OFC: An Original congestion-based Fine-grained Priority Flow Control

Wenli Xiao[1], Yuqing Yang[1], Peirui Cao[1], Zhuoran Liu[1], Shizhen Zhao[1*], Xinbing Wang[2]
[1] John Hopcroft Center, Shanghai Jiao Tong University, Shanghai 200240, China
[2] Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
{xiaowenli, yyq2017, caopeirui, cocopromenade-9, shizhenzhao, xwang8}@sjtu.edu.cn

***Abstract*—With the proliferation of online data intensive applications and virtualized services, the growing complexity of traffic patterns in data centers increases the likelihood of congestion, especially in incast scenarios and with a combination of short and large flows. To ensure lossless transmission, RDMA over Converged Ethernet networks rely on Priority-based Flow Control (PFC) to prevent packet loss due to buffer overflow. However, it is widely acknowledged that PFC gives rise to several issues, such as Congestion Spreading, Head-of-Line Blocking, and Deadlock, which are increasingly prominent in modern highly congested data centers. In this paper, we analyze the primary causes of congestion spreading and head-of-line blocking issues associated with PFC and propose Original congestion-based fine-grained priority Flow Control (OFC) as a solution. The performance of OFC is assessed using the programmable switch Tofino and simulations carried out with a packet-level simulator across various scenarios, encompassing incast, realistic, deadlock, and in-depth scenarios. The validation of the simulation results through testbed evaluation confirmed that OFC effectively reduces flow completion time, buffer occupancy, and deadlock occurrence by up to 60.28%, 51.47%, and 48.7%, respectively.**

***Index Terms*—priority flow control, congestion spreading, head-of-line blocking, data stream, reliability**

I. Introduction

Modern data centers have increasingly embraced RDMA over Converged Ethernet version 2 (RoCEv2) to attain low latency and lossless transmission capabilities [1]–[4], recognizing that packet loss can lead to heightened latency [5]. RoCEv2 utilizes Priority-based Flow Control (PFC) [6] to optimize performance, facilitating hop-by-hop flow control to mitigate buffer oversaturation resulting from high upstream traffic. However, the growing prevalence of Online Data Intensive (OLDI) applications and virtualized services has led to increasingly complex traffic patterns within data centers, elevating the probability of congestion, particularly in scenarios involving incast, burst and a mix of short and large flows [7]–[9]. Consequently, issues related to PFC, including Congestion Spreading [10], Head-of-Line Blocking (HLB) [11], and Deadlock [12], have become more prominent in modern highly congested data centers.

The significant issues of PFC stem from its reliance on a local congestion-based coarse-grained pause scheme. To prevent buffer overflow, PFC triggers a pause in all upstream port transmissions if the ingress queue length of the downstream port exceeds the pause threshold. However, this method solely hinges on the ingress queue length to assess congestion, disregarding the distinction between original congestion and local congestions incurred from the congestion spread through hop-by-hop. Consequently, as the congestion level escalates, the problem of congestion spreading results in prolonged flow completion times [13]. Furthermore, the uniform pause action affecting all flows fails to differentiate between congested and uncongested flows, which exacerbates the HLB issue [14]. In highly congested networks, the diminished throughput of uncongested flows significantly impacts overall performance.

To address the concerns of congestion spreading and HLB issues, we conduct a comprehensive analysis focusing on their primary reasons. Our proposed approach centers on identifying the original congestion, which serves as the underlying cause of congestion spreading. It is imperative to inform all upstream ports about the original congestion as the true cause of congestion can become increasingly complex with the spread of congestion. Furthermore, to implement specific actions for congested and uncongested flows, a fine-grained pause scheme is essential. However, a key challenge lies in ensuring the isolation of different types of packets without inducing out-of-order delivery. Therefore, we outline two key objectives: (1) conveying information about the original congestion to all upstream ports; (2) implementing a fine-grained pause scheme.

To achieve the first objective, we utilize the fact that the ingress port of the original-congestion port is the first to send a PAUSE frame to the upstream port. We can ascertain whether a port is the original-congestion port by determining if it has received a PAUSE frame before. If a port triggers a PAUSE condition without having received a PAUSE frame, it is identified as the original-congestion port responsible for the congestion. Then, we include information about the congested flow in the PAUSE frame, ensuring that all upstream ports have access to the congested flow. To implement the second objective, a Nested Hierarchical Scheme is devised, which includes a normal queue and two backup queues. The normal queue functions accommodates all incoming packets of this priority. One of the backup queues is used to reserve paused flows, while the other remains empty to isolate the paused and resumed packets upon receiving the RESUME frame. The RESUME frame function as the Order Mark, facilitating the isolation of all types of packets and ensuring in-order delivery.

In summary, the issues related to PFC are mitigated by achieving the aforementioned two objectives. We name this approach as Original congestion-based priority Flow Control (OFC). Our contributions are listed below:

1) We analyze the primary causes of congestion spreading and head-of-line blocking issues, ultimately proposing general solutions to address these concerns raised by PFC. Our initial proposal involves the utilization of PAUSE frames to notify all upstream ports about the original congestion and a fine-grained pause scheme to achieve isolation without out-of-order delivery.
2) We introduce the Original congestion-based fine-grained priority Flow Control (OFC) as a feasible solution to the issues associated with PFC. OFC is capable of identifying the original congestion and implementing specific actions for congested and uncongested flows through a fine-grained pause scheme.
3) The performance of OFC is evaluated using the programmable switch Tofino and packet-level simulations under various scenarios, including incast, realistic, deadlock, and in-depth. The testbed evaluation confirms and validates the simulation results, all of which demonstrate that OFC effectively reduces flow completion time, buffer occupancy, and deadlock occurrence, with reductions of up to 60.28%, 51.47%, and 48.7%, respectively.

II. Analysis & Challenges of the Ideal PFC

We will analyze the reasons behind the issues with standard PFC and the challenges of their solutions. Standard PFC uses ingress queue length to prevent packet loss from buffer overflow, which can lead to congestion spreading. When the queue length crosses the pause or resume threshold, PFC pauses or resumes all flows, causing issues like HLB.

A. Local Congestion & Original Congestion

Standard PFC leads to congestion spreading by pausing based on local congestion instead of addressing the root cause. Preventing this requires addressing the original congestion.

