# FLOWSAIL: Fine-Grained and Practical Flow Control for Datacenter Networks

Wenxue Li, *Student Member, IEEE*, Chaoliang Zeng, Jinbin Hu, *Member, IEEE*, and Kai Chen, *Senior Member, IEEE*

***Abstract*— As datacenter networks continue to support a wider range of applications and faster link speeds, they face the challenge of managing bursty traffic and transient congestion. End-to-end congestion controls (CCs) find it increasingly difficult to maintain effectiveness due to the inherent feedback delay. To address this issue, per-hop flow control (FC) has gained popularity due to its ability to react promptly to transient congestion. However, existing FC mechanisms either lack fine-grained (i.e., per-flow granularity) control or require an impractical number of queues that exceeds the capabilities of commodity switches. In this paper, we introduce FLOWSAIL, an innovative FC scheme that enables fine-grained control at the per-flow level while requiring a practical number of switch queues, theoretically as few as two. The core of FLOWSAIL is an effective approximation of ideal FC by three key design components: dynamic flow-to-queue mapping, hierarchical congested flow identification, and on-demand isolation. We have implemented a prototype of FLOWSAIL using the programmable P4 switch and conducted extensive testbed experiments and simulations. The results indicate that FLOWSAIL effectively sustains performance with significantly fewer queues compared to existing FC schemes. For instance, FLOWSAIL achieves 4.3× lower tail latency under the same number of queues, matches existing FC schemes with 4× fewer queues, and holds robust performance with a minimum of 2 queues.**

***Index Terms*— Datacenter networks, flow control.**

## I. INTRODUCTION

NOWADAYS, datacenters support an ever-growing variety of applications, including distributed computing, machine learning systems, and large-scale data analysis [1], [2], [3], [4], [5], [6], [7], [8]. These applications often exhibit partition/aggregation traffic patterns, leading to more frequent congestion events. The underlying network fabric must simultaneously support the application's specific requirements, such as high throughput, low latency, and quality of service (QoS). Congestion control (CC) is pivotal in facilitating effective communication. Current datacenter networks (DCNs) primarily depend on complex end-to-end CC mechanisms that utilize the receiver-echoed congestion signals to regulate the sending rate. Examples of such mechanisms include DCTCP [9], DCQCN [10], Swift [11], and HPCC [12].

Nonetheless, end-to-end CC faces challenges in managing bursts, as senders need at least one network round-trip time (RTT) to receive the receiver-echoed signals before adjusting sending rates, leading to a loss of control over bursty flows. The increasing DCN link speed and shadow buffer size exacerbate this issue (§II-A). Firstly, the proportion of flows that can complete within a single RTT increases with link speed [13], [14]. These flows not only generate tricky transient congestion but also cause non-negligible sub-RTT fluctuations, disrupting the control of larger flows [15], [16]. Secondly, the buffer size of commodity switches struggles to keep up with the rising link speed, making it more challenging for switches to manage the transient congestion and wait for the response from end-to-end CC [13], [17].

Per-hop flow control (FC) has gained popularity due to its timely and effective response in addressing transient bursts. Typically, an FC scheme uses queue length as a congestion signal and pauses (or resumes) the upstream entity to mitigate congestion (or increase utilization) within a 1-Hop RTT (usually 1∼2$\mu$s), which is significantly faster than the end-to-end RTT [18], [19]. However, existing FC mechanisms often fail to maintain effectiveness in practice; they either lack fine-grained (*i.e.*, per-flow granularity) control or demand an impractical number of queues beyond the capacities of commodity switches (§II-B).

On one hand, Priority Flow Control (PFC) [20] is a coarse-grained mechanism, operating on a per port (or priority queue) basis. Consequently, PFC introduces a multitude of well-known issues, such as Head-of-Line (HoL) blocking, congestion spreading, and deadlock [10], [12], [21], [22], [23], [24]. On the other hand, the ideal FC [25] allocates each flow to an exclusive queue in the switch, which is fine-grained but impractical. BFC [13], considered the state-of-the-art FC solution, still demands a large number of physical queues exceeding the typical switch capabilities, and BFC compromises isolation granularity and suffers performance degradation when queues are insufficient.

Considering the importance of FC and the inefficiency of existing FC schemes, we pose the question: *Is it possible to design an FC scheme that offers fine-grained control (i.e., per-flow granularity) and requires a feasible number of queues?*

Manuscript received 25 September 2023; revised 15 April 2024; accepted 6 May 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Wu. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101400001, in part by Hong Kong RGC TRS under Grant T41-603/20R, in part by GRF under Grant 16213621, in part by the ITF Access, in part by NSFC under Grant 62062005, in part by TACC, in part by the National Natural Science Foundation of China under Grant 62102046, in part by the Natural Science Foundation of Hunan Province under Grant 2022JJ30618, and in part by the Scientific Research Fund of Hunan Provincial Education Department under Grant 22B0300. *(Corresponding author: Kai Chen.)*

The authors are with the iSING Laboratory, The Hong Kong University of Science and Technology, Hong Kong, SAR, China (e-mail: wlicv@connect.ust.hk; czengaf@connect.ust.hk; jinbinhu@ust.hk; kaichen@cse.ust.hk).

Digital Object Identifier 10.1109/TNET.2024.3406613

In this paper, we offer a cautiously optimistic response through FLOWSAIL[1]. The key idea of FLOWSAIL is to emulate the behaviors of ideal FC with a practical number of switch queues, theoretically as few as two.

We find that the effectiveness of ideal FC stems from two aspects: *(i)* the precise determination of which set of flows are responsible for congestion on the congested port; *(ii)* the independent pausing or resuming the transmission of flows upon receiving control frames from downstream entities. We observe that we can emulate the two aspects' behavior of ideal FC without the requirement for per-flow queues (§III-A).

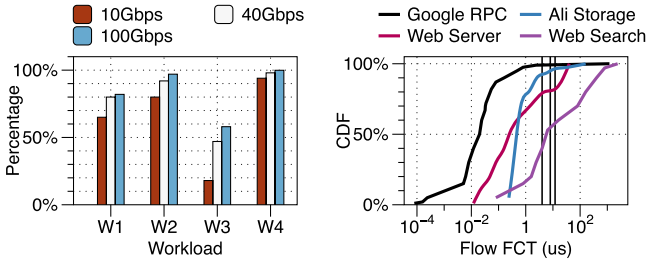
- At the congested port, if we can monitor each flow's buffer size, we can make the same per-flow level control decisions as the ideal FC, eliminating the need for a per-flow queue.
- At the upstream port, if we can direct all flows that need to be paused to a single paused queue while keeping other queues active, we can approximate the control action of the ideal FC, theoretically needing only two queues.

