A Flow Control Scheme based on Per Hop and Per Flow in Commodity Switches for Lossless Networks

SHIE-YUAN WANG1, (Senior Member, IEEE), YO-RU CHEN1, HSIEN-CHUEH HSIEH1, RUEI-SYUN LAI1, AND YI-BING LIN1, (Fellow, IEEE)

1Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Shie-Yuan Wang (e-mail: shieyuan@cs.nctu.edu.tw).

This work was financially supported by the Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

ABSTRACT Performing flow control inside a network can effectively avoid packet loss due to buffer overflow in switches. IEEE 802.3x exercises a per-hop link-based flow control scheme to achieve this goal. IEEE 802.1Qbb Priority-based Flow Control (PFC) improves IEEE 802.3x by proposing a per-hop per-service-class flow control scheme. Although PFC is better than IEEE 802.3x, it still suffers from several serious problems such as congestion spreading, deadlock, and packet loss. In this work, we propose a per-hop per-flow scheme to mitigate these problems. We design, implement, and evaluate the performance of our scheme in P4 hardware switches. Experimental results show that (1) our scheme outperforms PFC in many aspects including avoiding congestion spreading, deadlock, and packet loss, and (2) the bandwidth overhead of our scheme is only slightly higher than that of PFC.

INDEX TERMS Flow control, IEEE 802.3x, IEEE 802.1Qbb, PFC, P4

I. INTRODUCTION

P A C K E T losses due to buffer overflow in network switches may occur due to network congestion. If the UDP protocol is used at the transport layer, a UDP sending host will not automatically retransmit lost packets, which may result in poor Quality-of-Service (QoS) of the network applications on the receiving host. Although a TCP sending host will automatically retransmit lost packets, the packet loss detection time of TCP is longer than the round-trip time (RTT) of the TCP connection. For delay-sensitive network applications such as a network file server and its clients or a remote desktop server and its clients, the delays will severely harm their performances.

To prevent dropped packets due to buffer overflow in network switches, IEEE 802.3x [1] proposed a per-hop link-based flow control scheme to ensure zero packet loss between two adjacent network nodes. Because IEEE 802.3x is link-based, when a link is stopped, all flows on the link are stopped, including those flows that are not congested in the downstream node. Besides, when a link is stopped due to a high buffer usage of low-priority flows, flows of high priorities are stopped as well. Such problems can easily lead to a low link utilization and unfairness among flows.

To solve these problems, the IEEE 802.1Qbb priority-based flow control (PFC) scheme [2] was proposed. To support QoS, the IEEE 802.1p task group defines a 3-bit Priority Code Point (PCP) field to define 8 priority levels, where 7 represents the highest priority and 0 represents the lowest priority. The default value is 0. Each priority level defines a unique class of service (CoS). To comply with IEEE 802.1p, PFC supports eight different priorities and in PFC an input port can have up to eight different CoS ingress queues. PFC has been widely adopted for a variety of lossless networks such as Fiber Channel over Ethernet (FCoE) and RDMA over Converged Ethernet version 2 (RoCEv2) [3], which is used by some data center networks [4].

Although PFC enhances IEEE 802.3x, it still suffers from problems such as unfairness and head-of-line blocking [4]. Furthermore, PFC can cause congestion spreading problem [4] [5] [6] and result in a significant performance degradation. Even worse, PFC can lead to deadlock [6] [7] [8] [9] [10] that causes an entire network to enter a standstill situation.

Enhancing PFC with a per-hop per-flow flow control
mechanism can potentially solve many PFC problems. For example, the pause/resume flow control can be applied to an individual flow rather than to all flows belonging to the same service class. As a result, there is no victim flow in the network and the unfairness problem among flows no longer exists. Besides, because each flow is individually allocated a separate buffer space in the switch, the deadlock and congestion spreading scenarios and their resulting low link utilization problems no longer exist. For this reason, BFC [11] proposed a clean-slate per-hop per-flow flow control design and showed its superior performance under different situations via simulations.

In an ideal situation, each flow should have its own buffer usage state and can be paused and resumed using dedicated egress queue independently in a per-hop per-flow flow control scheme. Currently, to implement IEEE 802.1p, most hardware switches support at least eight priority queues in an output port. However, most of them lack the capability of per-flow queuing in an output port due to its high complexity and cost. To resolve this issue, this paper proposes a novel flow control scheme called PFFC based on per hop and per flow.

In PFFC, we design a per-flow based buffer usage accounting mechanism to decide which flow should be paused or resumed according to per-flow thresholds. This mechanism uses the concepts of default egress queue (DEQ) and paused egress queue (PEQ). If a flow needs to be paused, PFFC immediately directs its packets to a PEQ. When the flow can be resumed, PFFC immediately directs its packets back to the DEQ. PFFC reuses PFC’s PAUSE and RESUME control frames to pause and resume an egress queue. When the number of flows passing an output port is too large, PFFC maps multiple flows to an egress queue. In addition to using per-flow buffer usage thresholds, PFFC also uses system buffer usage thresholds to protect the usage of buffers.

We successfully implement PFFC in programmable hardware switches [12] [13] using the P4 language [14]. These switches use Intel/Barefoot’s Tofino chip as their switching ASIC [15] and their data planes are programmable. We evaluate the real-world performance of PFFC on a multi-hop network testbed composed of several P4 switches and hosts. Experimental results show that PFFC outperforms PFC in many aspects and its bandwidth overhead is only slightly higher than that of PFC. In summary, our contributions are listed as follows.

- We design a novel per-hop per-flow flow control scheme named PFFC and successfully implement it in P4 hardware switches.
- We identify the conditions in which packets may be dropped in PFC.
- We evaluate the real-world performance of PFFC in different aspects including multi-hop, multi-flow, multi-protocol-type (TCP vs. UDP), and multi-flow-type (mice flows vs. elephant flows) scenarios and show that PFFC maintains high-performance and low-overhead over multi-hop networks.
- We show that PFFC does not have the circular wait deadlock. Experimental results show that congestion spreading and packet loss problems are mitigated, and the average flow completion time of mice flows in PFFC is shorter than that in PFC.

This paper is organized as follows. Section II gives the background, compares our work with related work, briefly presents the P4 switch architecture to help readers understand the design and implementation of PFFC. Section III presents the detailed design and implementation of PFFC. In Section IV, we evaluate and compare the performances of PFFC with those of PFC. We discuss our future work in Section V. Lastly, we conclude the paper in Section VI.

II. BACKGROUND AND RELATED WORK

The concept of IEEE 802.3x is illustrated in Fig. 1, in which a small box represents a packet and the number on it represents the flow ID of the packet. The ingress queue is the queue used in an input port of the downstream node \(N_R\) while the egress queue is a queue used in an output port of the upstream node \(N_S\). When the buffer usage of the ingress queue of \(N_R\) exceeds a specific threshold \(X_{off}\), \(N_R\) will promptly send a PAUSE frame to \(N_S\) to ask it to stop sending packets. Later on, when the buffer usage drops below another threshold \(X_{on}\), \(N_R\) will send a RESUME frame to \(N_S\) to resume transmission of packets on the link. To ensure that no packets are dropped in a network due to buffer overflow in any switch, IEEE 802.3x is exercised on every link. As mentioned in Section I, IEEE 802.3x suffers from several problems. This section describes the existing solutions for these problems.

A. PRIORITY-BASED FLOW CONTROL (PFC)

PFC, like IEEE 802.3x, is a per-hop flow control scheme and needs to be exercised on the links where reliable Ethernet is required. Furthermore, it supports priority differentiation when performing flow control. When packets enter a switch via an input port, they are placed in different ingress queues of the input port based on their priorities, and the pause/resume flow control scheme is independently applied to these ingress queues.

Fig. 2 shows the design of PFC for a shared-buffer switch in a logical view. PFC uses the ingress perspective and the egress perspective at the same time to limit the maximum buffer usage of a specific CoS in a switch [16] [17]. As explained previously, each input port has eight CoS ingress queues and each output port has eight CoS egress queues. When a packet enters a switch, it is stored in a packet buffer.
shared by all ports of the switch. This shared packet buffer does not belong to any input port or output port. Based on the CoS of a packet, the packet is logically linked into the CoS queue of the input port where it comes from and at the same time linked into the CoS queue of the output port where it should be forwarded out of the switch. To illustrate this relationship, in Fig. 2, if there are \( n \) packets of a specific flow in the ingress queues, there are exactly \( n \) packets of that flow in the egress queues.

Each CoS ingress queue in an input port is configured a maximum queue length and its \( X_{off} \) and \( X_{on} \) thresholds are properly set based on its maximum queue length setting. As explained previously, based on the current buffer usage of a CoS ingress queue, the switch will send PAUSE or RESUME frames to the upstream node to prevent packet dropping in its CoS ingress queue. Each CoS egress queue in an output port also has a maximum queue length setting. When its current buffer usage exceeds the specified maximum queue length setting, the packet to be linked into it will be dropped.

![FIGURE 2. The PFC design in a switch](image)

In PFC, the PAUSE frame contains a timeout value. After the timer expires, the transmission at the upstream node automatically resumes. Setting the \( X_{off} \) value should consider the bandwidth-delay product (BDP) of the link between the switch and the upstream node. After the switch sends a PAUSE frame to the upstream node, the packets from the upstream node will keep filling up the CoS ingress queue of the switch during the RTT between the switch and the upstream node. Therefore, the \( X_{off} \) value should be set to a value such that the space between \( X_{off} \) and the size of the CoS ingress queue is larger than the BDP to accommodate these in-flight packets. This space is called the headroom of the queue [16] [17]. A RESUME frame is used to shorten the timeout duration at the upstream node. Because when the switch sends a RESUME frame, it takes the RTT for packets from the upstream node to arrive, to achieve high switch utilizations, setting the \( X_{on} \) value should also consider the BDP of the link between the switch and the upstream node and it should not be too small.