Local Congestion: Ports with Congestion State. Local congestion refers to a specific area or segment within a network where network traffic encounters a bottleneck or overload, resulting in reduced performance or delays. This type of congestion can be identified using local information, such as the egress queue length of the current port, without the need for data from other ports in the network. However, it includes both congestion spreading and the root cause, so relying solely on local congestion for pause actions can lead to congestion spread issues, similar to standard PFC.

Original Congestion: The Root of Congestion Spreading. On the other hand, original congestion signifies the initial occurrence of network congestion, often leading to subsequent issues and performance degradation in interconnected areas of the network. Initially, this original congestion arises as local congestion within the network [15] and subsequently leads to the formation of additional local congested ports as the congestion spreads. This progression forms a congestion tree, with the original congestion acting as the root and the subsequent local congestions as the leaves. The hierarchy within the congestion tree, from the root to the leaves, is determined by the back-pressure mechanism of PFC. Identifying original congestion allows a clear understanding of the root cause of congestion and enables the reduction of congestion spread.

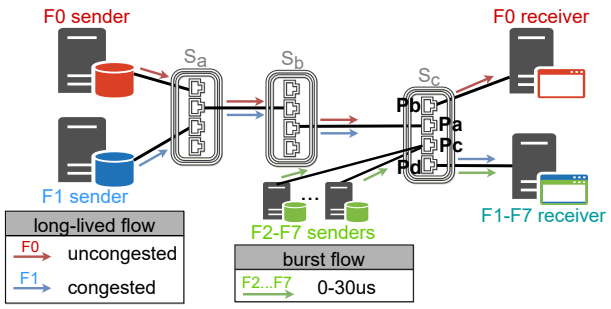


Fig. 1. An incast topology with uncongested flow F_0 and congested flow F_1 - F_7. Note that F_0 and F_1 are long-lived flows, F_2 - F_7 are burst flows lasting for 0 - $30us$.

Given the challenge of differentiating between original congestion (root) and congestion stemming from the upstream port (leaves) based solely on local information at the current hop, the exchange of information from downstream ports becomes crucial in identifying the source of the congestion.

B. Coarse-grained & Fine-grained Pause Scheme

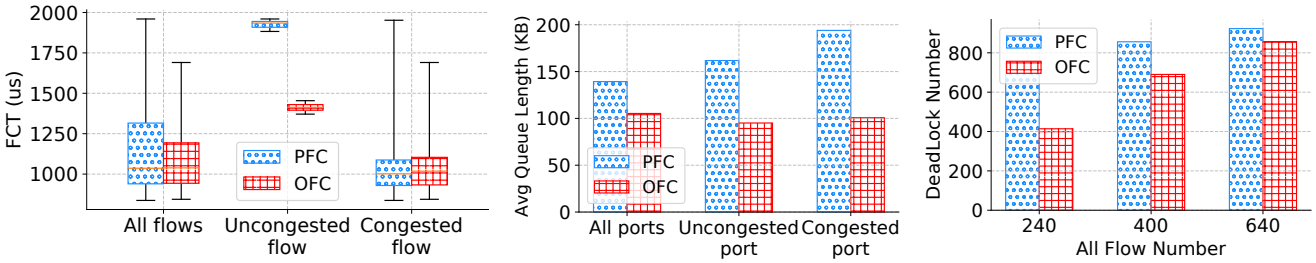
Flows in network can be categorized as congested flow, causing congestion, and uncongested flow, unrelated to it. Taking appropriate action for each type is critical to preventing HLB, making fine-grained pause schemes essential.

Coarse-grained Pause Scheme: Equally treatment of Congested and Uncongested Flows. In a coarse-grained pause scheme, two operational states exist: *all flows paused* and *all paused flows resumed*. Consequently, all packets are subjected to the same action, irrespective of whether the flow is responsible for the congestion. This leads to the emergence of the HLB issue, akin to standard PFC.

Fine-grained Pause Scheme: Pausing Congested Flows and Transmitting Uncongested Flows. In comparison to the coarse-grained pause scheme, the fine-grained approach involves taking specific actions based on different flow types. It distinguishes by pausing congested flows and transmitting normal flows, thereby alleviating the HLB issue.

The fine-grained pause scheme encompasses four states: *all flows paused*, *partial flows paused*, *partial paused flows resumed*, and *all paused flows resumed*. Both the *all flows paused* and *all paused flows resumed* states apply the same action, either pausing or resuming, to all packets. However, the other two states involve multiple types of packets, including incoming, normal, paused, and resumed packets. Further details are provided in § III-B. These states require careful handling to prevent Head-of-Line Blocking and out-of-order delivery issues, necessitating the isolation of different packet types.

A more straightforward method to achieve this isolation would be to assign a dedicated queue for each flow. However, the constrained number of FIFO queues per port in commodity switches renders this impractical [16], [17]. Consequently, this limitation poses a challenge in achieving full isolation of different packet types when utilizing a shared buffer.



(a) FCT Slowdown. (b) Buffer Occupancy Reduction. (c) Deadlock Number Reduction.

Fig. 2. Enhanced Performance of original congestion-based fine-grained ideal PFC (OFC) compared to local congestion-based coarse-grained standard PFC.

C. Experimental Observation

1) Simulation setting: Fig. 1 shows an incast topology, which is used to simulate the congestion scenario and introduce the specific actions taken for different types of flows. For better understanding, this topology is taken as a fundamental component of typical data center network CLOS topologies (Fat-Tree [18], Spine-leaf [19]) and encompasses the common features of congestion occurrence and handle [18], [20]. There are three switches connecting ten servers, and all links are 40 Gbit/s with propagation delay as 20 ns. Within this setup, the traffic is generated following the common traffic distribution in the data center as 20% long-lived and 80% burst flow [8]. The two long-lived flows, F_0 and F_1, share the link capacity from S_a to S_b and S_b to S_c. These two long-lived flows traverse the same switches ($S_a, S_b,$ ingress port P_a of S_c) but dequeue from different egress ports to reach their respective destinations. Specifically, long-lived flow F_0 dequeues from egress port P_b, while F_1 dequeues from egress port P_d. Additionally, six burst flows, F_2 - F_7, with a duration of 0-30us, are intended for the same receiver as long-lived flow F_1, resulting in congestion at egress port P_d.