At its core, FLOWSAIL addresses several challenges to ensure its effective implementation (§III-B). *(i) How to define the granularity of congested flows?* A flow can experience intermittent congestion during its lifespan, leading to a non-constant congestion status. The definition of congested flow must account for this variability. *(ii) How to effectively and correctly identify congested flows at the congested port?* The decision of which set of flows should be paused or resumed must be made with an acceptable computational complexity while identifying the real congestion-responsible flows. (3) *How to isolate flows and avoid the out-of-order issue?* FLOWSAIL tries to allocate all the flows requiring pausing to a single queue, which could potentially lead to an out-of-order problem if flows are already buffered in other queues.

FLOWSAIL carefully designs its components and systematically integrates them (§IV) to address the aforementioned challenges. First, similar to BFC [13], FLOWSAIL identifies congested flows at the granularity of *active flows*, allowing for adaptation to the varying congestion conditions during a flow's lifetime. Second, FLOWSAIL dynamically maps flows to available queues and adopts a *hierarchical congested flow identification* method, utilizing two-dimensional states to decide which set of flows should be controlled. Third, FLOWSAIL performs an *on-demand isolation* mechanism after receiving control frames, where FLOWSAIL reassigns all the flows requiring pausing to a single isolation queue and ensures definite in-order delivery when scheduling flows.

We have implemented a FLOWSAIL prototype using the commodity programmable P4 switch [27] and demonstrated that FLOWSAIL can operate at a 100Gbps line rate (§V). Through extensive testbed experiments and simulations, we illustrate that FLOWSAIL significantly outperforms existing schemes (§VI). The testbed experiments reveal that FLOWSAIL mitigates congestion at a per-flow granularity. Concurrently, large-scale simulations demonstrate that FLOWSAIL achieves superior performance under various workloads and settings. For example, compared with existing FC schemes, FLOWSAIL achieves 4.3× lower tail ($99^{th}$ percentile) latency for short flows and 2× higher average throughput for large flows under the same number of queues and maintains comparable performance with 4× fewer queues. FLOWSAIL exhibits resilience in the face of incast degrees and parameter settings and holds robust performance with a minimum of 2 queues.

[1]An earlier version has been published in IEEE ICNP 2023 [26].




(a) With increasing link speeds. (b) CDF of theoretical FIT.

Fig. 1. The left graph displays the percentage of flows that complete injecting within a 12$\mu$s RTT across four workloads and three link speeds. The right graph presents the CDF of flow injection time (FIT) on a 100Gbps link; three vertical lines represent 4$\mu$s, 8$\mu$s, and 12$\mu$s RTTs, respectively.

In summary, we make the following key contributions:

- We observe that the behavior of the ideal FC can be effectively approximated with a practical number of queues, which, in theory, could be as few as two.
- We introduce FLOWSAIL, an FC scheme that supports fine-grained control while only requiring a feasible number of queues. FLOWSAIL employs several key design components to tackle the non-trivial design challenges encountered.
- We implement a FLOWSAIL prototype using the commodity programmable P4 switch and demonstrate its ability to operate at line rate. The superior performance of FLOWSAIL is further substantiated through extensive testbed experiments and simulations.

## II. BACKGROUND AND MOTIVATION

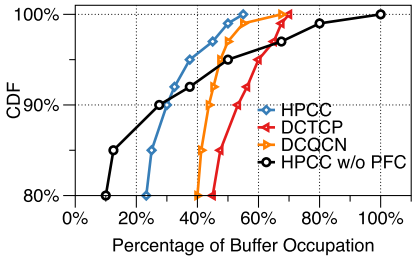
### A. End-to-End Congestion Control Falls Short

The escalating DCN link speed and shadow buffer size render it increasingly difficult to achieve satisfactory performance solely with end-to-end CC protocols.

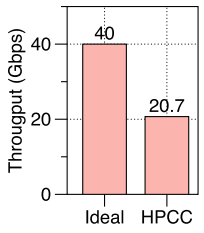
**Rising link speeds result in increasingly short flows** . As datacenter link speeds swiftly expand from tens to hundreds of Gigabytes per second, a larger proportion of flows become "smaller," and can be transmitted within a single RTT. To highlight this, we analyze four production datacenter workloads: Web Server [28] (W1), Alibaba Storage [12] (W2), Web Search [9] (W3), and Google RPC [29] (W4).

First, we measure three link speeds and calculate their bandwidth-delay products (BDPs), assuming a 12$\mu$s RTT,[2] and then classify flows with sizes smaller than the BDP as capable of finishing injecting within one RTT. Fig. 1a displays the percentage of such flows. The result indicates that as link speeds rise, more flows can complete their injection within one RTT. We then present the cumulative distribution function (CDF) of flow injection time (FIT) under W1∼W4 and a

[2]The 12$\mu$s is a typical 6-hops end-to-end latency in a 3-layer fat-tree topology.




(a) Overall buffer occupation.




(b) Throughput of a long-lived flow.

Fig. 2. (a) End-to-end CC alone results in large switch buffer occupation. (b) PFC causes throughput degradation.

100Gbps link speed in Fig. 1b. The rack-level (4$\mu$s), pod-level (8$\mu$s), and 3-layer (12$\mu$s) RTTs are explicitly indicated as three vertical lines. As the results demonstrate, even for rack-level communication with a considerably small RTT, a significant proportion of flows can complete within the first RTT. Numerous studies have also confirmed this trend [13], [14], [15], [16], [30].

The increasing short flows could potentially induce greater burstiness in network traffic. This not only injects uncontrolled congestion into the network but also affects large flows. Large flows may encounter rapidly emerging and disappearing cross-traffic bursts, resulting in non-negligible sub-RTT network fluctuations and disrupting the proper control [16], [30].

**End-to-end CC alone is insufficient for managing transient congestion.** End-to-end CC protocols primarily depend on receiver-echoed signals (*e.g.*, ECN [9], RTT [31], and INT [12]) to modulate sending rates. Consequently, these protocols struggle to handle transient congestion. We first demonstrate this issue using an experiment where we evaluate DCQCN, DCTCP and HPCC with PFC, and HPCC without PFC in a 3-layer fat-tree topology, using a Web Server workload with a 55% average load and 5% incast traffic. The switch employs a shared buffer scheme with a 12MB total buffer size. The CDF of the overall buffer occupation on the switch is presented in Fig. 2a. The results indicate that end-to-end CC alone cannot maintain low switch buffer occupancy. For example, HPCC alone ("HPCC w/o PFC") experiences high tail buffer occupation and a potential buffer overflow. Integrating PFC reduces the buffer occupation, highlighting the necessity of per-hop FC. Despite this, PFC's efficacy is restricted since it is ingress-queue-based and coarse-grained.