Numerous approaches have been proposed for solving the problems with PFC, which can be classified into four categories: (i) to focus on solving PFC deadlock problem [9] [10], (ii) to enhance PFC with other methods such as congestion control [7] [18] or telemetry technique [7] [18] [19], (iii) to avoid using PFC [20] [21], and (iv) to revisit finer-grained flow control schemes [11].

Although these methods can work around some problems with PFC, none of them can completely solve all of these problems using off-the-shelf hardware. For example, both Tagger [9] and GFC [10] can solve deadlock problems but cannot solve congestion spreading issues. The congestion control methods such as DCTCP [22], QCN [23], DCQCN [7], TIMELY [24], HPCC [18], and Swift [25] rely on using an end-to-end control loop, which causes larger tail latency than the tail latency caused by hop-by-hop based approaches. Besides, these methods cannot solve PFC deadlock and congestion spreading problems.

### B. CONGESTION SPREADING IN PFC

Fig. 2 shows an example of congestion spreading in PFC. Each small box in the queues represents a packet and the number on the box indicates its flow ID. Flow 1 comes from \( N_{S1} \) and Flow 3 comes from \( N_{S2} \) and they are destined to \( N_{R1} \); Flow 2 comes from \( N_{S1} \) and Flow 4 comes from \( N_{S2} \) and they are destined to \( N_{R2} \). All of these flows use the default CoS and thus the packets of Flow 1 and Flow 2 share Ingress queue 1 while the packets of Flow 3 and Flow 4 share Ingress queue 2. The link bandwidth between the switch, senders (\( N_{S1}, N_{S2} \)), and receivers (\( N_{R1}, N_{R2} \)) are the same.

Consider the following condition: Flows 1 and 3 are sending at the line rate but Flows 2 and 4 are sending at a half of the line rate. This means that the total sending rate of Flows 1 and 3 exceeds the bandwidth of Link A. As a result, Egress queue 1 quickly builds up. Suppose that the size of Egress queue 1 is larger than the sum of the sizes of Ingress queue 1 and Ingress queue 2, which are used by the two flows, respectively. In such a setting, before Egress queue 1 overflows, the buffer usages of Ingress queue 1 and Ingress queue 2 will have exceeded the \( X_{off} \) value configured for them. (Note that the \( X_{off} \) configured for an Ingress queue must be less than the size of the Ingress queue.) This means that PFC at the two input ports has already been triggered to send PAUSE frames to \( N_{S1} \) and \( N_{S2} \). Thus, the packets of Flows 1 and 3 will not be dropped in Egress queue 1.

When the PAUSE frames are received by \( N_{S1} \) and \( N_{S2} \), because in this case all flows use the same CoS, \( N_{S1} \) and \( N_{S2} \) will immediately pause all of their flows destined to the switch. Thus, although the congestion is caused by Flows 1 and 3, Flows 2 and 4 are also paused. Because the packets of Flows 2 and 4 are blocked from transmitting at \( N_{S1} \) and \( N_{S2} \), they quickly build up and cause congestion at \( N_{S1} \) and \( N_{S2} \). Such congestion will be spread to all upstream nodes of Flows 2 and 4 even though their packets are destined to \( N_{R2} \), where no congestion exists. In this case, Flows 2 and 4 are victim flows and their packets sometimes cannot be forwarded by the switch. Ideally, the utilization of Link B should be 100%. However, when PFC is enabled, because Flows 2 and 4 are unnecessarily paused, the utilization of Link B will be unnecessarily reduced.
Because the number of flows on a link generally is much larger than eight, which is the number of CoS ingress queues in an input port, inevitably multiple flows will share the same CoS ingress queue in an input port. Thus, some flows sharing the same CoS ingress queue may become victim flows due to congestion spreading. The congestion spreading problem can be avoided in PFFC. The details will be given in Section IV-F.

C. DEADLOCK IN PFC

Tagger [9], GFC [10] and PCN [19] study the deadlock formation problem in PFC-enabled lossless networks. PFC deadlock is mainly caused by the Cyclic Buffer Dependency (CBD) problem [8] [9]. When PFC is enabled in a loop topology, the occupied buffers are waiting for each other, and packet transmission will be halted immediately after one of the switches get congested. PFC deadlock also occurs in a Clos network topology [4] as transient loops form in data center networks. The deadlock is persistent even after stopping packet transmission from all senders. The deadlock can only be removed until all switches get rebooted.

A deadlock situation on a resource arises if and only if all of the following conditions are met in a system: (1) no preemption, (2) mutual exclusion, (3) hold and wait, and (4) circular wait. In PFFC, every flow is allocated its own buffer space in default egress queue, and no circular waiting condition exists. Therefore, condition (2) does not hold and the deadlock situation will not occur in PFFC. More details will be given in Section IV-G.

D. PACKET LOSS IN PFC

Although PFC was proposed to prevent packet loss in a switch, we observed a situation in which packets can still be dropped due to the overflow of an egress queue. Consider Flows 1 and 3 in Fig. 2. Suppose that the sizes of all ingress queues and egress queues are the same. It may happen that although none of Ingress queues 1 and 2 exceed the $X_{off}$, threshold, the buffer usage of Egress queue 1 has exceeded its size, causing some packets of Flows 1 and 3 to be dropped. For example, if $X_{off}$ is set to 3/4 of the ingress queue size, before the buffer usages of Ingress queues 1 and 2 reach $X_{off}$, the sum of their buffer usages may already exceed the size of Egress queue 1. That is, 3/4 + 3/4 = 6/4 > 1. In this situation, because neither Ingress queue 1 nor Ingress queue 2 exceeds the $X_{off}$ threshold, PFC will not send any PAUSE frame to $N_{S1}$ or $N_{S2}$ to stop their packet transmission. If such a traffic pattern persists, this packet dropping phenomenon will continue. Therefore, PFC is vulnerable to TCP incast [26] [27] [28], especially when the number of ingress queues is large.

One way to resolve this issue is to allocate each egress queue a buffer space that is $N$ times the buffer size of an ingress queue, where $N$ is the number of ports of the switch, which generally is 24, 48, or 64 for a commodity switch. This number is called the buffer oversubscription ratio used for an egress queue. However, since a switch may encounter traffic patterns in which the packets are spread among all ports, the buffer space of a switch should be shared fairly among all ports and between ingress queues and egress queues. To better utilize the buffer space allocated to an egress queue, some commodity switches can allow an egress queue to have a buffer oversubscription ratio of up to 3:1 [16] [17]. However, it is unrealistic to use the number of ports of a switch as the buffer oversubscription ratio for each egress queue. Therefore, the packet loss problem in PFC cannot be completely avoided.

In PFFC, a flow is allocated its own buffer space so that its buffer usage is individually accounted and controlled. PFFC only uses the egress perspective to limit the buffer usage of a flow. As a result, PFFC does not have the packet loss problem with PFC. The details will be given in Section IV-D.

E. COMPARISON WITH RELATED WORK

Per-hop flow control can generally be classified into the credit-based flow control and Ethernet flow control. Credit-based flow control [29] [30] have been proposed for ATM networks decades ago and InfinitiBand also uses it [31]. In credit-based flow control, each receiver sends credits to the sender to indicate the availability of receive buffers. The sender needs to obtain credits before transmitting packets to the receiver. The packets of each flow are buffered in separated queues in the upstream node and can only be transmitted after the downstream node has granted permission.

Because ATM networks are not widely used, credit-based flow control is rarely used in today’s networks. Since PFC is the primary flow control scheme proposed for Ethernet networks, nowadays PFC is used to implement a lossless Ethernet network. Recently, because the emergence of RDMA over Converged Ethernet version 2 (RoCEv2) [3] and the deployment of RDMA NICs in data center networks made PFC gain wider traction [4], a variety of approaches have been proposed to either enhance, disable, or solve problems caused by PFC, and they have been discussed Section II.

Our work differs from prior works as follows. First, we have successfully designed and implemented a per-hop per-flow flow control scheme (PFFC) in P4 hardware switches. In contrast, most of prior works did not use per-hop per-flow flow control. Among these prior works, Back pressure Flow Control (BFC) [11] has independently proposed similar ideas to our work. That is, when the number of flows is larger than the number of queues in an output port, some flows need to share the same queue. Except for this unavoidable measure, however, there are many design differences between PFFC and BFC. For example, PFFC uses migrate and migrate-back mechanisms to pause and resume a flow, respectively. On the other hand, the BFC scheme does not have such mechanisms. Instead, it pauses the queue that a congested flow shares with other flows. As a result, these other flows may be unnecessarily paused as well. The updated BFC work [32] briefly described its implementation on Tofino2 P4 switches without details. The BFC authors claimed that, flows exceeding the number of egress queues will be rare in practice and even that happens a BFC switch can leverage the larger number of
queues available at the upstream switches to limit congestion spreading. In this paper, PFFC is proposed as a fresh take and an interesting idea implemented in a capability-limited Tofino1, which also works for Tofino2. In our opinion, both BFC and PFFC are novel implementations for Tofino. We have also tested the behavior of our PFFC scheme and measured its performance in P4 hardware switches under many different settings such as multi-hop, multi-flow, multi-protocol (i.e., TCP vs. UDP), multi-flow-type (i.e., elephant flow vs. mice flow [33]), etc. These real-world performances are more convincing than simulation results.