2) Specific actions for types of flows: The contention between the burst flows F_2 - F_7 and the long-lived flow F_1 at egress port P_d of switch S_c identifies the original congestion at port P_d. The flows F_1 - F_7 are recognized as the congested flows. Upon sending the PAUSE frame to upstream ports S_b from ingress port P_a of switch S_c, with time, the congestion may propagate to upstream port S_a. In the context of the local congestion-based coarse-grained standard PFC, both long-lived flows F_0 and F_1 will be paused. Conversely, in the original congestion-based fine-grained pause scheme, the congested flow F_1 will be paused, while the normal flow F_0 will continue to transmit without interruption.

3) Results: Fig. 2(a), Fig. 2(b) and Fig. 2(c) illustrate that the original-congestion-based fine-grained pause scheme can effectively lower the Flow Completion time (FCT), occupied buffer and deadlock occurrence, compared to the local congestion-based coarse-grained standard PFC (see details in § IV). This underscores the critical need for the ideal PFC.

D. Summarize about the ideal PFC

In summary, the ideal PFC's key function is to implement a fine-grained pause scheme to manage congestion and mitigate HLB and congestion spreading issues. The challenges include:

1) The need to differentiate between local congestion and original congestion to prevent congestion spreading.
2) Notifying upstream congestion ports about the information pertaining to original congestion.
3) Addressing the challenge of isolating different types of packets, given the limitation on the number of FIFO queues per port in commodity switches.

III. Design of OFC

To achieve the ideal PFC, we propose OFC (Original congestion-based priority Flow Control) and implement it from two aspects: (1) Identifying original congestion and disseminating to upstream ports; (2) providing a fine-grained pause scheme. Next, we will explain their solutions respectively.

Before that, we outline the tables required to record data at each port for each priority. Each flow is identified by a 5-tuple: source and destination IP addresses, source and destination ports, and IP protocol. Flow distinction is based on the source and destination port numbers. OFC uses the following tables to monitor packet counts and manage congestion:

- flow_in_port_tab : Stores flow information for each priority in the port. The key includes the queue priority pri and the source-destination pair $\text{flow}_{\text{info}} = src, dst$, while the value holds the packet number.
- congested_flow_tab : Records information about congested flows $\text{flow}_{\text{info}} = src, dst$ in each priority queue, acquired from previously received PAUSE frames.
- pause_frame_tab : Stores PAUSE frames transmitted to the upstream port of each priority, using the upstream port p_{up} and the priority pri as the key, and the information of the congested flow $\text{flow}_{\text{info}}$ as the value.

A. Original Congestion Identification and Notification

Identification of Original Congestion. Following the analysis of local congestion (leaves of a congestion tree) and original congestion (root of congestion tree) in § II-A, a key distinction between them pertains to the congestion order. As a result, a practical approach to identifying the original congestion involves determining whether it has previously received a pause frame from a downstream port. Specifically, when a port initiates a pause condition, if it has not previously received a PAUSE frame, it is identified as the original congestion port, signifying the first port within this congestion tree to trigger the pause condition. Conversely, if the port has stored

Algorithm 1: Send PAUSE/RESUME frame

```
// downstream port
while a packet enqueues ingress queue do
    obtain the packet's upstream port p_up, egress queue
     length q_out, priority pri.
    if q_in ≥ X^c_OFF then
        if X^c_OFF ≤ q_in < X_OFF && q_out ≥ X^c_OFF then
            if congested_flow_tab == null then
                // original congestion port
                flow_info = flow_in_port_tab[pri].keys() ;
            else // local congestion port
                flow_info = congested_flow_tab[pri] ;
        end
        else if q_in ≥ X_OFF then
            flow_info = flow_in_port_tab[pri].keys().
        end
        Put flow_info into PAUSE frame.
        Send PAUSE frame to upstream port p_up.
         pause_frame_tab[p_up][pri] = flow_info.
    end
    else if q_in ≤ X_ON then
        Put flow_info =pause_frame_tab[p_up][pri] into
         RESUME frame.
        Remove pause_frame_tab[p_up][pri].
        Send RESUME frame to upstream port p_up.
    end
end
```

Algorithm 2: Receive PAUSE/RESUME frame

```
// upstream port
while a packet pkt enqueues egress queue do
    extract the packet's priority pri, source src, and
     destination dst.
    if pkt is a PAUSE frame then
        store flow_info into congested_flow_tab[pri].
        drop pkt.
    end
    else if pkt is a RESUME frame then
        remove flow_info from congested_flow_tab[pri].
        enqueue the copied pkt into the paused queue.
    end
    else
        flow_in_port_tab[pri][src, dst] += 1.
    end
end
```

information from a previous PAUSE frame, it indicates that its congestion is a result of its downstream congested port, thus classifying it as a local congestion port.

Efforts are focused on obtaining information about the original congestion and congested flow in Algorithm 1. For an ingress port p_{in}, it transmits packets to a set of egress ports $Q_{out} = \{p^1_{out}, p^2_{out}, \cdots, p^I_{out}\}$, where the length of each egress queue $p^i_{out} \in Q_{out}$ is denoted as q^i_{out}. Consequently, the ingress queue length is calculated as $q_{in} = \sum_1^I q^i_{out}$. The OFC is activated to halt the congested flow when the length of the in-coming queue surpasses the congested pause threshold X^c_{OFF} and is smaller than X_{OFF}, where $X^c_{OFF} \leq q_{in} \leq X_{OFF}$.The criterion $q_{out} \geq X^c_{OFF}$ is used to identify potential congestion at any of the egress ports. For a congested port without stored information in the congested_flow_tab, it is identified as the original congestion port, with the flows within it causing the original congestion. Conversely, if it has stored information in the congested_flow_tab, it is termed the local congestion port, and the stored congested flows from previously received PAUSE frames are considered as the congested flows that should be paused.

To avert buffer overflow during urgent congestion, OFC pauses all flows when the ingress queue length surpasses pause threshold X_{OFF}. It subsequently resumes the flows paused by this port which is stored by the pause_frame_tab, once the ingress queue length falls below resume threshold X_{ON}.

Notification and Handling of Original Congestion. When the pause condition is triggered, a PAUSE frame, carrying information about the paused flows, is transmitted to the upstream port. Following Algorithm 2, upon reception of the PAUSE frame, the upstream port is mandated to store the paused flow information in the congested_flow_tab and proceed to pause the congested flows. A local congestion port may be impacted by multiple congestion trees simultaneously. If two congestion trees affect a local congestion port, the local congestion port will include the congested flow information of both these congestion trees in the congested_flow_tab.