To further illustrate the impact of PFC on flow-level metrics, we conducted a micro-benchmark using NS-3 [32]. The topology consists of a ToR switch connecting several servers. We assign one server as the destination and allow another server to deliver a long-lived flow to it. Meanwhile, the remaining servers transmit cross-traffic at a 60 Gbps total load, following the Web Server distribution, to it. The long-lived flow and cross-traffic share a single 100Gbps link. We evaluate HPCC with PFC enabled and measure the average throughput of this long-lived flow. The results in Fig. 2b demonstrate that the long-lived flow experiences nearly a 50% reduction in throughput compared to an ideal situation. This is because cross-traffic causes significant transient congestion, eluding HPCC's control and leading to continuous PFC pausing.

This issue is further complicated by the fact that the commodity switch buffer size lags behind bandwidth, thus posing a challenge for switches in managing transient congestion while awaiting the response from end-to-end CC [14], [17], [33].

### *B. Existing Flow Control Schemes Are Insufficient*

Per-hop FC can more promptly address transient congestion. Specifically, per-hop FC can directly regulate the upstream entity's transmission to mitigate its congestion or increase utilization within a 1-Hop RTT (usually 1-2$\mu$s), as opposed to an end-to-end RTT. However, existing FC mechanisms either cannot manage congestion on a fine-grained basis or require an impractical amount of physical switch queues.

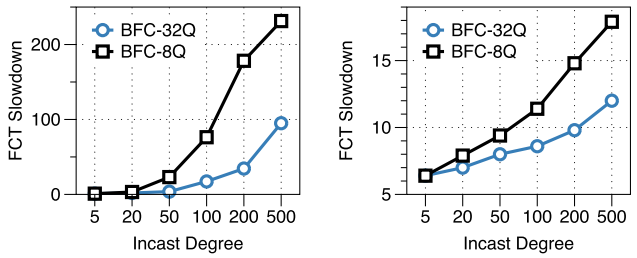
**PFC is coarse-grained.** PFC [20] pauses the upstream entity at a per-priority-queue granularity when the ingress queue length exceeds a specified threshold. Since PFC does not differentiate between flows, innocent flows may be paused when they share queues with congested flows. As a result, PFC leads to a mass of well-known problems [10], [12], [21], [22], [23]. Industries attempt to prevent PFC triggering by using complex end-to-end protocols. However, as we have previously discussed, these end-to-end protocols have inherent limitations and per-hop FC is necessary for handling transient congestion.

**Ideal FC is fine-grained but impractical.** The ideal implementation of per-hop FC [25] allocates a dedicated queue to every flow, thus providing promising per-flow level control. Despite its potential, current switch capabilities cannot accommodate the number of physical queues demanded by the ideal FC, rendering it impractical.

**BFC compromises isolation granularity when queues are limited.** BFC [13] is currently considered the state-of-the-art FC solution that can be implemented in today's programmable switches. BFC relies on physical queues for regulating flows. The original implementation of BFC uses 32-128 queues per port; however, this number exceeds the typical capacity of commodity switches for two main reasons. Firstly, the majority of switches cannot support such a large number of queues per port and are usually equipped with 8 or fewer queues [27], [34], [35]. Secondly, even if we assume access to high-capacity switches, the physical queues are critical resources and are typically reserved for *strong physical isolation and differentiation* between applications of different tenants. Consequently, dedicating all the queues to intra-tenant traffic is impractical [36], [37]. Note that FLOWSAIL primarily focuses on reducing the number of required physical queues. However, in realistic implementation, FLOWSAIL still requires several advanced switch features (details in §V).

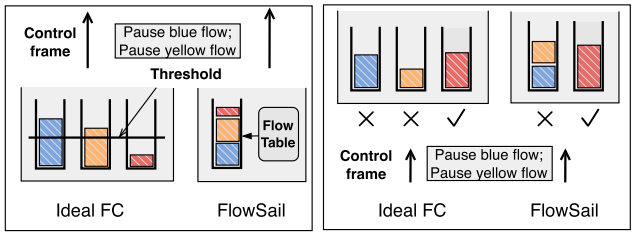
BFC dynamically assigns a dedicated queue to each active flow. However, when there are no available queues, BFC permits multiple flows to share a queue and *manages all flows within the same queue collectively*, thereby diminishing isolation granularity. Consequently, its performance is critically dependent on the number of available queues and experiences considerable degradation when queues are limited.

We conduct a simulation to demonstrate BFC's performance with 32 queues (BFC-32Q, used in the original paper) and




(a) Tail for short flows (b) Avg for large flows

Fig. 3. Performance of BFC with varying incast degrees.




(a) At the congested port (b) At the upstream port

Fig. 4. An example where FLOWSAIL approximates the ideal FC's behavior with only two queues: at the congested port, FLOWSAIL makes the same decision that flows $\{blue, yellow\}$ should be paused, and at the upstream port, FLOWSAIL accurately pauses flows that should be paused (*i.e.*, $\{blue, yellow\}$) while keeping other flows active.

8 queues (BFC-8Q) per port. We use the same settings as in Fig. 2a, and evaluate BFC solely. The incast degrees are varied and we measure the tail FCT slowdown for short flows and the average FCT slowdown for large flows. The results, illustrated in Fig. 3, indicate that BFC-8Q suffers substantial degradation, with up to 11.2× higher tail latency for short flows (Fig. 3a), and 1.7× lower throughput for large flows (Fig. 3b), compared to BFC-32Q. It is worth noting that the transmitting capacity per port remains constant for both 8 and 32 queues; thus, the performance degradation results from severe interference between flows within the same queue, such as HOL blocking and unfair pausing to innocent flows.

## III. FLOWSAIL OVERVIEW

### A. Opportunities

The efficacy of the ideal FC is derived from two key aspects. Firstly, at a congested port, where the input rates surpass the output capacity, the ideal FC accurately identifies the set of flows responsible for the congestion. This is accomplished by verifying whether the queue length associated with each flow exceeds a predetermined threshold, given that each flow is allocated an exclusive queue (left side of Fig. 4a). The ideal FC subsequently transmits control frames to its upstream entities. Secondly, at the ports receiving control frames from downstream entities, the ideal FC independently pauses or resumes the transmission of flows by manipulating the corresponding queue (left side of Fig. 4b).

The opportunity behind FLOWSAIL lies in its ability to approximate the dual-aspect behavior of the ideal FC without requiring per-flow queues. Simultaneously, commodity programmable switches [27], [38], [39] offer the requisite data-plane programmability, such as stateful operation, capability to program the assignment of flows to queues, and independent pausing/resuming each queue within the data plane.