F. P4 SWITCH ARCHITECTURE

The programmability of the data plane of a switch is the key enabler to develop a clean-slate PFFC scheme. P4 is one of the emerging programmable data plane technologies that allow developers to design and implement their packet processing logics in the data plane [34]. Based on the P4 language specification, Fig. 3 shows the architecture of a P4 hardware switch [14]. Each packet received by an input port is processed by the following components in a sequential order: Ingress Match-Action Pipeline (IMAP) (Fig. 3 (A)), Traffic Manager & Packet Buffer (TMPB) (Fig. 3 (B)), Egress Match-Action Pipeline (EMAP) (Fig. 3 (C)), and an output port. The two Match-Action pipelines are main programmable components for packet processing. TMPB serves to provide packet queuing (ingress queues and egress queues) and scheduling functions.

![FIGURE 3. The architecture of a P4 hardware switch](image)

In a P4 switch, an important data structure is used to describe the current state of a packet or to specify the operation that the switch should apply to the packet. This structure is called intrinsic metadata and each packet is associated with a metadata instance in the switch. (We use the “metadata” to refer to the “intrinsic metadata.”) For example, when a packet passes IMAP, the values of the output port ID and egress queue ID fields of its metadata instance will be determined and set. When the packet leaves IMAP, TMPB will store it in the specified egress queue of the specified output port based on the information stored in its metadata instance. In PFFC, a flow may need to be directed to a different egress queue over time and this is achieved by using the above mechanism.

A P4 program is mainly executed in IMAP and EMAP. Both IMAP and EMAP are comprised of a Parser, several Match-Action Units (MAUs), and a Deparser. The Parser is used to analyze each packet, extract packet header fields, and pass them to a sequence of MAUs. The MAU is the core component to define the packet processing conditions and action functions. Each MAU is described by a set of Match-Action Tables (MAT), which are defined by a set of Match Keys (packet header fields and metadata used for matching) and Action Functions (AF) (some operations to be applied to the packet). There are multiple MAU stages in a Match-Action Pipeline. Each Match-Action Table can be assigned to a specific stage. Each stage has hardware resources such as memory and ALUs (Arithmetic Logic Units).

The MAUs work in a packet-driven manner. No action function can be invoked without a packet entering the P4 switch. The life scope of the metadata of a packet is the same as that packet. The information that needs to be stored in the P4 switch in a persistent manner should be stored in a memory called register arrays. Every register can be read or written by the ALUs of a MAU. Each register can only be accessed in a specific MAU stage to prevent race condition problems caused by concurrent accesses from multiple stages.

A program written in the P4 language is only a half part of the complete packet processing logic. The set of runtime Match-Action rules is the other half, which is used to describe the association of Match Keys and corresponding Action Functions. When a packet is processed by an MAU, it may be matched by multiple rule entries and the action function(s) of the highest-priority entry will be applied to the packet.

After a packet is processed by IMAP, it is passed to TMPB. TMPB is composed of shared Packet Buffer and the packet queuing and scheduling modules. The egress port set for a packet in IMAP can be an output port or a recirculation port (Fig. 3 (E)). The recirculation port is a special mechanism that enables the packet to be processed again by re-entering IMAP. When a packet is mirrored in IMAP or EMAP, the packet mirroring is implemented using an operation, called “cloning.” The cloned copy will be stored in Mirror Buffer (Fig. 3 (D)) and later on it will be processed by TMPB.

G. ACRONYMS AND SYMBOLS USED IN THIS PAPER

The acronyms and symbols used in this paper are summarized in Table 1.

III. DESIGN AND IMPLEMENTATION

We have designed and implemented PFFC in Inventec D10056 [13] P4 switches, which use Barefoot’s Tofino chip as their switching ASIC [15]. In the rest of the paper, we use the “P4 switches” to refer to these hardware switches. The work flow of PFFC is shown in Fig. 4. Our design includes three main mechanisms: A per-flow based buffer...
usage accounting mechanism to decide which flow should be paused or resumed according to the per-flow threshold $X_{off_f}$ (where $f$ represents a specific flow) and $X_{on_f}$ values (Fig. 4 (A)), the capability to generate and send per-flow control frames to notify upstream nodes which flows should be paused or resumed (Fig. 4 (B)), and the ability to react to the received per-flow control frames (Fig. 4 (C)). Our PFFC implementation uses PFC’s PAUSE and RESUME control frames to pause and resume an egress queue. We also create custom PFFC control frames named MIGRATE and MIGRATE-BACK to migrate a flow among egress queues. A packet enters the ingress pipeline following the workflow in Fig. 4. Each packet will be parsed and processed. A normal packet will be forwarded to a downstream node. An accounting packet will be recirculated to the ingress pipeline to decrement the counters of the packet. A PFC/PFFC control frame will be mirrored to an upstream node to pause or migrate a flow according to the thresholds.

To support per-flow pause and resume operations under this constraint, when the number of flows passing an output port is large, PFFC maps multiple flows to an egress queue. In our design, if a flow is not paused, we direct its packets to the default egress queue (DEQ, Fig. 5 (B)) of its output port for forwarding. On the other hand, if a flow needs to be paused, we direct its packets to an egress queue that has been paused from a downstream node. However, due to the high cost and complexity to support many per-flow queues in an output port, the number of egress queues per output port in most commodity switches is limited to only eight for supporting the QoS priorities defined in IEEE 802.1p. Modern switches can provide more than eight egress queues per output port. Without loss of generality, we assume that there are eight egress queues in an output port. Having more egress queues per output port can mitigate the issues discussed in Section III-D.

### TABLE 1. The acronyms and symbols

| Name   | Description                                      |
|--------|--------------------------------------------------|
| ALU    | Arithmetic Logic Unit                            |
| $B$    | Packet buffer space                              |
| $B_c$  | Egress packet buffer space                       |
| $B_i$  | Ingress packet buffer space                      |
| $B_p$  | Output port buffer space                         |
| $B_q$  | Output queue buffer space                         |
| BDP    | Bandwidth-Delay Product                          |
| CoS    | Class-of-Service                                 |
| DEQ    | Default Egress Queue                             |
| EMAP   | Egress Match-Action Pipeline                     |
| FBU    | Per-flow Buffer Usage                            |
| FCT    | Flow-Completion Time                             |
| FID    | Flow ID                                          |
| IMAP   | Ingress Match-Action Pipeline                    |
| MAU    | Match-Action Unit                                |
| $N_R$  | Downstream node (receiver)                       |
| $N_S$  | Upstream node (sender)                           |
| PBU    | Packet Buffer Usage                              |
| PEQ    | Pause Egress Queue                               |
| PFC    | Priority-based Flow Control                      |
| PFFC   | Per-hop per-Flow Flow Control                    |
| RTT    | Round-Trip Time                                  |
| TMPB   | Traffic Manager & Packet Buffer                  |
| $X_{off_f}$ | PFC’s pause threshold                        |
| $X_{off_s}$ | PFC’s per-flow pause threshold                 |
| $X_{off_f}$ | PFC’s system buffer pause threshold            |
| $X_{on_f}$ | PFC’s resume threshold                         |
| $X_{off_s}$ | PFC’s per-flow resume threshold               |
| $X_{on_s}$ | PFC’s system buffer resume threshold          |

### FIGURE 4. The high-level workflow of PFFC

A. PER-FLOW PAUSE AND RESUME OPERATIONS

If the number of flows passing an output port is less than the number of egress queues in the output port, our P4 code in IMAP (Fig. 3 (A)) can direct the packets of these flows to different egress queues and independently pause or resume these queues when receiving a PAUSE or RESUME frame from a downstream node. However, due to the high cost and complexity to support many per-flow queues in an output port, the number of egress queues per output port in most commodity switches is limited to only eight for supporting the QoS priorities defined in IEEE 802.1p. Modern switches can provide more than eight egress queues per output port. Without loss of generality, we assume that there are eight egress queues in an output port. Having more egress queues per output port can mitigate the issues discussed in Section III-D.

### FIGURE 5. The DEQ and PEQs in an output port in PFFC
by our design. In an output port, one egress queue is used as DEQ while the other egress queues are used as the \textit{pause egress queue} (PEQ, Fig. 5 (C)). Later on, when the paused flow is resumed, we resume the PEQ that is being used by this flow to drain the packets blocked in the PEQ and then redirect the newly incoming packets of the flow back to the DEQ for forwarding. Since an output port of a commodity switch has eight egress queues in an output port (Fig. 5 (A)), one egress queue is used as the DEQ and the other seven egress queues are used as PEQs shared by all flows.

PFFC identifies a flow by its flow ID (FID), which is the CRC-32 hash value of the 5-tuple information (including source IP address, source port number, destination IP address, destination port number, and the value of the protocol type) in the packet header. The FID is mapped by a function to a value between 1 and 7 to index one of the seven PEQs. When a switch needs to pause a specific flow, it first uses the FID of the flow to pause the indexed PEQ and then directs newly arriving packets of the flow from the DEQ to the indexed PEQ. This design soon stops the packets of the flow from being transmitted out of the output port. In PFFC, the process of changing a flow’s egress queue between the DEQ and its indexed PEQ is called \textit{flow migration}.