Upon receiving a RESUME frame, the upstream port removes resumed flows from the congested_flow_tab. The RESUME frame is then duplicated and placed in the paused queue to resume these flows. Further details will be provided in the next section. Additionally, when a data packet enqueues the egress queue, its information, including source and destination, are stored in the flow_in_port_tab.

B. Fine-grained Pause Scheme

Upon notification of congestion, each upstream port identifies and categorizes flows as either congested or uncongested. To implement a fine-grained pause mechanism, based on the analysis in § II-B, the challenges are as follows:

i) Isolation of Different Packet Types: It is crucial to isolate different types of packets under the *partial flows paused* and *partial paused flows resumed* states to prevent issues such as HLB or out-of-order delivery.

ii) Constraint of Limited FIFO Queues: Due to the limited availability of FIFO queues, effectively segregating each type of packet into distinct queues presents a challenge.

We firstly outline our analysis and then offer our solutions.

The minimum number of required queues. When a switch receives a control frame, it classifies the packets within itself as normal packets, paused packets, and resumed packets. And all newly arrived packets are categorized as in-coming packets. Therefore, the switch must adhere to two constraints with regards to isolation: firstly, ensuring isolation among different types of packets within the switch, and secondly, guaranteeing

isolation between in-coming packets and existing packets of the same type. In this context, packets from normal flows, paused flows, and resumed flows are categorized as normal packets, paused packets, and resumed packets, respectively. Given that normal packets can be promptly transmitted, our primary focus is on the other three types of packets.

In the four states of the fine-grained pause scheme, the *all flow paused* and *all paused flow resumed* states do not necessitate isolation. However, the remaining states involving multiple types of packets require isolation. Under the *partial flows paused* state, two types of packets are involved: in-coming packets and paused packets. Meanwhile, the *partial paused flows resumed* state involves three types of packets: in-coming packets, paused packets, and resumed packets. As the mixing of resumed packets with paused packets can result in HLB issue, and the mixing of in-coming paused with existing packets may lead to out-of-order delivery, each pair of packet types should not share a queue. Within these two states, there are at most three types of packets. Therefore, the minimum number of queues needed to achieve isolation is 3.

The function of two backup queues. Given the limited number of queues, we use use a minimum of three queues for isolation: the original priority queue for incoming packets and two backup queues for isolation. We'll show how the backup queues achieve isolation during the *partial flows paused* and *partial paused flows resumed* states. The key is consistently directing paused packets to the empty backup queue.

In the *partial flows paused* state, achieving isolation between in-coming packets and paused packets involves storing the paused packets in an empty backup queue, as the in-coming packets are already stored in the normal queue. When a packet is dequeued from the normal queue, if it is a normal packet, it can be promptly transmitted to the next hop. However, if it is a paused packet, it should be enqueued in the backup queue. Thus, isolation for the *partial flows paused* state can be effectively achieved with just one backup queue.

Emphasizing the need for two backup queues is crucial, as a single backup queue fails to achieve isolation under the *partial paused flows resumed* state. Paused and resumed packets share the same queue after receiving a RESUME frame, requiring two separate backup queues for effective isolation.

In the *partial paused flows resumed* state, three types of packets are involved: in-coming packets, paused packets, and resumed packets. Given that in-coming packets are already isolated by the normal queue, the primary challenge is to isolate the paused and resumed packets, as they are mixed in a backup queue called the paused queue. The key approach involves directing the paused packets to the additional empty backup queue. Specifically, when a packet is dequeued from the paused queue, if it is a resumed packet, it can be promptly transmitted to the next hop. However, if it is a paused packet, it should be enqueued in another empty backup queue. This is where the additional backup queue plays a crucial role. Consequently, by utilizing two backup queues, effective isolation of the *partial paused flows resumed* state can be achieved.

Nested Hierarchy Scheme. After thorough analysis, we devised a Nested Hierarchical scheme, as outlined in Algorithm 3, to enable a fine-grained pause scheme. This scheme incorporates a normal queue and two backup queues. The normal queue serves as the original priority queue for incoming packets. One backup queue is designated for paused flows (paused queue), while the other remains empty, ready to handle remaining paused packets in a *partial paused flows resumed* state. These queues are initialized as egress queue ($q_{Id} = 0$), paused queue ($q_{Id} = 1$) and another backup queue ($q_{Id} = 2$).

Algorithm 3: Pause and Resume

```
// upstream port
while a packet pkt dequeues egress queue do
    get the packet's downstream port p_down, priority
     pri, source src, and destination dst.
    // recall from Algorithm 2 line 9
    if pkt is a RESUME frame then
        egress queue.q_Id = Rotate(egress queue.q_Id).
        paused queue.q_Id = Rotate(paused queue.q_Id).
    end
    else if pkt is not in congested_flow_tab[pri] then
        send pkt to p_down.
        flow_in_port_tab[pri][src, dst] -= 1.
        if flow_in_port_tab[pri][src, dst] == 0 then
        remove flow_in_port_tab[pri][src, dst] ;
    end
    else
        pkt enqueue paused queue.
    end
end
function Rotate(q_Id):
    q_Id = (q_Id + 1) % 2.
    return q_Id
end function
```

The Nested Hierarchical scheme must ensure packet isolation to prevent HLB and guarantee that incoming packets are transmitted after existing ones to avoid out-of-order delivery. This is achieved in all four transitional states. Initially, all flows are considered normal, and the egress queue functions as the normal queue. When a PAUSE frame is received, the system transitions to the *partial flows paused* state, where the egress queue continues to transmit normal packets while directing paused packets to the paused queue, effectively isolating paused and incoming packets. If all flows are paused, the system enters the *all flows paused* state, where all flows are stored in the paused queue and cannot be transmitted.

When some paused flows are resumed after receiving a RESUME frame, the system moves to the *partial paused flows resumed* state. In this state, to maintain isolation between paused and resumed packets, the egress queue is rotated to the paused queue(egress queue.$q_{Id} : 0 \rightarrow 1$), allowing resumed packets to be transmitted normally, while the paused queue is rotated to an empty backup queue (paused queue.$q_{Id} : 1 \rightarrow 2$) to store the remaining paused packets. Once the RESUME frame is dequeued, the queues are rotated back to their original states (egress queue.$q_{Id} : 1 \rightarrow 0$, paused queue.$q_{Id} : 2 \rightarrow 1$), enabling the transmission of incoming packets in the normal queue and ensuring in-order delivery. The rotation process is

outlined in Algorithm 3 from line 17 to 20. And the RESUME frame serves as both the rotation signal and the in-order delivery signal, often referred to as the Order Mark. Once all paused flows are resumed, the system transitions to the *all paused flows resumed* state, where all flows continue normal transmission. Thus, the Nested Hierarchical scheme effectively implements the desired functions.