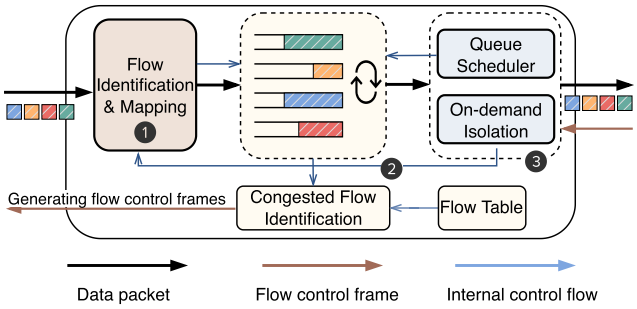



Fig. 5. FLOWSAIL Architecture.

**O1:** At the congested port, if we can monitor each flow's buffer size, we can make control decisions at the per-flow level, the same as the ideal FC, thereby obviating the need for a per-flow queue, as demonstrated on the right side of Fig. 4a.

**O2:** At the upstream ports receiving control frames, given that there are only two flow control actions (*i.e.*, halting or starting flow transmission), if we can redirect all flows that need to be paused to a single paused queue while maintaining other queues active, we can emulate the control action of the ideal FC. Theoretically, only two queues would be necessary, as depicted on the right side of Fig. 4b.

### B. Challenge & Overview

Fig. 5 illustrates the FLOWSAIL architecture, which carefully integrates its components to address the challenges.

**C1: How to define the granularity of congested flow?** As mentioned in §I, FLOWSAIL adopts a similar method to BFC [13] by defining flows at the granularity of *active flows* (❶). FLOWSAIL independently manages different active flows even if they share the same flow identifier, thereby accommodating the non-constant congestion status experienced during a flow's lifespan.

**C2: How to effectively and correctly identify congested flows?** FLOWSAIL employs a *hierarchical method* for identifying congested flows (❷). Specifically, FLOWSAIL initially maps flows to available queues, utilizing the egress queue length as the primary congestion signal. When a queue length exceeds a given threshold and serves more than one flow, FLOWSAIL then uses a per-flow table to facilitate a secondary-level decision. This strategy allows FLOWSAIL to minimize the frequency of complex flow-table calculations while maintaining fine-grained control granularity.

**C3: How to isolate flows and avoid the out-of-order issue?** Upon receiving control frames, FLOWSAIL implements an *on-demand isolation* mechanism, reassigning all flows slated to be paused to a single paused queue while maintaining other queues active (❸). FLOWSAIL also guarantees deterministic in-order delivery by *Order Mark Matching* when scheduling flows, accommodating the go-back-N retransmission scheme prevalent in most commodity RNICs [10], [12], [40]

**Behavior Comparison:** We illustrate a comparison of the behavior among ideal FC, BFC, and FLOWSAIL under a

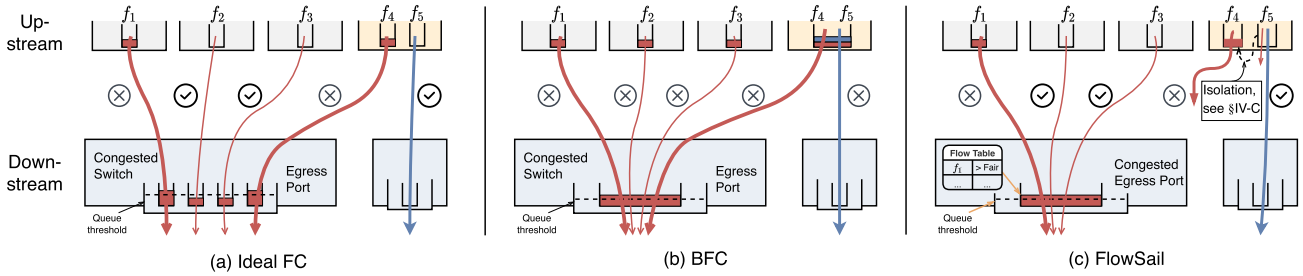



Fig. 6. An example showing behaviors of ideal FC, BFC, and FLOWSAIL. BFC unnecessarily pauses uncongested ($f_2$ and $f_3$) and victim flows ($f_5$). FLOWSAIL precisely pauses the congestion-responsible flows while leaving other flows unaffected, consequently approximating the ideal FC's behavior.

specific scenario depicted in Fig. 6, where four flows ($f_1 \sim f_4$) with varying arrival rates traverse a congested egress port. As the ideal FC assigns a dedicated queue to each flow, flows are independently controlled, *i.e.*, the congestion-responsible flows $f_1, f_4$ are paused, while others remain unimpeded. In the case of BFC and FLOWSAIL, we assume that only one queue is available at the congested egress port, and $f_5$ is initially mapped to the same upstream egress queue as $f_4$. Given that BFC *regulates traffic within a queue collectively*, all flows $f_1 \sim f_5$ are paused, leading to an unfair degradation of $f_2, f_3, f_5$. In contrast, FLOWSAIL correctly pauses flows $f_1$ and $f_4$ at the congested port while leaving $f_2$ and $f_3$ unaffected. Additionally, at the upstream port, FLOWSAIL isolates $f_4$ to an isolation queue while keeping $f_5$ uninterrupted. As a result, FLOWSAIL approximates the behavior of the ideal FC.

## IV. FLOWSAIL DESIGN

We first introduce the flow identification and mapping of flows to queues in §IV-A, then elaborate on the hierarchical congested flow identification in §IV-B, and finally describe the control frame handling and on-demand isolation in §IV-C.

### A. Flow Identification and Mapping

Algorithm 1 presents the methodology for identifying flows and establishing flow-to-queue mapping logics. Flows fall into two categories: *congested* and *normal*. Congested flows are allocated to a specially reserved isolation queue, denoted as `rsvQ` (Line 3). The set of congested flows is recognized by control frames from downstream entities (thoroughly described in §IV-C). Regarding the mapping of normal flows, FLOWSAIL inherits the approach used in BFC [13]. FLOWSAIL dynamically assigns a new flow to an empty egress queue when available by maintaining a bitmap of empty queues at the ingress pipeline. If all queues are currently engaged, FLOWSAIL assigns the flow to a random queue, essentially implementing the stochastic fair queuing [41]. For old flows (*i.e.*, those with packets already queued at the switch), subsequent packets are deposited into the same FIFO queue (Line 7∼16).