On the other hand, when an upstream node resumes a specific flow, it first resumes the PEQ indexed by the FID of the flow, which enables the packets blocked in the PEQ to be forwarded out immediately. Besides, the egress queue for this flow in the upstream node is \textit{migrated} back to the DEQ so that its newly arriving packets will be forwarded using the default queue. This flow migration need not be performed immediately as the paused queue has been resumed and newly arriving packets can still be forwarded through the newly resumed queue. However, since a PEQ may be shared by multiple flows and it may be paused again due to the need to pause a different flow that shares this PEQ, this flow migration still needs to be performed soon so that the packets of the resumed flow will not be unnecessarily blocked again.

Another issue with this many-to-one mapping design is that a flow blocked in a PEQ may be unexpectedly resumed. This phenomenon can happen when another flow shares the same PEQ and the switch needs to resume it. In such a case, the packets of the flow that should be paused in the PEQ may have different sizes and a packet is divided and stored as multiple 80-byte cells in the used P4 switches, to accurately track the buffer usage, the unit of FBU is cell. FBU counters are implemented as a register array in our P4 program to track the usage of Packet Buffer of a switch. The FBU counter is incremented by every incoming packet and its value is stored in a register. The design and implementation of PFFC is based on both IEEE 802.3x and IEEE 802.1Qbb (PFC) standards. The $X_{off_s}$ threshold used in PFFC can prevent packet loss in the packet buffer effectively since $X_{off_s}$ plays the same role as $X_{off}$ used in the link-level lossless standard 802.3x.

Fig. 6 illustrates the complete process of per-flow buffer usage accounting in PFFC. When a packet enters IMAP (Fig. 6 (A)), the FBU and PBU counters are incremented by the packet size respectively as $FBU[FlowID] += PacketSize$ and $PBU += PacketSize$. Then, the updated counter values are compared with the $X_{off_s}$ and $X_{off_f}$ thresholds. If FBU is greater than $X_{off_f}$, our P4 code in IMAP will generate and send a PAUSE frame to the upstream node of the flow to pause the PEQ used by the flow. If PBU is greater than $X_{off_s}$, our P4 code will generate and send a PAUSE frame to the upstream node of the flow to pause both the DEQ and the PEQ used by the flow. The packet is then sent to TMPB (Fig. 6 (B)) for queuing and scheduling. Next, the packet is sent to EMAP (Fig. 6 (C)) for further processing and finally forwarded out of the switch (Fig. 6 (D)).

Although EMAP in Fig. 6 is the right place to decrement the FBU and PBU counters, this cannot be done in the P4 node.

When a flow is migrated back to the DEQ, to minimize the chance of packet reordering that may occur during this transition period, we set the highest \textit{strict priority} (i.e., 7) for PEQs and the lowest (i.e., 0) for DEQ. Thus, for a resumed flow, all of its packets blocked in its indexed PEQ will be forwarded earlier than its newly arriving packets.

### B. PER-FLOW BUFFER USAGE ACCOUNTING

Neither the P4 language nor the P4 switches provide per-flow buffer usage accounting. Thus, we have to design and implement our own per-flow buffer usage (FBU) counters (Fig. 6 (F)) to track the buffer usages of every flows. In PFFC, each flow is allocated its own buffer space. Because packets may have different sizes and a packet is divided and stored as multiple 80-byte cells in the used P4 switches, to accurately track the buffer usage, the unit of FBU is cell. FBU counters are implemented as a register array in our P4 program to track the buffer usage of every flow by its FID. When a packet enters the switch, PFFC calculates its FID and increments the corresponding FBU counter.

Each flow has its own buffer usage threshold $X_{off_f}$ (the subscripted $f$ represents the flow ID of a specific flow). When the FBU counter of a specific flow exceeds $X_{off_f}$, a PAUSE frame is sent to the upstream node to pause the flow. To avoid overflowing the system buffer of the switch, PFFC also uses $X_{off_s}$ and $X_{on_s}$ to trigger the sending of PAUSE and RESUME frames to the upstream node. As shown in Fig. 6, PFFC uses the Packet Buffer usage (PBU) counter (Fig. 6 (E)) to track the usage of Packet Buffer of a switch. The PBU counter is incremented by every incoming packet and its value is stored in a register. The design and implementation of PFFC is based on both IEEE 802.3x and IEEE 802.1Qbb (PFC) standards. The $X_{off_s}$ threshold used in PFFC can prevent packet loss in the packet buffer effectively since $X_{off_s}$ plays the same role as $X_{off}$ used in the link-level lossless standard 802.3x.
switch due to its hardware limitation. To avoid the race condition problem [35] caused by concurrent accesses to a shared register in different stages of the pipelines, the P4 switch does not allow a register to be accessed in multiple stages of its pipelines. Because both IMAP and EMAP operate in parallel, a register cannot be accessed in both of them to avoid race condition problems. Due to this constraint, we cannot directly decrease the FBU and PBU counters in EMAP, which increases the difficulty of per-flow buffer usage accounting in PFFC.

To overcome this hardware restriction, PFFC uses the recirculation [36] mechanism to decrement the FBU and PBU counters. The detailed steps are shown in Fig. 6. When a packet has been retrieved from TMPB and is processed in EMAP, PFFC makes a cloned copy of the packet. After a packet is cloned, the cloned copy is temporarily stored in Mirror Buffer (Fig. 6 (G)), where it will enter TMPB. When the cloned copy leaves TMPB and enters EMAP, PFFC recirculates it by sending it to Recirculation Port (Fig. 6 (H)), where the cloned copy will enter IMAP. To save the usage of Mirror Buffer and the internal bandwidth used for the recirculation, PFFC truncates the packet payload of the cloned copy before recirculating it. The size of the original packet is stored in an additional header that PFFC inserts into the truncated packet. This information goes with the truncated packet to IMAP and is used to decrement the PBU counter and the FBU counter of the flow as

\[ FBU[FlowID] - = PacketSize \text{ and } PBU - = PacketSize. \]

After decrementing the counters, PFFC drops the truncated packet from IMAP (Fig. 6 (I)). The updated counter values are then compared with the \( Xon_s \) and \( Xon_f \) thresholds. If FBU is less than \( Xon_f \), PFFC will send a RESUME frame to the upstream node of the flow to resume the PEQ that is shared by the flow. If PBU is less than \( Xon_s \), PFFC will send a RESUME frame to the upstream node of the flow to resume both the DEQ and the PEQ shared by that flow.

C. PER-FLOW PAUSE/RESUME FRAMES GENERATION AND PROCESSING

The P4 language does not support any primitive action function that can be used to pause or resume an egress queue. Thus, PFFC needs to generate and send PFC-compliant PAUSE or RESUME frames to an upstream node to pause or resume an egress queue. Since the P4 switches (and most commodity switches) support PFC, PFFC directly uses existing PFC functionality in real time.

Fig. 7 shows the standard PFC control frame format. The time fields are used to store the time of the pause duration for the eight different CoSes. When the pause duration for a specific CoS ends, the paused CoS automatically resumes. The unit of the pause time is quanta, which is 512-bit time on a used link. Because the link bandwidth used in our experiments is 10 Gbps, a bit time is 0.1 nanosecond and a quanta is 51.2 nanoseconds. The maximum value for the time field is 65,535. Thus, the maximum pause time on the used link is about 3.4 milliseconds. In our experiments, we let PFFC and PFC use a half of the maximum value as the pause time, which is about 1.7 milliseconds. In PFC, actually no Opcode value is defined to represent a RESUME frame. When a time field is set to 0, it indicates to the upstream node that the corresponding CoS should be resumed even if its pause time is not over yet.

Although PFFC directly uses PFC to pause or resume an egress queue in the upstream node, this capability alone is not enough for PFFC to notify the upstream node to migrate a specific flow. The downstream node needs to convey the FID of the flow to be paused or resumed to the upstream node. However, from Fig. 7 there is no field defined for this purpose. Although there are some padding bytes that may be used to store the FID, we observed that when a PFC control frame is received by an upstream node, the control frame is “absorbed” (i.e., directly processed and then dropped) by the port module without entering the IMAP of the upstream node. This means that although our P4 program can generate PFC control frames to pause or resume the PEQ in the upstream node that a specific flow is mapped to, our P4 program still needs to generate and send our own control frames to the upstream node to convey the FID information. Since this user-defined control frame will not be “absorbed” by the port module but instead will be processed by our P4 program, we can use this information to migrate a specific flow in the upstream node.

Note that in PFFC all nodes use the same function to map the FID of a flow to the index of the shared PEQ that it uses. As a result, to pause a specific flow in the upstream node, the downstream node can compute the FID and its corresponding index to know which shared PEQ in the upstream node is used by a specific flow. It then sets the corresponding time field in the generated PAUSE frame to the intended pause duration and sends the PAUSE frame to the upstream node to pause the PEQ shared by the flow.

After performing the above step, the downstream node immediately sends a PFFC MIGRATE control frame to the upstream node to migrate the flow from the DEQ to this shared PEQ. To convey FID to the upstream node for flow migration, the downstream node generates and sends the PFFC control frame to the upstream node. Fig. 8 shows the format a custom PFFC control frame. The Flow ID field stores the FID of the flow to be migrated in the upstream node.
node. There are two types of migration: migrate and migrate-back. The migrate type requests the upstream node to migrate the specified flow from the DEQ to the PEQ shared by the flow. The migrate-back type requests the upstream node to migrate the flow back to the DEQ. As soon as receive the MIGRATE control frame, the migration is completed immediately since it simply changes the assignment of an egress queue to a flow (from DEQ to PEQ or in reverse).