C. Threshold Setting

In the OFC framework, there are three thresholds: the pause threshold X_{OFF}, the resume threshold X_{ON}, and a newly introduced congested pause threshold X^c_{OFF}. Typically, the congested pause threshold is expected to be lower than the pause threshold and higher than the resume threshold, meaning that $X_{ON} < X^c_{OFF} < X_{OFF}$. Therefore, we establish $X^c_{OFF} = \alpha X_{OFF}$, where $\frac{X_{ON}}{X_{OFF}} < \alpha < 1$. We also assess the parameter sensitivity as outlined in § IV-C2.

IV. EVALUATION

A. Settings

1) Parameters: Each link has 40Gbit/s bandwidth with 20 ns propagation delay. We set pause threshold as $X^l_{\text{OFF}} = 68$KB and $X_{\text{OFF}} = 75$KB, resume threshold as $X_{\text{ON}} = 45$KB. The packet size is limited by a MTU of 1500 bytes.

2) Topology and Traffic: Next, we will introduce the evaluated network topology and traffic patterns for testbed cross-validation and simulation, encompassing three key scenarios.

Testbed topology and traffic. To validate the proof-of-concept, we implement OFC on the Intel Tofino switch [21] using $P4_{16}$ and Intel SDE 9.6.0(§ IV-B1), featuring 32 ports with 40Gbit/s link capacity. The setup uses port connections to emulate three programmable switches within one, matching the topology in Fig. 1, which includes four servers and three switches. The traffic generator uses two Mellanox CX5 NICs, with traffic distribution as described in § II-C1.

Simulation topology and traffic. To comprehensively assess real-world network performance, we evaluate the following three distinct scenarios. All simulations are examined using the packet-level transmission simulator, NetBench [22].

[*Incast scenario*] To estimate incast traffic with ratios from $4:1$ to $10:1$, we use a dumbbell topology with 2 rack switches, each connected to 32 servers. Thousands of flows are generated, mixing long-lived and burst flows [8]. We focus on the performance of all flows, especially uncongested flow.

[*Realistic scenario*] We simulate the realistic traffic of data center under a three-layer Fat-Tree [18] network (k = 8), with total 80 switches which is composed of 32 Core switches, 32 Aggregation switches, 16 Edge switches, and 256 servers, with an over subscription ratio of $2:1$. The real-world data center traffic traces are identified as EDU1 and EDU2 [9]. The traces consist of thousands of flows, both short and large. About 60% of flows in EDU1 are under 10 KB, while 80% of flows in EDU2 are under 20 KB. We observe the performance of short flows ($<$ 10KB) and large flows ($\geq$ 10KB). Additionally, the OFC's parameter sensitivity is also evaluated.

[*Deadlock scenario*] To assess deadlock frequency, we use a leaf-spine topology with two spine switches, four leaf switches, and two failed links, a setup prone to Cyclic Buffer Dependency (CBD) [12]. We create four flows $\{f_1, f_2, f_3, f_4\}$ that form a CBD. With a total of $M = 240,400,640$, the specific number of each flow is (m_1, m_2, m_3, m_4) with constraints $m_1 + m_2 + m_3 + m_4 = M$ and $m_1, m_2, m_3, m_4 \geq 0$. we randomly distribute each flow's count over 1000 iterations

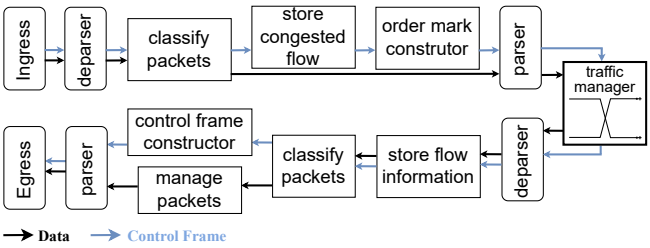


Fig. 3. The OFC pipeline in the programmable switch with ASIC structure.

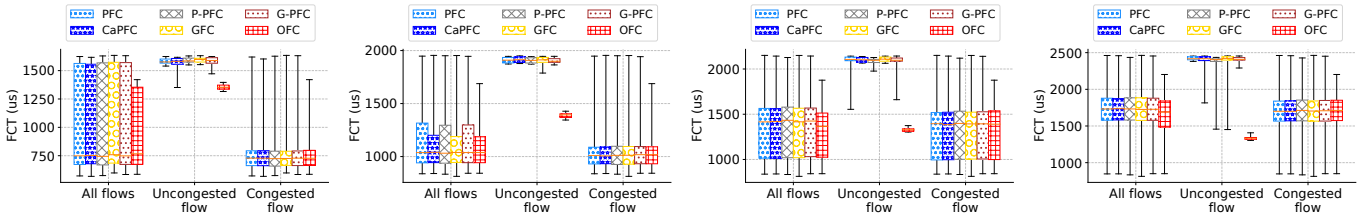
3) Performance Metric: • **Flow Completion Time (FCT):** The FCT measures the time needed to transmit a group of flows, with a shorter duration being more favorable. • **Average Queue Length:** Reducing the average queue length is important to minimize buffer occupancy and alleviate congestion. • **Deadlock Number:** Minimizing the occurrence of deadlocks is crucial, as they can cause significant disruptions.

4) Baseline: We compare OFC's performance with state-of-the-art solutions, excluding those with out-of-order delivery, which are unsuitable for order-sensitive networks like RDMA. (i) **PFC** [6]: Standard PFC uses pause and resume thresholds to prevent buffer overflow by maintaining buffer occupancy below a set level. (ii) **caPFC** [23]: Adds a fixed egress threshold to the standard PFC to identify congested ports and pause flows. (iii) **P-PFC** [24]: Utilizes a dynamic egress threshold (average buffer occupancy of all egress ports) to determine paused ports. (iv) **GFC** [25]: Gradually decreases transmission rates to zero by incorporating multiple lower ingress thresholds based on standard PFC. (v) **G-PFC** [26]: Selectively pauses low-priority flows using a lower ingress threshold, building on the standard PFC scheme.

B. Testbed Micro-benchmark

We first show the performance evaluation under testbed.