Each flow is distinguished by a unique 5-tuple, consisting of source and destination address, port, and IP protocol, which are collectively referred to as the flow identifier (FID). To track flow size and queue assignment, FLOWSAIL manages a *flow table* indexed by the hash of the FID. Congested and normal flows are tracked by congested flow table (`CFT`) and normal flow table (`NFT`), respectively. Each entry in `NFT` maintains the following states: the assigned queue (`qIdx`), the number of buffered packets (`size`), and the number of packets that have encountered congestion (`pauseNum`, used in §IV-B). The implementation details of the flow table are in §V.

**Switching of flow between `CFT` and `NFT`.** Once a flow is identified as congested, its FID is recorded in the `CFT`, and subsequent packets from this flow are tracked by the `CFT`. Note that packets already buffered will remain in the original[3] queue until they are completely dequeued. When a previously congested flow no longer experiences congestion, it is removed from the `CFT`, and its subsequent packets are enqueued in the normal queues once again. The condition for releasing a flow from the `CFT` is detailed in §IV-C. The determination of whether a flow is congested is made by checking its presence in the `CFT`, as outlined in Line 3.

### B. Hierarchical Congested Flow Identification

FLOWSAIL adopts a *hierarchical congested flow identification* approach, facilitating per-flow level control granularity. Specifically, FLOWSAIL initially employs the queue length as a primary congestion signal. When a queue length exceeds a given threshold and serves more than one flow, FLOWSAIL utilizes the states in the *flow table* to make a secondary-level decision. Utilizing the queue length as a triggering condition reduces the frequency of flow-table calculation. Lines 17∼25 in Algorithm 1 illustrate this hierarchical identification process. Delving into the details of the secondary-level decision, FLOWSAIL uses the recorded flow's `size` in the flow table to verify whether it exceeds the fair sharing size ($S_{fair}$) of the congested egress queue (Line 23). When a flow exceeds $S_{fair}$, it is identified as the cause of the congestion and should be paused.

In addition to the flow table, FLOWSAIL manages a queue table (`QT`), indexed by the port number and queue index. Each entry in `QT` retains the following states: the number of active flows (`flowNum`), and the count of received control frames (`cfNum`, used in §IV-C). The `flowNum` increases by 1 when a new flow arrives as shown in Line 15, and decreases by 1 when the switch has drained off a flow's packets (Algorithm 2, Line 15).

To calculate $S_{fair}$, FLOWSAIL employs the logarithm, which can be implemented by counting the number of non-zero bits of data, and the hardware-friendly

[3]The term `CFT/NFT` means that the congested flows use `CFT`, and the normal flows use `NFT`. The real implementation can use the P4's intrinsic metadata to distinguish these two types of packets.

**Algorithm 1** Enqueueing, Flow-to-Queue Mapping, Congested Flow Identification, and PAUSE Generation

**Inputs:** $\mathbb{NFT}$: The flow table for normal flows; $\mathbb{CFT}$: The flow table for congested flows; $\mathbb{QT}$: The queue table; $Q_l$, $Q_h$: The two queue length thresholds; $rsvQ$: The queue reserved for congested flows;

1: **function** ENQUEUE($pkt$)
2: $key$ = hash($pkt.FID$)
▷ *Identify flow and map it to queue*
3: **if** $key$ is in $\mathbb{CFT}$ **then** ▷ *Congested flow*
4: $pkt.qIdx$ = $rsvQ$
5: $\mathbb{CFT}[key].size$ += $pkt.size$
6: **else**
7: **if** $\mathbb{NFT}[key].size \neq 0$ **then** ▷ *Old flow*
8: $pkt.qIdx$ = $\mathbb{NFT}[key].qIdx$
9: **else** ▷ *New flow*
10: **if** empty $q$ available at $pkt.egressPort$ **then**
11: $\mathbb{NFT}[key].qIdx$ = emptyQ
12: **else**
13: $\mathbb{NFT}[key].qIdx$ = randomQ
14: $pkt.qIdx$ = $\mathbb{NFT}[key].qIdx$
15: $\mathbb{QT}[pkt.qIdx].flowNum$ += 1 ▷ *Update QT*
16: $\mathbb{NFT}[key].size$ += $pkt.size$
▷ *Hierarchical congested flow identification*
17: **if** $pkt.qIdx.length > Q_h$ **then**
18: $\mathbb{CFT}/\mathbb{NFT}[key].pauseNum$ += 1 [1]
19: $pkt.ifCongested$ = $true$
20: **else**
21: **if** $pkt.qIdx.length > Q_l$ **then**
22: Calculate $S_{fair}$, the the fair sharing size at $pkt.qIdx$
▷ *Per-flow level control decision*
23: **if** $\mathbb{CFT}/\mathbb{NFT}[key].size > S_{fair}$ **then**
24: $\mathbb{CFT}/\mathbb{NFT}[key].pauseNum$ += 1
25: $pkt.ifCongested$ = $true$
26: **if** $\mathbb{CFT}/\mathbb{NFT}[key].pauseNum == 1$ **then**
27: send PAUSE($pkt.FID$) ▷ *Pause upstream entity*

**Algorithm 2** Dequeueing and Resume Generation

**Inputs:** $\mathbb{NFT}$: The flow table for normal flows; $\mathbb{CFT}$: The flow table for congested flows; $\mathbb{QT}$: The queue table; $rsvQ$: The queue reserved for congested flows;

1: **function** DEQUEUE($pkt$, $qIdx$) ▷ *$qIdx$: the dequeueing queue*
2: **if** $qIdx == rsvQ$ **then**
3: $key$ = hash($pkt.FID$)
4: **if** $pkg$ is an OrderMark packet **then**
5: **if** $\mathbb{NFT}[key].size \neq 0$ **then**
6: $\mathbb{QT}[rsvQ].cfNum$ += 1 ▷ *Pause rsvQ, waiting for a matched OrderMark*
7: Set state $OM\_waiting$ as $pkt.FID$
8: **else**
9: $\mathbb{CFT}[key].size$ -= $pkt.size$
10: **if** $pkt.ifCongested == true$ **then**
11: $\mathbb{CFT}[key].pauseNum$ -= 1
12: **if** $\mathbb{CFT}[key].pauseNum == 0$ **then**
13: send RESUME($pkt.FID$)
14: **if** $\mathbb{CFT}[key].size == 0$ **then**
15: $\mathbb{QT}[qIdx].flowNum$ -= 1
16: **if** $\mathbb{CFT}[key].status == resumed$ **then** ▷ *Checking if to release this flow from $\mathbb{CFT}$*
17: Delete $key$ from $\mathbb{CFT}$
18: **else**
19: $key$ = hash($pkt.FID$)
20: **if** $pkg$ is an OrderMark packet **then**
21: **if** $pkg.FID == OM\_waiting$ **then** ▷ *If not matching or $OM\_waiting$ is invalid, do nothing.*
22: $\mathbb{QT}[rsvQ].cfNum$ -= 1
23: Set state $OM\_waiting$ as invalid
24: **else** ▷ *Same as above, omitted as space limited*
25: Update $\mathbb{NFT}[key].size$, $\mathbb{NFT}[key].pauseNum$, $\mathbb{QT}$ $[qIdx].flowNum$, and check if to RESUME($pkt.FID$)