To pause a specific flow in the upstream node, the downstream node sends a PAUSE frame (which is a PFC control frame with a non-zero timeout value) followed by a MIGRATE frame (which is a PFFC control frame of the migrate type). To resume a specific flow, the downstream node first sends a RESUME frame (which is a PFC control frame with a zero timeout value) and then immediately sends a MIGRATE-BACK frame (which is a PFFC control frame of the migrate-back type). The order of the two operations is particularly chosen to avoid packet reordering during the tiny transition period.

Upon receipt of the MIGRATE-BACK control frame for a paused flow, the flow is migrated immediately and the output port of new incoming packets for the flow is set to DEQ. However, it is possible that the PEQ gets paused again before all packets of the migrated flow in the PEQ have been drained from the PEQ. In this case, the new packets of the migrated flow may be forwarded out of the switch via the DEQ before the old packets in the PEQ. In our experiments the average time period between a PEQ resume and the subsequent PEQ pause is larger than 68.8 microseconds, and this congestion spreading situation occurred infrequently, as evidenced by the measured 0.03% packet reordering ratio.

D. THE MANY-TO-ONE MAPPING ISSUE
In a network switch, the number of egress queues per output port is limited. This paper considers eight egress queues in a 10GbE port and 32 queues in a 100GbE port. Due to this hardware constraint, PFFC may have to map multiple flows to the same shared PEQ. Consequently, the same PEQ may be resumed for any of the flows that share it, causing a flow that should be paused for a longer time to be resumed earlier. To address this many-to-one issue, PFFC uses Xoff and Xon to control and protect the system buffer of a switch from overflowing. Therefore, even though a tiny number of packets may “leak” to the downstream node due to this phenomenon, they will be safely queued in the downstream node without being dropped. If the buffer usage of the flow in the downstream node is already higher than Xoff, when receiving these leaked packets, the downstream node will keep sending PAUSE frames to the upstream node for the flow. As a result, even though a paused flow may be resumed earlier by other flows, it will be paused again immediately. Our experimental results (see Section IV) show that PFFC can effectively pause hundreds of flows via the seven shared PEQs without causing any packet loss.

E. THRESHOLDS SETTINGS FOR PFC AND PFFC
Although PFC can protect an ingress queue from overflowing, Section II-D has described a situation in which packets may still be dropped due to buffer overflow in an egress queue. In contrast, PFFC does not have this problem as it uses only the egress perspective to account and control the buffer usage of a flow and the switch.

The proper settings for PFC Xoff and Xon depend on the link bandwidth and the delays between an upstream and a downstream node, which include the signal propagation time.

![FIGURE 7. The format of the PFC control frame](image)
delay of the link (RTT) and the packet processing delay. Such dependencies are universal. However, since the packet processing delay may vary for different types of hardware switches, and the link RTT may vary for cables of different lengths, the $Xoff_f$ and $Xon_s$ parameter values should be selected through measurements for different types of hardware.

The best value for PFFC $Xoff_f$ is the minimal headroom buffer size that can absorb all the inflight packets at line rate (i.e., the link bandwidth) to avoid buffer overflow. The best PFFC $Xon_s$ value is the minimal size of packets in the egress queue that can maintain 100% link utilization after the RESUME packet has been sent to the upstream node to resume its packet transmission. In this paper, we use a set of iterative tests (in Section IV-B) to find proper settings for $Xoff_f$ and $Xon_s$ to cover most traffic patterns. In our experiments, the best $Xoff_f$ value is 2.24 MB and the best $Xon_s$ value is 1.2 MB.

The same values for $Xoff_f$ and $Xon_f$ can be used for all flows. In our experiments, if the switch has a total buffer space $B$ and the number of flows passing it is $N$, we set $Xoff_f$ and $Xon_f$ to $3B/N$ and $B/N$ respectively for all flows. This policy allows a flow to have a buffer oversubscription ratio of 3:1. Since using the $Xoff_f$ and $Xon_s$ mechanism already guarantees that no packets will be dropped due to buffer overflow, using 3:1 as the buffer oversubscription ratio allows the system buffer of the switch to be more efficiently utilized. Alternatively, different sets of values can be used for different flows based on their bandwidth requirements or their relative priorities.

### IV. PERFORMANCE EVALUATION

We have implemented PFFC in Inventec D10056 [13] P4 switches. These switches have 3.2 Tbps or 6.4 Tbps switching capacity, 32 1x100G ports (or 4x25G or 4x10G ports), and 21.3 MB of Packet Buffer memory. Experimental results show that PFFC can avoid or mitigate serious problems with PFC such as congestion spreading, deadlock, and packet loss. We also evaluated the bandwidth overhead, performance, and scalability of PFFC on multi-hop network topologies. The tested traffic patterns include multi-flow, multi-protocol (TCP vs. UDP), and multi-flow-type (elephant vs. mice) traffic patterns. The performance metrics include throughput, packet drop rate, flow completion time, link utilization, and bandwidth overhead. In summary, experimental results show that PFFC outperforms PFC in many aspects.

#### A. EXPERIMENTAL SETTINGS

Five hosts are used in the experiments and each has a 12-core Intel 3.2 GHz i7-8700 CPU and 16 GB RAM with Intel X710 network interface cards. These network interface cards support PFC. The hosts run Ubuntu Linux 18.04 operating system. Since these network interface cards only support 10 Gbps, we used a 40G QSFP+ to 4xSFP+ breakout optic cable to transform a 40G QSFP+ port of a switch to four 10GbE ports and then connect each 10GbE port to a host. The P4 switch has eight ingress queues for every 10GbE port. We use iperf version 2.0.13 [37] to generate UDP and TCP traffic in our experiments.

To make a fair comparison between PFC and PFFC, the buffer allocation for PFC and PFFC should be configured using the same structure. Fig. 9 shows the buffer allocation structure used by the P4 switches [16]. We denote the Packet Buffer space as $B$, which is about 21.3 MB for the used P4 switches.

![Buffer Allocation Structure](image)

FIGURE 9. The buffer allocation structure used by the P4 switches used in our experiments.

In Fig. 9, the buffer space is divided into two parts: one for ingress traffic ($B_i$) and the other for egress traffic ($B_e$), and each has $B/2$ buffer space.

$$B = B_i + B_e = B/2 + B/2 \quad (1)$$

The overall buffer space of the switch can be flexibly allocated to different ports and queues in a P4 switch or other hardware switches [17]. Here, we assume that the switch needs to process packets that may come from any input port and depart from any output port. For handling such a traffic pattern, we evenly divide the overall buffer space of the switch for each queue in each port for both ingress and egress traffic. In a 32-port switch, each port is allocated a buffer of size $B_p$, for its ingress traffic and egress traffic, respectively. From Eq. (1),

$$B_p = B_i/32 = B_e/32 = B/64 \quad (2)$$

Every ingress (egress) queue is allocated a buffer of size $B_q$ in the evenly distributed buffer allocation. Because there are 8 ingress (and 8 egress) queues in each 10GbE port, from Eq. (2), we have

$$B_q = B_p/8 = B/512 \quad (3)$$

In our experiments, we used $64B_q$ as the upper bound of buffer size for each egress queue in PFC and this means that we used a buffer oversubscription ratio of 64:1 for each egress queue in PFC. Compared with using $B_q$ for an egress queue, this high oversubscription ratio can greatly reduce the chances of packet dropping in an egress queue when PFC is used. However, as will be shown later, even using such a high
oversubscription ratio, some packets were still dropped in a tested scenario when PFC was used.

We note that the specification [16] just explains the usage of the above buffer allocation structure and does not provide any recommendation about how to allocate buffer space to each port and queue. The above static provisioning will not be optimal for some traffic patterns where only a few input ports or output ports of a switch have heavy traffic to process. We use this static provisioning to illustrate that packets may be dropped in PFC if the configured buffer provisioning does not fit the traffic patterns.

**B. THRESHOLD SELECTION FOR PAUSE AND RESUME**

Before conducting experiments comparing PFC and PFFC, we first performed a series of tests to find a proper value for $X_{offs}$ to prevent packet loss in PFFC. The threshold selection for pause and resume is BDP (Bandwidth-Delay Product) dependent. The BDP depends on the link bandwidth and the RTT between nodes. Although it is difficult to accurately measure the BDP inside the switch, we still can approximate the maximal value of BDP. The maximal value of BDP is limited by hardware and is agnostic to traffic patterns. Fig. 10a shows the network topology (Topology I) that we used to explore a knee $X_{offs}$ value, which is the maximal value that does not cause packet loss.

![FIGURE 10. The network topologies used in our experiments](image)

We considered many-to-one (from 14:1 to 112:1) traffic patterns in this experiment. There are 4 sending hosts ($N_{S1}$ – $N_{S4}$), 1 switch, and 1 receiving host ($N_{R1}$). Each sending host has 4 ports and the receiving host has only one port. This configuration supports up to 16 (4 hosts * 4 ports/host) packet generators, each of which is a software program using a different port on the sending hosts to send traffic to the single receiving host. The total number of flows from these sending hosts to the receiving host was varied from 14 to 112. This was achieved by each packet generator launching 7 UDP flows (each using a different CoS) to the receiving host and we varied the number of used packet generators from 2 to 16. The sending rate of each flow was set to 1 Gbps and the duration was 10 seconds. The reason why using UDP flows in this test was to ensure that these packet generators could steadily generate traffic at a fixed rate.