1) Implementation: Fig. 3 illustrates the pipeline of OFC in the programmable switch Intel Tofino1 [21]. Given that the fundamental ASICs' pipeline of the programmable switch consists of the ingress parser, several Match-Action Units, ingress deparser, traffic manager, and a similarly structured egress, OFC incorporates additional Match-Action Units for classifying packets, storing congested flow, constructing order mark, storing flow in port, constructing control frame and managing packets to achieve its primary functions. The classification of normal/resumed packets and paused packets is dependent on whether the source and destination pair of the packet corresponds to the congested_flow_table, while the distinction between data packets and control frames relies on the packet header. As the capability to pause or resume a specific queue is exclusively supported in the Intel Tofino2 [27], we approximate the fine-grained pause by comparing the queue



(a) FCT of incast ratio $4:1$. (b) FCT of incast ratio $6:1$. (c) FCT of incast ratio $8:1$. (d) FCT of incast ratio $10:1$.

Fig. 4. FCT under the dumbbell topology with incast ratios varying ranging from $4:1$ to $10:1$.

TABLE I
MAXIMUM FCT

Kind	All flows		Uncongested flow		Congested flow	
	PFC	OFC	PFC	OFC	PFC	OFC
Testbed	2.69 ms	2.36 ms	2.69 ms	2.28 ms	2.69 ms	2.36 ms
Simulation	1.96 ms	1.69 ms	1.96 ms	1.45 ms	1.96 ms	1.69 ms

TABLE II
AVERAGE STORAGE TABLE SIZE

Kind	flow_in_port_tab	congested_flow_tab	pause_frame_tab
Testbed	432 byte	152 byte	288 byte
Simulation	425.66 byte	142.46 byte	276.47 byte

length calculated by flow_in_port_tab to the pause threshold or resume threshold. Subsequently, the paused queue is managed by circulating the paused packets to halt their transmission.

2) Performance under programmable switch: We present the testbed results of the maximum Flow Completion Time (FCT) and the average storage table size to further validate the simulation results. TABLE I illustrates consistent FCT levels and comparable trends in PFC and OFC, in line with the simulation results for the same topology and traffic as demonstrated in Fig. 2(a). Furthermore, to support the storage of three tables in each switch port for determining paused flows, we utilize Cuckoo Hash [28] to compress the necessary buffer, achieving approximately 70% compression. The tables in the programmable switch are implemented using multiple registers, with each register capable of storing 64 bits. The specific average table size, occupying the buffer in both the testbed and simulation, is detailed in TABLE II, indicating that the storage tables do not extensively utilize the buffer space.

C. Simulations

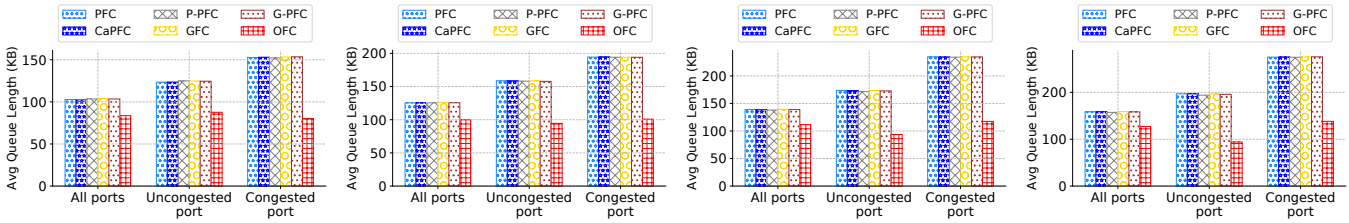
1) Performance under incast scenario: **FCT.** Fig. 4 illustrates the FCT of OFC in comparison to baselines under incast scenario with varying incast ratios from $4:1$ to $10:1$. The box plot shows the minimum value, 25^{th} percentile tail-latency, 75^{th} percentile tail-latency, and the maximum value. It is evident that the baselines exhibit uniform FCT levels for both uncongested and congested flows, indicating consistent behavior irrespective of flow type. Conversely, OFC demonstrates a notably lower FCT for uncongested flows, signifying its ability to facilitate the transmission of uncongested flows while pausing the transmission of congested flows. Moreover, the transmission of uncongested flows also results in a reduced FCT for congested flows, thereby lowering the FCT for all network flows. Furthermore, the reduction of FCT grows more larger with the increase of congestion degree. From incast ration $4:1$ to $10:1$, OFC lowers FCT of all network flows including uncongested flow and congested flow than baselines, by up to 19.28% to 22.6%, 23.08% to 43.06%, 20.59% to 54.55%, 15.59% to 60.28%, respectively.

Average Queue Length. The occupied buffer under the incast scenario with incast ratios ranging from $4:1$ to $10:1$ is depicted in Fig. 5. This figure shows the specific average queue lengths of ports through which all flows, uncongested flows, and congested flows traverse, labeled as all ports, uncongested port, and congested port, respectively. Observing the data, it is evident that the average queue length of congested ports is higher than uncongested ports, and since all flows include burst flows, the average queue length of all ports is lower than the uncongested port. Furthermore, as the incast ratio increases, there is a concurrent increase in the average queue length across all types of ports, indicating the correlation between the average queue length and the degree of congestion.

Moreover, analyzing the average queue length of each type of port across all incast ratios reveals that OFC effectively reduces the average queue length compared to the baseline, with a more substantial reduction observed as congestion potential intensifies. In detail, under incast ratios ranging from $4:1$ to $10:1$, OFC can decrease the average queue length of all ports including uncongested ports and congested ports compared to baselines by up to 21.74% to 49.33%, 21.81% to 50%, 22.22% to 51.47%, 20.64% to 49.74%, respectively.

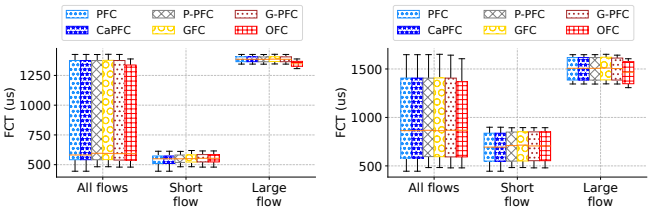
2) Performance under realistic scenario: **FCT.** Fig. 6 depicts the FCT of all flows, including short flows and large flows, under real-world traffic patterns EDU1 and EDU2. The relationship between flow size and FCT is observed, with the FCT of short flows being smaller than that of large flows, highlighting that the FCT of large flows significantly influences the overall FCT of all network flows. Furthermore, as EDU2 comprises a higher proportion of large flows compared to EDU1, its FCT is correspondingly larger. It is notable that under both traffic patterns, OFC reduces the FCT of large flows, consequently leading to a reduction in the FCT of all flows by around 5% compared to baselines.