shifting operations:

$$S_{fair} = Q \gg \lceil \log_2(QT[qIdx].flowNum) \rceil, \quad (1)$$

where $Q$ represents the egress queue length. $S_{fair}$ signifies the fair allocation of queue capacity among all active flows. In situations where the number of active flows is not a power of two, the logarithm of this number is rounded up to avoid under-regulation of flows. Note that this calculation assumes flows have the same weight. If flows have different weights, calculating $S_{fair}$ could potentially result in higher overhead.

**Thresholds.** FLOWSAIL permits flows not exceeding their sharing size to continue transmitting, even when they traverse a congested port. However, this could lead to a buffer overflow in extreme cases. For instance, when a large flow shares a queue with many bursty flows (each occupying a few packets), the bursty flows are under-regulated but the accumulation of them could trigger a buffer overflow. Therefore, we establish two thresholds, $Q_l$ and $Q_h$, to prevent the congested queue from becoming overloaded. When queue length $Q$ surpasses $Q_l$ but remains below $Q_h$, FLOWSAIL only sends PAUSE to flow that occupies more than $S_{fair}$ (Line 21∼25). Note that each flow's buffer size is recorded in *flow table* (mentioned in §IV-A). If $Q$ exceeds the conservative $Q_h$, FLOWSAIL pauses all passing flows to avoid severe buffer overflow (Line 17).

We set the pause thresholds $Q_l$ and $Q_h$ to 1-Hop BDP and 3-Hop BDP, respectively. Let $N_{active}$ denotes the number of non-paused queues in the port, $HRTT$ signifies the 1-Hop RTT to the upstream, and $\mu$ represents the port capacity. Thus, $Q_l$ and $Q_h$ are determined by $HRTT * \mu/N_{active}$ and $3 * HRTT * \mu/N_{active}$, respectively. A pre-configured match-action table indexed with $N_{active}$ and $\mu$ can calculate these values efficiently.

The primary intuition behind setting $Q_l$ is to minimize queueing delay. We configure $Q_l$ as 1-Hop BDP, ensuring that the switch can drain off packets within 1-Hop RTT if the PAUSE takes effect immediately. $Q_l$ is unlikely to lead to under-utilization of the link capacity since it selectively pauses flows. On the other hand, the purpose of configuring $Q_h$ is to tackle (as a safeguard) the extreme congestion, while also avoiding link under-utilization. Therefore, $Q_l$ is set to a relatively large value, the 3-Hop BDP. The sensitivity of these configurations is examined in §VI-C.

### C. Control Frame Handling & On-Demand Isolation

FLOWSAIL pauses an upstream flow when it is identified as congested and resumes it when congestion subsides. This process is monitored via the `pauseNum` counter in the flow table. As a packet enqueues, if it is identified as a contributor to congestion, the `pauseNum` increases by 1 (Lines 17∼25). When the packet (*i.e.*, the contributor) leaves the switch, the `pauseNum` decreases by 1

**Algorithm 3** Control Frame Handling

**Inputs:** $\mathbb{NFT}$: The flow table for normal flows; $\mathbb{CFT}$: The flow table for congested flows; $\mathbb{QT}$: The queue table; $rsvQ$: The queue reserved for congested flows;

```
1:  function ReceiveControlFrame(f)
2:      cf_key = hash(f.cFID)                    ▷ The FID to be paused
3:      if f == PAUSE then
4:          if cf_key is not in CFT then         ▷ New congested flow
5:              CFT[cf_key].size = 0
6:              CFT[cf_key].status = paused
7:              if NFT[cf_key].size ≠ 0 then     ▷ In-order support
8:                  Enqueue OrderMark to rsvQ and
                    NFT[cf_key].qIdx
9:                  Pause NFT[cf_key].qIdx       ▷ Original queue
10:         else
11:             CFT[cf_key].status = paused
12:         QT[rsvQ].cfNum += 1
13:     else                                     ▷ f = RESUME
14:         QT[rsvQ].cfNum -= 1
15:         if NFT[cf_key].size ≠ 0 then
16:             Resume NFT[cf_key].qIdx
17:         CFT[cf_key].status = resumed
18:         if CFT[cf_key].size == 0 then
19:             Delete cf_key from CFT           ▷ Release it from CFT
```

(Algorithm 2, Lines 10∼11). FLOWSAIL generates a PAUSE frame only when the `pauseNum` transitions from 0 to 1, and a RESUME frame when it switches from 1 to 0 (Algorithm 1, Line 26; Algorithm 2, Line 12), conserving bandwidth used for sending control frames. Both PAUSE and RESUME frames carry the flow's FID.

Given the current limitations of switch ASICs, flow control operations (*i.e.*, stopping or starting flow transmission) must be facilitated with the help of physical queues. Therefore, FLOWSAIL tries to approximate the ideal FC's behavior by rerouting all paused flows to a single paused queue (`rsvQ`), while keeping other queues active.

Algorithm 2 elaborates on the process of isolation. FLOWSAIL employs a dedicated flow table (`CFT`) indexed by the hash of the FID, to track the states of congested flows. Each entry in `CFT` includes the number of packets in `rsvQ` (`size`), the flow status (`status`), the `pauseNum` (as previously explained), and the `FID`. Upon receiving a PAUSE frame containing the FID of a congested flow (denoted as $cFID$), FLOWSAIL verifies the presence of $cFID$ in `CFT` by checking if `cFID == CFT[hash(cFID)].FID`. If not, FLOWSAIL adds it to `CFT` (Line 4∼6). Subsequent packets from $cFID$ are then directed to `rsvQ`, as described in §IV-A.

**Queue Scheduler.** FLOWSAIL's queue is regulated by the `cfNum` in the queue table `QT`, counting the number of received control frames. Algorithm 3 elaborates the process of control frame handling. Each PAUSE frame increments the counter by 1, and a RESUME frame decreases it by 1 (Algorithm 3 Line 12, 14). FLOWSAIL pauses a queue when `cfNum` increases from 0 to 1, and resumes it when the count decreases from 1 to 0.