The buffer space for the output port that connects to $N_{R1}$ was set to $64B_q$ as defined in Section IV-A, which is 21.3/8 = 2.66 MB. Since all flows went through the same output port, for the buffer of this output port, we tested a range of threshold values for its $X_{offs}$ and only show the five values that are most close to the knee value in the figure. The five different $X_{offs}$ values are 2.56 MB, 2.48 MB, 2.4 MB, 2.32 MB, and 2.24 MB, respectively. Fig. 11 shows the packet loss rates versus different numbers of flows (and the used ports) under these five threshold settings over the Topology I of Fig. 10.

![FIGURE 11. The packet loss rate vs. number of flows for $X_{offs}$ threshold exploration. Each data point is the average value of 30 runs.](image)

When the threshold value is lower than 2.24 MB, there is no packet loss. It means that the $0.42 – 2.66 – 2.24$ MB headroom space is large enough to absorb inflight packets of the 112 UDP flows. Thus, in the following experiments, we used the value 2.24 MB for the $X_{offs}$ threshold so that no packet will be dropped.

We also explored the minimum $X_{ons}$ value that does not cause any degradation of the utilization of the link between the switch and $N_{R1}$. Fig. 12 shows the experimental results. We consider five candidates $X_{ons}$ values and compare the link utilization under four different numbers of flows: 14, 35, 77, and 112, respectively. The link utilization decreases a bit when the $X_{ons}$ value is less than or equal to 0.8 MB. When the $X_{ons}$ value is greater than or equal to 1.2 MB, 100% link utilization is achieved. Since this threshold should be as small as possible on the condition that the link utilization can maintain 100%, in all of our following experiments, we chose 1.2 MB as the $X_{ons}$ threshold.

The reason to find a maximum value for $X_{offs}$ without dropping any packet and a minimum value for $X_{ons}$ without lowering the link utilization is because in this way the frequency to cross these thresholds can be greatly reduced. As a result, the number of generated PAUSE and RESUME frames and thus the consumed link bandwidth for these frames can be greatly reduced. In Section IV-H, our experimental results will compare the link bandwidth consumed by the control frames generated in PFC and PFFC, respectively.

C. SCALABILITY OF PFFC ON MULTI-HOP NETWORK TOPOLOGIES

Since PFFC is a per-hop per-flow scheme and will be used over multi-hop network topologies, it is important to see when PFFC is enabled whether the throughput of a flow may
The purpose of using TCP flows for this evaluation is to see whether PFFC may cause any packet loss or reorder during its operations. Due to the design of TCP congestion control algorithm, if any TCP packet is dropped, the TCP sender will immediately cut its sending rate by half to reduce its current sending rate. In addition, if there are packet reorderers that cause more than three duplicate ACK packets to be sent to the TCP sender, the TCP sender will view that some of its packets have been dropped and thus also cut its current sending rate by a half. In short, a TCP flow is very sensitive to packet loss and reorder and its achieved throughput over time will fluctuate when some of its packets are intermittently dropped or reordered. From the results shown in Fig. 14, one sees that the average throughputs of TCP flows over different numbers of hops are very stable over time. The extent of packet reordering and retransmission are shown in Table 2. The reordering rate is the number of the reordered packets (in bracket) divided by the number of the packets sent. The extent of reordering is about 0.03% and the retransmission rate is less than 1% when the number of hops is up to 4. These results show that PFFC can serve TCP flows quite well.

### Table 2. The extent of reordering and retransmission of TCP flows on multi-hop topologies

| Hops | Reordering rate | Retransmission rate | Packets sent |
|------|----------------|---------------------|--------------|
| 1    | 0.00% (0)       | 0.00% (0)           | 8,153,364    |
| 2    | 0.00% (100)     | 0.00% (31)          | 8,153,364    |
| 3    | 0.02% (1,474)   | 0.01% (581)         | 8,158,518    |
| 4    | 0.03% (2,466)   | 0.01% (773)         | 8,153,364    |

### D. PACKET LOSS IN MULTI-FLOW SCENARIOS

This subsection evaluates another scalability of PFFC to see whether PFFC can handle a large number of flows without dropping their packets. The network topology is shown in Fig. 10a (Topology I). As we did in Section IV-B, we varied the number of UDP flows from 14 to 112 and the sending rate of each flow was set to 1 Gbps. (Note that the 7 UDP flows originated from the same packet generator used different CoSes.) We compared the packet loss rate of PFFC with that decrease as the number of hops increases. For this evaluation, we used the network topologies shown in Fig. 13.

![Multi-hop topologies: the number of hops \(N\) is ranged from 1 to 4.](image)

As can be seen, there are four topologies and they have 1, 2, 3 and 4 hops (i.e., intermediate switches) connected in a row, respectively. A sending host \((N_{S1})\) sends traffic to the receiving host \((N_{R1})\). On each topology, we connected another sending host \((N_{S2})\) to Switch N so that the traffic of \(N_{S1}\) and \(N_{S2}\) merged in Switch N and generated congestion over the link from Switch N to \(N_{R1}\).

In the experiment, each of \(N_{S1}\) and \(N_{S2}\) used iperf to generate 7 TCP flows each sending at 1 Gbps for 10 seconds. The total sending rate of these 14 flows exceeded the bandwidth of the 10 Gbps link between Switch N and \(N_{R1}\) and caused congestion immediately. The optimal average throughput of these TCP flows should be 10 Gbps / 14 = 714 Mbps. Fig. 14 shows that the average throughputs of these TCP flows over different numbers of hops are almost the same. These results show that the overhead of the design and implementation of PFFC is very small and thus the flow-controlled throughput of a TCP flow does not degrade when it traverses more and more hops on a network. From Fig. 14, one sees that there is a small difference between the average throughput of these TCP flows and the optimal value. This is because TCP header size is 20-byte long and a TCP header usually carries several TCP options. These headers and options consume the link bandwidth and cause the application-level average throughput of TCP flows to be slightly less than the optimal value.
of PFC. We set $X_{off}$ to 2.24 MB and $X_{on}$ to 1.2 MB for the PFFC experiments. For PFC, we allocated $64B_q$ (2.663 MB) buffer space to the egress queue of the output port that connects to $N_{R1}$ and $B_q$ (0.042 MB) buffer space to each ingress queue.

Experimental results show that no packet was dropped in PFFC when the number of flows was varied from 14 to 112. This shows that PFFC can totally avoid packet dropping after its $X_{off}$ and $X_{on}$ thresholds have been calibrated. As for PFC, its packet loss rates under different numbers of flows are shown in Fig. 15.

![FIGURE 15. Packet loss rate vs. number of flows in PFC](image)

When PFC was used, some packets started to be dropped when the number of flows went beyond 59. We have explained why packets may be dropped in PFC in Section II-D. Because in this experiment the egress queue used a buffer oversubscription ratio of 64:1, in theory as long as the number of flows does not exceed 64, PFC will not drop any packet. On the other hand, when the number of flows exceeds 64, some packets will be dropped without triggering the PFC flow control. However, even if the number of flows was smaller than 64, some packets may still be dropped.

Although in Fig. 15 there was some inconsistency in the boundary conditions, outside the boundary conditions, the measured packet loss rate under a specific number of flows in PFC is very close to what it should be. When there are $N$ flows, since the sending rate of each flow is 1 Gbps, the total sending rate of these flows is $N$ Gbps. Because the bandwidth of the bottleneck link is 10 Gbps, a $(N - 10)/N$ percentage of sent packets will be dropped in the egress queue. For example, when $N$ is 70, $(N - 10)/N$ is 85.7%, which is very close to the measured packet loss rate.

### E. FLOW COMPLETION TIME OF MICE FLOWS

Due to the fast flow control over hops, compared with end-to-end based methods, per-hop based flow control is considered a suitable scheme to handle mice flows when congestion occurs. This is because an end-to-end congestion control scheme requires at least a RTT to take effect; however, all of the packets of a mice flow may have been sent into the network within a RTT. In this experiment, we compared the performance of PFFC and PFC in processing mice flows using flow completion time (FCT) as the performance metric. The network topology was configured as shown in Fig. 16. There are two sending hosts ($N_{S1}$ and $N_{S2}$), three switches, and one receiving host ($N_{R1}$). The bandwidth of all links is 10 Gbps.

![FIGURE 16. The network topology used for evaluating mice flows mixed with elephant flows](image)

We considered the request/reply (mice flow) and file transfer traffic patterns. Each of $N_{S1}$ and $N_{S2}$ launched 7 UDP elephant flows sending at 1 Gbps to $N_{R1}$ for 10 seconds to create congestion in Switch 1. After the first second, in each second of the following 5 seconds, each of $N_{S1}$ and $N_{S2}$ launched 56 mice flows to send a file to $N_{R1}$, respectively. Each mice flow is a TCP flow and its size is 10 KB. Fig. 17 shows the 1st, 25th, 50th, 75th, and 95th percentile FCT over these mice flows when using PFC and PFFC as the flow control scheme, respectively. As can be seen, the 1st, 25th, 50th, 75th, and 95th percentile FCT of PFFC are shorter than those of PFC.

![FIGURE 17. The flow completion time of TCP mice flows mixed with UDP elephant flows](image)

The reason why the average FCT of mice flows in PFFC is shorter than that in PFC is because in PFC there are only eight CoSes and thus multiple mice flows may need to be mapped to the same CoS. When PFC pauses a queue of a specific CoS, all of the flows using that queue are paused, including mice flows. In contrast, in PFFC each mice flow is paused and resumed independently without being affected by other flows. As a result, the average FCT of mice flows in PFFC is shorter than that in PFC.

