Comput. Architect. News*, vol. 41, no. 3, pp. 583–594, 2013.
[46] R. S. Marques et al., "A flow-based multi-agent data exfiltration detection architecture for ultra-low latency networks," *ACM Trans. Internet Technol.*, vol. 21, no. 4, pp. 1–30, Nov. 2021.
[47] (2020). *DPDK Test-Pipeline*. [Online]. Available: https://github.com/dpdk/dpdk/tree/master/test/test-pipeline
[48] (2023). *Connectx®-5 En Card*. [Online]. Available: https://www.nvidia.com/en-us/networking/ethernet/connectx-5/
[49] (2023). *Connectx®-6 En Card*. [Online]. Available: https://www.nvidia.com/en-us/networking/ethernet/connectx-6/
[50] (2020). *Open Fabrics Enterprise Distribution (OFED) Performance Tests*. [Online]. Available: https://github.com/linux-rdma/perftest
[51] D. Shan, W. Jiang, and F. Ren, "Analyzing and enhancing dynamic threshold policy of data center switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2454–2470, Sep. 2017.
[52] S. Huang, M. Wang, and Y. Cui, "Traffic-aware buffer management in shared memory switches," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.
[53] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA: ACM, 2017, pp. 239–252.
[54] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2022, pp. 779–805.
[55] S. Hu et al., "Tagger: Practical PFC deadlock prevention in data center networks," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.* New York, NY, USA: ACM, Nov. 2017, pp. 451–463.
[56] K. Qian, W. Cheng, T. Zhang, and F. Ren, "Gentle flow control: Avoiding deadlock in lossless networks," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 75–89.

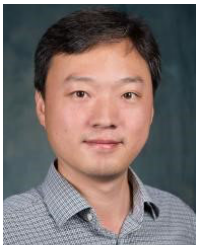
Hanlin Huang (Graduate Student Member, IEEE) received the B.E. degree from the Software College, Nankai University, Tianjin, China, in 2021. He is currently pursuing the Ph.D. degree with Tsinghua University. His research interests include data center networks and AQM.

Xinle Du received the B.E. degree from the Department of Computer Science and Technology, Xidian University, Xi'an, China, in 2018, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2023. He has been working as the Chief Engineer of the Computer Network and Protocol Laboratory, Huawei, since 2023. His research interests include networking and LLM systems.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

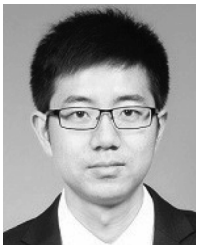
Tong Li (Member, IEEE) received the B.E. degree from the School of Computer Science, Wuhan University, China, in 2012, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2017. He held a Visiting Scholar with the School of Computer Science and Electronic Engineering, University of Essex, U.K., in 2014 and 2016. He worked as the Chief Engineer of Huawei before 2022. He is currently an Associate Professor with the Renmin University of China. His research interests include networking, distributed systems, and big data.



Haiyang Wang (Member, IEEE) received the Ph.D. degree in computer science from Simon Fraser University, Burnaby, BC, Canada, in 2013. He is currently an Associate Professor with the Department of Computer Science, University of Minnesota Duluth. His research interests include cloud computing, peer-to-peer networking, social networking, big data, and multimedia communications.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science, Tsinghua University. He has published more than 200 technical articles and holds 11 U.S. patents in the research areas of next-generation internet, blockchain systems, the Internet of Things, and network security. He serves as the Steering Committee Chair for IEEE/ACM IWQoS. He has guest-edited several special issues for IEEE and Springer journals. He is an Editor of IEEE INTERNET OF THINGS JOURNAL.

Mowei Wang received the B.E. degree in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, in 2022. He has been working as the Chief Engineer of Huawei Datacom Production Line since 2022. His research interests include networking and artificial intelligence.



Huichen Dai (Member, IEEE) received the Ph.D. degree in computer science and technology from Tsinghua University in 2016. He is currently the Team Leader of ByteDance, specializing in AI network transport protocols and algorithms to optimize LLM training and inference performance. Before joining ByteDance, he was a Senior Engineer with Huawei, with a focus on high-performance network protocol design and implementation on smart NICs. He has a strong background in high-performance networking (RDMA/RoCE) and his research interests include congestion control, collective communication offloading, and multi-path transmission.

Authorized licensed use limited to: Tsinghua University. Downloaded on July 24,2024 at 08:01:04 UTC from IEEE Xplore. Restrictions apply.