**In-order Support.** When FLOWSAIL attempts to add a flow to `CFT` and that flow has buffered packets in original queue (*e.g.*, $q_1$), FLOWSAIL also pauses $q_1$ to prevent these buffered packets from exacerbating downstream congestion

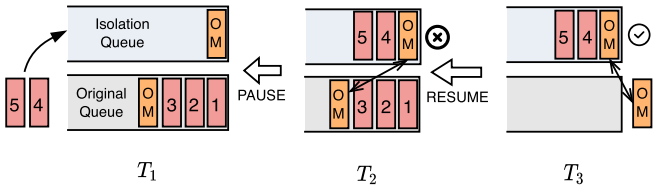



Fig. 7. A high-level view of Order Mark Matching (*i.e.*, how Order Mark packets support in-order delivery).

(Line 9). Conversely, a RESUME frame resumes $q_1$ too (Line 16). Although pausing $q_1$ may result in an unfair degradation for innocent flows in $q_1$, we consider this to be insignificant because the re-direction of congested flow to `rsvQ` prevents $q_1$ being paused continuously in the future. Using a 2-flow case for an example, assuming a congested flow ($f_1$) and innocent flow ($f_2$) initially share $q_1$. If there is no `rsvQ`, $f_1$ will continue arriving at $q_1$, so $q_1$ (and $f_2$) will be paused many times until CC eventually reduces $f_1$'s rate. In contrast, FLOWSAIL redirects the subsequent packets of $f_1$ to `rsvQ`, preventing $q_1$ (and $f_2$) from being paused constantly in the future.

Besides, FLOWSAIL enqueues two Order Mark (OM) packets containing the flow's FID to $q_1$ and `rsvQ` (Line 7∼8) to facilitate the in-order delivery. The OM packet in `rsvQ` must wait for another matched OM before its transmission begins. We name this scheme *Order Mark Matching*, with a high-level view shown in Fig. 7 and implementation details in §V.

**Configuration of isolation queue and release of congested flow.** The `rsvQ` is configured statically and solely used for accommodating congested flows. This means it cannot be used for holding normal flows, even when it is empty; this is a simplification that facilitates hardware implementation. A congested flow is removed from `CFT` once it is resumed and has no buffered packets in `rsvQ` (Line 17, 19). To prevent frequent addition and deletion of the same flow in `CFT`, we incorporate a timestamp in `CFT` which is updated with every packet enqueue or dequeue operation. Thus, a congested flow is only released when it is resumed, has no buffered packets, and the timestamp exceeds a given threshold. Since the congested port can drain off packets for one HRTT, we set this threshold to a small multiple of the HRTT (2 HRTT).

### D. Compatibility With Priority-Based Scheduling and PFC

Datacenter operators typically partition application traffic into distinct traffic groups, each assigned different priorities. Correspondingly, packets are enqueued into separate switch queues managed by priority-based scheduling mechanisms [5], [42]. This approach is commonly adopted in modern datacenters to cater to applications with varying QoS requirements. The FLOWSAIL framework is inherently compatible with priority-based scheduling.

Specifically, we can statically allocate each queue on a port to a specific priority level, creating multiple queue groups, each comprising a minimum of two queues sharing the same priority level. Upon packet arrival, we can extract its priority level and subsequently map it to a physical queue associated with its specified priority level. Each queue group performs

fair scheduling and independently employs the FLOWSAIL scheme as its FC mechanism, while the switch conducts priority-based scheduling among these groups. We conducted an experiment to illustrate the performance of combining priority-based scheduling and FLOWSAIL in §VI-C.

As an FC scheme, FLOWSAIL does not necessitate exclusive deployment and can integrate with other FC schemes, such as PFC [20]. This is because the fundamental control actions of FLOWSAIL, namely PAUSE and RESUME for a specific queue, align with those of PFC. Hence, the integration of PFC and FLOWSAIL is feasible. For instance, in a three-layer fat-tree topology, we can deploy FLOWSAIL selectively to ToR switches suffering from congestion and to their connected aggregation switches, while retaining the default PFC scheme for other switches. When congestion arises at FLOWSAIL-managed ToR switches, they manage their upstream aggregation switches using FLOWSAIL. Subsequently, if congestion propagates to the aggregation switches, they resort to PFC to manage their upstream switches. This compatible deployment strategy mitigates deployment costs and upgrade risks.

## V. IMPLEMENTATION

**Hardware Feasibility.** As mentioned in §II-B, FLOWSAIL implementation requires certain capabilities in programmable switches: ($i$) line-rate stateful operations support; ($ii$) at least two FIFO queues per port with flow-to-queue programmability; ($iii$) dynamic pausing and resuming of each queue at the data-plane. The newest Tofino2 ASIC [39] meets all these requirements. However, due to a lack of access to Tofino2, we implemented FLOWSAIL using Tofino [27] (that satisfies the first two conditions only) with an approximation of the 3rd condition. Specifically, we observe that, in terms of a UDP flow, pausing it and discarding all its packets when it should be paused have the same effect on its *average throughput*. Consequently, we chose to evaluate UDP flows in testbed experiments and replicate the effect of data-plane pausing on their average throughput using data-plane dropping. Note that this method is solely for the purpose of completing a prototype validation; it is not a fully comprehensive implementation and cannot resemble the overall RDMA network. In §VI-A, we complement simulations to cross-validate the testbed results.

**FLOWSAIL Pipeline.** Fig. 8 illustates the pipeline of FLOWSAIL implementation. Upon into the ingress pipeline, packets are classified into three types: data packets, control frames, and mirror packets. For data packets, the *Flow-to-Queue Mapping* calculates the assigned queue, and then *Flow Control Decision* determines whether to send a PAUSE frame. The resuming decision is made in the egress pipeline. Control frames, after being classified, are used to update the `CFT` and are then converted into an Order Mark, which becomes two by mirroring using the Tofino module *Mirror*. A new control frame is generated by mirroring the data packet's header and then is constructed in the *Control Frame Constructor* at the egress pipeline. Both the Order Mark and control frame are assigned specific Ethernet types for classification.

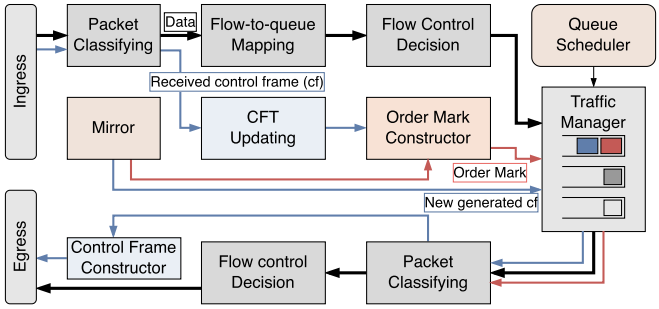



Fig. 8. Pipeline of FLOWSAIL implementation. The black, blue and red lines represent the data packets' paths, control frames' paths, and Order Marks' paths, respectively.