Average Queue Length. The average queue length of all ports, core layer, and aggregation layer within the fat-tree network under real-world traffic patterns EDU1 and EDU2 is presented



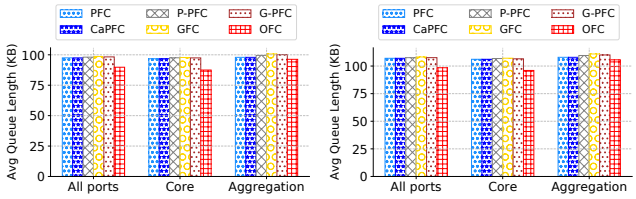
(a) Average queue length of incast ratio $4:1$. (b) Average queue length of incast ratio $6:1$. (c) Average queue length of incast ratio $8:1$. (d) Average queue length of incast ratio $10:1$.

Fig. 5. Average queue length under the dumbbell topology with incast ratios ranging from $4:1$ to $10:1$.



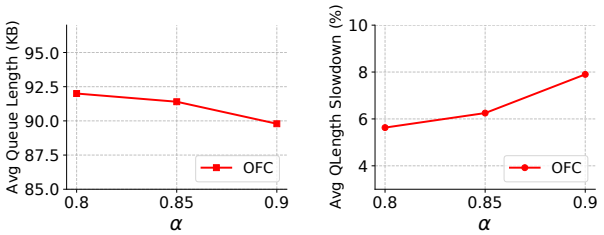
(a) FCT of real-world traffic EDU1. (b) FCT of real-world traffic EDU2.

Fig. 6. FCT under the fat-tree topology with real-world traffics.

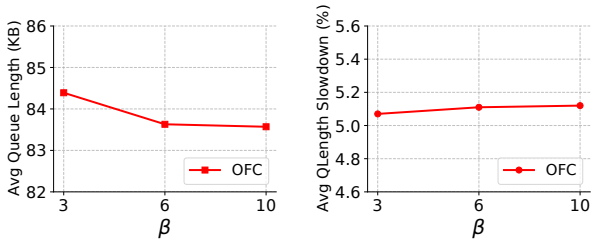


(a) Average queue length of real-world traffic EDU1. (b) Average queue length of real-world traffic EDU2.

Fig. 7. Average queue length in the fat-tree topology with real-world traffics.



(a) Avg queue length with varying X^c_{OFF}, fixed β. (b) Avg queue length Slowdown than PFC with varying X^c_{OFF}, fixed β.



(c) Avg queue length with varying X_{OFF}, fixed α. (d) Avg queue length Slowdown than PFC with varying X_{OFF}, fixed α.

Fig. 8. Average queue length and slowdown than standard PFC while varying parameters under the fat-tree topology with real-world traffic EDU1.

in Fig. 7. It is notable that due to the higher workload in EDU2 compared to EDU1, the average queue length of each type of port is elevated in EDU2. Moreover, OFC reduces the buffer occupancy of all types of ports under both real-world traffic scenarios. Specifically, the average queue length of OFC is decreased than baselines by up to 4.31% to 11.83%, 5.8% to 11.24% under EDU1 and EDU2, respectively.

Parameter Sensitivity. We accessed the sensitivity of OFC to parameter changes by varying the pause threshold settings under the fat-tree network with real-world traffic pattern EDU1. By fixing $X_{OFF} = 75$KB and varying α from $\{0.8, 0.85, 0.9\}$ to modify the lower pause threshold $X^c_{OFF} = \alpha X_{OFF}$, we can find that despite the changes in α, OFC exhibited resilience. Specifically, the average queue length does not exceed a 3% variation (Fig.8(a)), and its slowdown compared to PFC was approximately 5.8% to 8% (Fig.8(b)). Furthermore, we varied the pause threshold $X_{OFF} = \beta \cdot 75$KB by fixing $\alpha = 0.9$ and changing β from $\{3, 6, 10\}$. This analysis reveals that OFC maintains stability with varying pause thresholds X_{OFF}, as the change in the average queue length of all ports does not exceed 2.5% (Fig. 8(c)) and its slowdown than PFC remains at approximately 5.1% (Fig. 8(d)).

3) Performance under deadlock scenario: **Deadlock Number.** Fig. 9 illustrates the incidence of deadlocks in the deadlock scenario for both the baselines and OFC. Evidently, across 1000 repeated simulations with three different workloads, OFC consistently and significantly reduces the number of deadlocks by 7.15% - 45.24%, 4.14% - 40.43%, 4.99% - 41.53%, 8.64% - 48.70% and 7.95% - 47.33% compared to PFC, CaPFC, P-PFC, GFC, and G-PFC. The increased occurrence of deadlocks in baselines is mainly due to their coarse-grained pause policy, where deadlock occurs when every switch in a CBD pauses all transmissions simultaneously. In contrast, OFC reduces deadlock risk with a fine-grained pause policy, selectively pausing only congested flows.

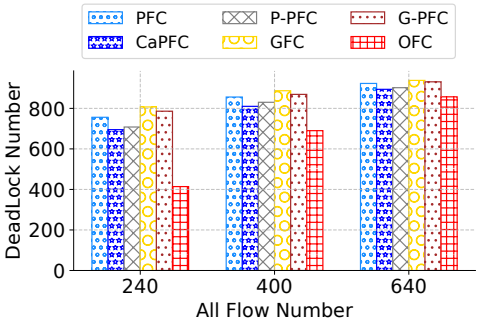


Fig. 9. Deadlock occurrence under deadlock scenario.

V. Related work

A. PFC

Coarse-grained pause scheme. CaPFC [23] and P-PFC [24] modify the pause trigger condition from ingress queue length to include both ingress and egress queue lengths, differing in their fixed or adaptive egress queue length thresholds. GFC [25] introduces new ingress queue length thresholds below X_{OFF} to gradually reduce the pause rate. G-PFC [26] also uses an ingress queue length threshold below X_{OFF} to trigger the pause of low-priority flows before PFC. However, all lack a fine-grained scheme, leading to the HLB issue.