Fig. 18 shows the results of the second experiment, which used TCP flows as the elephant flows. As in the first experiment, the sender of each TCP elephant flow sent its traffic at 1 Gbps for 10 seconds. The results show that PFFC still
performed better than PFC for mice flows when the elephant flows were TCP flows.

![Flow Completion Time Graph](image)

**FIGURE 18.** The flow completion time of TCP mice flows mixed with TCP elephant flows

However, comparing Fig. 17 with Fig. 18, one can see that the performance gains of PFFC over PFC in supporting mice flows were slightly reduced when the elephant flows were TCP flows. The reason is that in an upstream node, received TCP ACK packets delayed the processing of received PFFC control frames, thus increasing the FCT of mice flows in PFFC.

### F. CONGESTION SPREADING EVALUATION

As explained in Section II-B, congestion spreading can occur in PFC. We used the following experiment to show this phenomenon. The used network topology is shown in Fig. 10b (Topology II).

We purposely configured the bandwidth of the link between Switch 2 and \( N_{R2} \) to be only 1 Gbps and left other links to use their default bandwidth of 10 Gbps. We launched a 3 Gbps UDP flow from \( N_{S2} \) to \( N_{R1} \) (Flow 1) and another 3 Gbps UDP flow from \( N_{S1} \) to \( N_{R2} \) (Flow 2). Because the sending rate of Flow 2 is larger than the bandwidth of the link between Switch 2 and \( N_{R2} \), many packets of Flow 2 got queued in Switch 2.

Because both flows used the same default CoS in PFC, their packets were queued in the same CoS ingress queue of Switch 2. However, because many packets of Flow 2 were queued in Switch 2, this CoS ingress queue soon built up and PFC was triggered to constantly send PAUSE frames to Switch 1 to throttle the packet transmissions from Switch 1 to Switch 2. It is clear that Flow 1 should not be paused at any time as there was no bottleneck on its path and it should achieve 3 Gbps throughput. However, since Flow 1 and Flow 2 used the same default CoS, their packets were paused by Switch 1 at the same time.

In PFFC both flows achieved what they could achieve and none of them was a victim flow. That is, Flow 1 achieved 2990 Mbps while Flow 2 achieved 1000 Mbps. However, in PFC Flow 1 only achieved 1 Gbps, which was the same as that achieved by Flow 2, rather than 3 Gbps. These results show that congestion spreading occurred when PFC was used but did not occur when PFFC was used.

### G. DEADLOCK EVALUATION

In Section II-C, we stated that deadlock can occur in PFC but not in PFFC. We used the network in Fig. 19 to show this phenomenon. In this experiment, we launched 3 UDP flows: \( F_1 \), \( F_2 \), and \( F_3 \) from hosts: \( N_1 \), \( N_2 \) and \( N_3 \) respectively. All of them used the default CoS and the sending rate of each of them was set to 3 Gbps. The paths of these flows are shown in Fig. 19, where the data path of \( F_1 \) is \( N_1 \rightarrow Switch 1 \rightarrow Switch 2 \rightarrow Switch 3 \rightarrow N_3 \), the data path of \( F_2 \) is \( N_2 \rightarrow Switch 2 \rightarrow Switch 3 \rightarrow Switch 1 \rightarrow N_3 \), and the data path of \( F_3 \) is \( N_3 \rightarrow Switch 3 \rightarrow Switch 1 \rightarrow Switch 2 \rightarrow N_2 \). Suppose that there is one queue per output port in every switch. In PFC, it is possible that the queue of Switch 1 is fully occupied by, for example, the packets of flow 1, where I can be 1, 2 or 3. Then Switch I cannot receive any packet from the other two flows and the circular wait deadlock situation occurs. In PFFC, the buffers for flows \( F_1 \), \( F_2 \), and \( F_3 \) in the DEQ of Switch I are denoted by \( DEQ_{I,1} \), \( DEQ_{I,2} \), \( DEQ_{I,3} \) respectively. Suppose that the DEQs in the three switches are full, and all flows are paused. That is, all flows are put in the PEQs of the switches. In this case, all sending hosts are also paused and cannot send packets out. If no further action is taken, then the network also encounters deadlock. Fortunately, the hosts \( N_1 \), \( N_2 \) and \( N_3 \) can continuously receive the packets from these switches without any pause. For example, \( N_3 \) can receive the packets from \( DEQ_{3,1} \) of Switch 3, and therefore, \( DEQ_{3,1} \) will have more vacant space to receive packets from Switch 2.

In other words, Switch 3 will resume \( F_1 \) at Switch 2, and \( F_1 \) will be migrated from the PEQ of Switch 2 to DEQ, and the packets of \( F_1 \) can be sent from Switch 2 to Switch 3. Similarly, Switch 2 will resume \( F_1 \) at Switch 1, and Switch 1 will resume \( F_1 \) at \( N_1 \). Therefore, no circular wait deadlock situation will occur with the DEQ/PEQ mechanism in PFFC. We claim that the DEQ/PEQ mechanism works for \( I > 3 \) with various flow paths. We will validate our claim in mathematical derivation and conduct experiments in a large-scale P4 network in the future.

Now, consider another example. To generate congestion in the network, we deliberately set the bandwidth of the unidirectional link from Switch 2 to \( N_2 \) to 1 Gbps; the bandwidth of all other links was set to their default 10 Gbps.

![Network Topology Graph](image)

**FIGURE 19.** The network topology used for showing PFC deadlock problems
At the beginning of the experiment, we launched $F_1$. After the first second, we launched $F_2$. Now the achieved throughputs of both flows are the same as their sending rates. After the second second, we launched $F_3$. Since the last hop of $F_3$ was the link from Switch 2 to $N_2$ and its bandwidth was limited to only 1 Gbps, many packets of $F_3$ immediately got queued in Switch 2. At this time, Switch 2 sent PAUSE frames to Switch 1 to stop any further packet transmission. Because all flows used the same CoS, Switch 1 also paused the transmission of the packets of $F_1$ on the link from Switch 1 to Switch 2. Now the packets of $F_1$ got queued in Switch 1, which caused Switch 1 to send PAUSE frames to Switch 3 to stop any further packet transmission. (Note that Switch 1 also sent PAUSE frames to $N_1$ to stop any further packet transmission of $F_1$.) This caused the packets of $F_2$ and $F_3$ to be queued in Switch 3, and Switch 3 then sent PAUSE frames to Switch 2 to pause any further packet transmission. Switch 3 also sent PAUSE frames to $N_3$ to block its packet transmission. As many packets got queued in Switch 2, Switch 2 sent PAUSE frames to $N_2$ to stop any further packet transmission. At this moment, all of the flows were blocked and the deadlock has been formed.

Fig. 20 shows the dynamic interactions among the three flows in PFC and PFFC. As can be seen, deadlocks occurred in PFC but did not occur in PFFC. This is because flow control is applied in a per-flow manner in PFFC and thus a flow does not affect other flows.

### H. CONTROL FRAME BANDWIDTH OVERHEAD OF PFC AND PFFC

Both PFC and PFFC send PAUSE and RESUME control frames on links to perform per-hop flow control. The link bandwidth consumed by their control frames is their bandwidth overhead. Here, we conducted an experiment to compare their bandwidth overhead. The network topology used in the experiment is shown in Fig. 10a. In the experiment, we generated 56 UDP flows. Each run lasted 10 seconds and each performance number reported below is the average over ten runs.

Table 3 shows the bandwidth overhead of PFC and PFFC in terms of: 1) control frame count, 2) control frame Mbyte count, and 3) the percentage of link bandwidth consumed by control frames. As can be seen, the bandwidth overhead of PFFC was only slightly higher than that of PFC. Since PFFC generates PFC frames to pause and resume flows, we note that the statistics of PFFC reported in this table include its generated PFC frames. Given that PFFC outperforms PFC in many aspects, we think that the added small bandwidth overhead of PFFC is acceptable.

|               | Frame count | Mbyte count | Overhead over link (%) |
|---------------|-------------|-------------|------------------------|
| PFC           | 156,461     | 100         | 0.40                   |
| PFFC          | 211,010     | 135         | 0.54                   |

### I. MEMORY FOOTPRINT OF PFFC

We maintain the following states using P4 registers, which use SRAM in match-action pipelines.

- System buffer usage (the PBU counter): 32 bits.
- Per-flow buffer usage (the FBU counters): 32 bits for each flow.
- Flow state (paused or resume): 8 bits for each flow.
- Flow-to-queue mapping (which flow in which PEQ): 8 bits for each flow.

In a P4 switch, every 32-bit register is stored in two blocks of SRAM. Each block size of SRAM is $2^{16}$ bits. For tracking 4096 flows, we need 8 blocks to bookkeep PBU (2 blocks), FBU (2 blocks), flow state (2 blocks) and flow-to-queue mapping (2 blocks). It occupies 1.39% SRAM space of a P4 switch. For tracking 40,960 flows, we need 13.9% of SRAM space. We can optimize the memory usage by storing flow state and flow-to-queue mapping data in a single 32-bit register and reduce the usage of SRAM for 40,960 flows to 10% of SRAM space.

### V. DISCUSSION

The design and implementation of PFFC presented in this paper are only a proof-of-concept demonstration showing the potential of our method. There are still several places that can be further improved and we discuss them as follows.
First, the handling of potential collision of hash function used for generating Flow ID should be addressed in the future (we expect that a CRC-32 collision occurs with a probability of \( 2^{-32} \)). If the generated Flow IDs for two flows are the same, PFFC will use the same FBU counter to count the buffer usages of both flows and these flows will share the same PEQ. As a result, these flows may be paused more often than necessary. To address this issue, PFFC can use the solution proposed in [38], which uses hash functions to identify flows and stores the states of up to 1 million flows in a P4 switch. According to [38], using a 16-bit hash digest, the chance of collision can be reduced to only 0.01\% and can be resolved with a marginal software overhead.

Second, in general packet size is not available in IMAP (at the time when packet headers are being processed there), which is the standard limitation for every high-speed pipelined switch. The current design of PFFC requires the switch to know the packet length at IMAP. This is possible with the IP protocol as there is a length field in the IP header. For an IP packet, PFFC uses an additional table to compute the length of the preceding headers and then adds it to the length carried in the IP header to obtain the packet length. For non-IP packets whose headers do not provide the length information, we may need to modify the design of PFFC to move the “get the packet length” logic to EMAP, where the packet length is available.

Other interesting research issues include: 1) support different priorities among flows; 2) guarantee a minimum bandwidth for a specific flow; 3) support tens of thousands of flows using large scale simulation; 4) study how different forms of attacks many affect the operations of PFFC. Right now, we are collaborating with Accton/Edgecore (one of the largest P4 switch vendors in the world) to build a large-scale P4 network (more than 40 P4 switches). In the future, we will conduct measurement experiments in this network to test new P4 mechanisms including PFFC with congestion control protocols like HPCO, DCQCN, DCTCP, and so on.

VI. CONCLUSION

In this paper, we designed, implemented, and evaluated the performance PFFC in P4 hardware switches. Compared with PFC, PFFC outperforms it in many aspects. Many serious problems with PFC such as congestion spreading, deadlock, packet loss, etc. are avoided or mitigated in PFFC. In addition, the average flow completion time of mice flows in PFFC is shorter than that in PFC. As shown in our experiments, PFFC works well over multi-hop network topologies. Regarding the link bandwidth consumed by the control frames of PFFC and PFC, the bandwidth overhead of PFFC is only slightly higher than that of PFC. With these great benefits and minor added bandwidth overhead, PFFC is a good candidate scheme to be used for lossless networks.

ACKNOWLEDGMENTS

This work was financially supported by the Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan. We thank Barefoot/Intel, Accton, and National Cheng Kung University for their supports of this work.

REFERENCES

[1] “IEEE Standard for Ethernet,” IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008), pp. 1–374, 2012.
[2] “IEEE Standard for Local and metropolitan area networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment 17: Priority-based Flow Control,” IEEE Std 802.1Qbb-2011 (Amendment to IEEE Std 802.1Q-2001 as amended by IEEE Std 802.1Qae-2011 and IEEE Std 802.1Qbe-2011), pp. 1–40, 2011.
[3] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, and P. Grun, “Remote Direct Memory Access over the Converged Enhanced Ethernet Fabric: Evaluating the Options,” 2009 17th IEEE Symposium on High Performance Interconnects. IEEE, 2009, pp. 123–130.
[4] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, “RDMA over Commodity Ethernet at Scale,” Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 202–215.
[5] A. Mohammad and A. Berk and et al., “Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization,” 2008 46th Annual Allerton Conference on Communication, Control, and Computing, 2008, pp. 1270–1277.
[6] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, “Practical DCB for Improved Data Center Networks,” IEEE INFOCOM 2014-IEEE Conference on Computer Communications, IEEE, 2014, pp. 1824–1832.
[7] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Linor, J. Padhye, S. Papadakis, M. H. Yahia, and M. Zhang, “Congestion Control for Large-Scale RDMA Deployments,” ACM SIGCOMM Computer Communication Review, vol. 45, no. 4, pp. 523–536, 2015.
[8] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, “Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them,” Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 92–98.
[9] H. Shuihai, Z. Yibo, C. Peng, G. Chuanxiong, T. Kun, P. Jitendra and C. Kai, “Tagger: Practical PFC Deadlock Prevention in Data Center Networks,” Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies, 2017, pp. 451–463.
[10] K. Qian, W. Cheng, T. Zhang, and F. Ren, “Gentle Flow Control: Avoiding Deadlock in Lossless Networks,” Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 75–89.
[11] P. Goyal, P. Shah, K. Zhao, N. K. Sharma, M. Alizadeh, and T. E. Anderson, “Backpressure Flow Control,” Proceedings of the 2019 ACM Workshop on Buffer Sizing, 2019, pp. 1–3.
[12] Edge-Core AS4600-54T high-performance switch, accessed on July 4, 2021. [Online]. Available: https://www.edge-core.com/productsInfo.php?cids=1&cls2=9&cls3=4&id=19
[13] Inventec D10056 Datacenter programmable 100GbE switch, accessed on July 4, 2021. [Online]. Available: http://productline.inventec.comswitch/download/D10056.pdf
[14] The P4 language specification version 1.0.5, accessed on July 4, 2021. [Online]. Available: https://p4.org/p4-spec/p4-14/v1.0.5/pdf/p4.pdf
[15] Barefoot Tofino, accessed on July 4, 2021. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/
[16] Barefoot Tofino Switch Architecture Specification (Barefoot internal document), Tech. Rep., 2017.
[17] Broadcom Programmer’s Reference Guide: Theory of Operation (Broadcom internal document), Tech. Rep., 2014.
[18] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, “HPCC: High Precision Congestion Control,” Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 44–58.
[19] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, “Re-architecting Congestion Management in Lossless Ethernet,” 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 2020, pp. 19–36.
[20] R. Mittal, A. Shpiner, A. Panda, E. Zabahi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, “Revisiting Network Support for RDMA,” Pro-
proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 313–326.

[21] P. Goyal, P. Shah, K. Zhao, N. K. Sharma, M. Alizadeh, and T. E. T. Hamada and N. Nakasato, “Infiniband Trade Association,” Proceedings of the ACM SIGCOMM 2010 Conference. 2010.

[22] “IEEE DCB. 802.1Qaz - Quantized Congestion Notification.”

[23] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats “TIMELY: RRT-based Congestion Control for the Datacenter,” ACM SIGCOMM, 2015.

[24] K. Gautam, et al. “Swift: Delay is Simple and Effective for Congestion Control in the Datacenter,” Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. 2020.

[25] V. Vasudevan, A. Phanishayee, H. Shah, E. Krovat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, “Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication,” ACM SIGCOMM Computer Communication Review 39.4 (2009): 303-314.

[26] H. Wu, Z. Feng, C. Guo, and Y. Zhang, “ICTCP: Incast Congestion Control for TCP in Data-Center Networks,” IEEE/ACM Transactions on Networking 21.2 (2012): 345-358.

[27] A. Singh, et al. “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network,” ACM SIGCOMM Computer Communication Review 45.4 (2015): 183-197.

[28] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, “High-Speed Switch Scheduling for Local-Area Networks,” ACM Transactions on Computer Systems (TOCS), vol. 11, no. 4, pp. 319–352, 1993.

[29] H. T. Kung and R. Morris, “Credit-Based Flow Control for ATM Networks,” IEEE Network, vol. 9, no. 2, pp. 40–48, 1995.

[30] T. Hamada and N. Nakasato, “Infiniband Trade Association, Infiniband Architecture Specification, volume 1, release 1.0,” http://www.infinibandta.com, “International Conference on Field Programmable Logic and Applications, 2005. Citeseer, 2005.

[31] P. Goyal, P. Shah, K. Zhao, N. K. Sharma, M. Alizadeh, and T. E. Anderson, “Backpressure Flow Control,” arXiv:1909.09923v3 [cs.NI], 23 Oct 2020.

[32] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the Social Network’s (Datacenter) Network,” Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. 2015.

[33] P. Bossart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-independent Packet Processors,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87–95, 2014.

[34] A. Silberschatz, G. Gagne, and P. B. Galvin, Operating System Concepts, 10th Edition. Wiley, 2018.

[35] R. Ben-Basat, X. Chen, G. Einziger, and O. Rottenstreich, “Efficient Measurement on Programmable Switches Using Probabilistic Recirculation,” 2018 IEEE 26th International Conference on Network Protocols (ICNP) (pp. 313-323). IEEE.

[36] iPerf - The Ultimate Speed Test Tool for TCP, UDP and SCTP, accessed on July 4, 2021. [Online]. Available: https://iperf.fr/

[37] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs,” Proceedings of the ACM Special Interest Group on Data Communication, 2017, pp. 15-28.

YI-BING LIN (M’96-SM’96-F’03) is Winbond Chair Professor of National Yang Ming Chiao Tung University (NYCU), Chair Professor of China Medical University, National Cheng Kung University and Asia University. During 2014 - 2016, Lin was Deputy Minister, Ministry of Science and Technology, Taiwan. Lin is AAAS Fellow, ACM Fellow, IEEE Fellow, and IET Fellow.

YO-RU CHEN currently is a Ph.D. candidate under the supervision of Prof. Shie-Yuan Wang. His research interests include software-defined networks and P4 programmable networks.

JASON HSIEH currently is a graduate student under the supervision of Prof. Shie-Yuan Wang. His research interests include software-defined networks and P4 programmable networks.

RUEI-SYUN LAI currently is a graduate student under the supervision of Prof. Shie-Yuan Wang. Her research interests include software-defined networks and P4 programmable networks.