Out-of-order delivery. PFC-S [29] judges the congested flow based on egress queue length, and reroutes the uncongested flow at upstream port by sending MOVE frame. FG-PFC [30] identifies the congested flow as the most count flow when trigger ingress queue length threshold, and pauses these flows without isolation of in-coming packets belonging to congested flow. Thus, both of them may arise out-of-order delivery.

B. Congestion Control rely on PFC to ensure lossless

End-to-end congestion control methods, such as QCN [31], DCQCN [14], TIMELY [32], SWIFT [33], HPCC [34], and Poseidon [35], focus on reducing the sending rate to alleviate congestion. While these techniques use various approaches to detect and manage congestion, they are generally indifferent to the victim flow. In contrast, other approaches [36]–[41], aim to achieve more accurate congestion control for congested flows. It is important to note that, regardless of their specific methodologies, all of these techniques must be integrated with PFC to prevent packet loss from buffer overflow.

VI. Conclusion

Regarding PFC issues, existing solutions are either coarse-grained or vulnerable to out-of-order delivery. Through testbed and simulation experiments, our proposed OFC effectively reduces FCT, buffer occupancy, and the occurrence of deadlocks.

VII. Acknowledgement

We sincerely thank the chairs and the anonymous reviewers for their constructive feedback. This work was supported by the NSF China (No. 61960206002, No. 62272292 and No. 61902246). Shizhen Zhao is the corresponding author.

References

[1] Y. Gao, Q. Li *et al.*, "When cloud storage meets rdma," in *NSDI*, 2021.
[2] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *SIGCOMM*, 2016.
[3] W. Bai *et al.*, "Empowering azure storage with rdma," in *NSDI*, 2023.
[4] P. Cao *et al.*, "Network load balancing with parallel flowlets for AI training clusters," in *NAIC*, 2024.
[5] C. H. Song, X. Z. Khooi, R. Joshi *et al.*, "Network load balancing with in-network reordering support for rdma," in *SIGCOMM*, 2023.
[6] IEEE. (2011) Priority-based flow control. [Online]. Available: https://1.ieee802.org/dcb/802-1qbb
[7] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *SIGCOMM*, 2010.
[8] M. Noormohammadpour *et al.*, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Commun Surv Tutor*, 2018.
[9] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *IMC*, 2010.
[10] J. Xue, M. U. Chaudhry *et al.*, "Dart: Divide and specialize for fast response to congestion in rdma-based datacenter networks," *TON*, 2020.
[11] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in *IEEE Globecom*, 2006.
[12] S. Hu, Y. Zhu, P. Cheng *et al.*, "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *HotNets*, 2016.
[13] G. Kim, C. Kim, J. Jeong, M. Parker, and J. Kim, "Contention-based congestion management in large-scale networks," in *MICRO*, 2016.
[14] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn *et al.*, "Congestion control for large-scale rdma deployments," *SIGCOMM*, 2015.
[15] M. Tang, X. Lin, and M. Palesi, "Local congestion avoidance in network-on-chip," *TPDS*, 2016.
[16] HPE. (2017) Hpe aruba 8400. [Online]. Available: https://www.hpe.com/us/en/networking/switches.html
[17] S. Hu, Y. Zhu, P. Cheng *et al.*, "Tagger: Practical pfc deadlock prevention in data center networks," *TON*, 2019.
[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.
[19] M. Alizadeh, T. Edsall, S. Dharmapurikar *et al.*, "Conga: distributed congestion-aware load balancing for datacenters," in *SIGCOMM*, 2014.
[20] C. Clos, "A study of non-blocking switching networks," *BSTJ*, 1953.
[21] Intel. (2018) Tofino. [Online]. Available: https://goo.gl/cdEK1E
[22] S. Kassing, A. Valadarsky, and A. Singla. (2016) Netbench. [Online]. Available: https://github.com/ndal-eth/netbench.
[23] S. N. Avci, Z. Li, and F. Liu, "Congestion aware priority flow control in data center networks," in *IFIP Networking*, 2016.
[24] C. Tian, B. Li *et al.*, "P-pfc: Reducing tail latency with predictive pfc in lossless data center networks," *TPDS*, 2020.
[25] K. Qian, W. Cheng, T. Zhang, and F. Ren, "Gentle flow control: avoiding deadlock in lossless networks," in *SIGCOMM*, 2019.
[26] Z. Cui and S. Y. Rim, "G-pfc: A packet-priority aware pfc scheme for the datacenter," in *APNOMS*, 2020.
[27] Intel. (2023) Tofino2. [Online]. Available: https://ieeexplore.ieee.org/document/9220636
[28] D. Zhou, B. Fan, H. Lim *et al.*, "Scalable, high performance ethernet forwarding with cuckooswitch," in *CoNEXT*, 2013.
[29] W. Gaoξ, J. Huang, Q. Wang, S. Zhou, and Z. Li, "Pfc-s: Reducing tail latency with pfc sensitive in lossless data center networks," *SSRN*, 2023.
[30] S. Li, C. Wang, Y. Zhang, C. Ma, L. Li, X. Cui, and J. Liu, "Fg-pfc: A fine-grained pfc mechanism for lossless rdma," *JPCS*, 2023.
[31] IEEE. (2010) Congestion notification. [Online]. Available: 802.11Qau.
[32] R. Mittal, L. Vinh The, N. Dukkipati *et al.*, "Timely: Rtt-based congestion control for the datacenter," in *SIGCOMM*, 2015.
[33] G. Kumar, N. Dukkipati *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *SIGCOMM*, 2020.
[34] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, and L. o. Tang, "Hpcc: High precision congestion control," in *SIGCOMM*, 2019.
[35] W. Wang, M. Moshref, Y. Li *et al.*, "Poseidon: Efficient, robust, and practical datacenter cc via deployable int," in *NSDI*, 2023.
[36] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *NSDI*, 2020.
[37] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *SIGCOMM*, 2021.
[38] P. Dong, X. Lu, T. Huang *et al.*, "Predictive queue-based rate control for low latency in lossless data center networks," *TNSM*, 2024.
[39] P. Goyal *et al.*, "Backpressure flow control," in *NSDI*, 2022.
[40] IEEE. (2018) Qcz. [Online]. Available: https://1.ieee802.org/tsn/
[41] W. Li, C. Zeng, J. Hu, and K. Chen, "Towards fine-grained and practical flow control for datacenter networks," in *ICNP*, 2023.