**Flow Table Implementation.** A flow table in FLOWSAIL comprises multiple register arrays, with each array representing an entry value. Each entry in FLOWSAIL flow table contains a maximum of 50 bits. Considering a common scenario with 10K active flows per switch [43], FLOWSAIL requires a total of 64KB memory. We use Cuckoo Hash [44] as the hash function, allowing for load factors of approximately 70%. As a result, we can statically set the overall size of the flow table to 100KB. Note that when a hash collision occurs (which should be rare), two flows that hash to the same entry are treated as a single flow with shared flow states.

**Order Mark Matching.** We facilitates the in-order delivery through a customized *Order Mark (OM) Matching* scheme. Specifically, when dequeuing an OM from `rsvQ`, the switch checks whether the flow ($OM.FID$) has buffered packets in normal queues. If so, $OM.FID$ is recorded, and the `pauseNum` of `rsvQ` is reduced by 1; this reduction requires `rsvQ` must wait for a matched OM. When an OM is dequeued from normal queues, and if a flow is waiting for matched OM, and matches $OM.FID$, the `pauseNum` of `rsvQ` is increased by 1. Note that this is not fundamental to FLOWSAIL as Tofino2 [39] has natively supported the *Order Mark Matching* feature in its *Advanced Flow Control* module.

## VI. EVALUATION

We provide a proof-of-concept validation of our FLOWSAIL implementation and extensively compare the performance of FLOWSAIL against previous schemes using large-scale NS-3 [32] simulations. Our evaluation seeks to answer:

- **What is the efficiency of FLOWSAIL prototype?** Our results of the testbed microbenchmark demonstrate that FLOWSAIL operates at a 100Gbps line rate and effectively mitigates congestion at a per-flow granularity (§VI-A).
- **How does FLOWSAIL perform under realistic workloads in large-scale DCNs?** Large-scale simulations illustrate that FLOWSAIL capably manages bursty traffic. FLOWSAIL reduces latency (by up to 15×) for short flows and improves throughput for large flows, compared to existing FC and CC schemes (§VI-B).
- **How sensitive is FLOWSAIL to various traffic patterns and parameters?** Our detailed simulations reveal that FLOWSAIL exhibits resilience in the face of changes in workload distributions, incast degrees, number of available queues, and threshold settings (§VI-C).

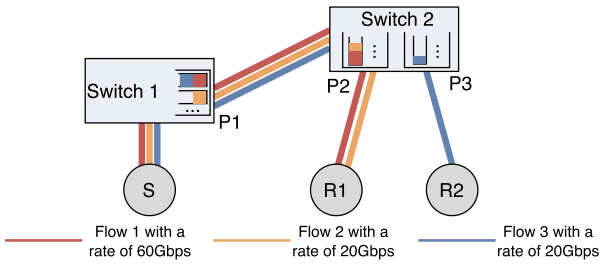



Fig. 9. Testbed topology. The depicted $f_1 \sim f_3$ are used to evaluate the per-flow control granularity of FLOWSAIL.

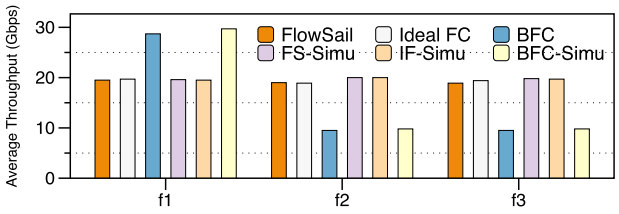



Fig. 10. Throughput of flows $f_1 \sim f_3$ depicted in Fig. 9, with simulations cross-validating the testbed results.

### A. Testbed Micro-Benchmark

We establish a testbed topology comprising two Wedge 100BF-32X [27] switches[4] and three servers, each equipped with a Mellanox ConnectX-5 100GbE RNIC (Fig. 9). All ports are 100Gbps and possess 8 queues. We evaluate UDP flows in the testbed experiment and use simulations to cross-validate the results, as described in §V.

**Operating Efficiency.** We initially enable two long-running flows $\{S-R_1, R_2-R_1\}$ that compete at the $P_2-R_1$ link, and measure their aggregated throughput at $R_1$. We measured that the aggregated throughput could reach 96∼98 Gbps, signifying that FLOWSAIL can operate at a 100Gbps line rate.

**Per-flow Control Granularity.** Next, we evaluate the per-flow control granularity offered by FLOWSAIL. As depicted in Fig. 9, we enable flows $f_1$ and $f_2$ route from $S$ to $R_1$ while $f_3$ goes from $S$ to $R_2$. The $f_1$, $f_2$, and $f_3$, start with rates 60Gbps, 20Gbps, and 20Gbps, respectively. We explicitly restrict the draining capacity of port $P_2$ to 40Gbps, thereby $P_2$ becoming a bottleneck as its input rate significantly exceeds its draining capacity. We compare FLOWSAIL with BFC [13] and ideal FC [25]. For the BFC and FLOWSAIL evaluation, we initially assign $f_1$ and $f_3$ to share the same egress queue on $P_1$, while $f_1$ and $f_2$ share the same egress queue on $P_2$.

We measure the average throughput of $f_1 \sim f_3$ and show the result in Fig. 10. Ideal FC controls flows independently, pausing only $f_1$ (which exceeds $P_2$'s fair sharing size) and maintaining $f_2$ and $f_3$ rates at 20Gbps. BFC, managing traffic collectively within the same queue, simultaneously pauses $f_1$ and $f_2$ when $P_2$ is congested, and $f_3$ is also paused because it shares an upstream queue with $f_1$, causing about 50% unfair degradation to $f_2$ and $f_3$. Conversely, FLOWSAIL performs similarly to ideal FC, correctly identifying that $f_1$ contributes to the $P_2$'s congestion and only pauses $f_1$'s upstream queue, thus maintaining $f_2$ unaffected. Furthermore, when the PAUSE frame is propagated to $P_1$, FLOWSAIL isolates $f_1$ to a paused isolation queue, ensuring $f_3$'s throughput is unaffected.

We use simulations, where we implement data-plane pausing and resuming, to cross-validate the testbed results. The simulation results (shown in Fig. 10) are highly consistent with the testbed results, validating the reliability of our prototype.

[4]Since we only have access to a single Wedge 100BF-32X switch, we emulate two switches by directly connecting two switch ports, which essentially involves sending dequeued packets back into the switch.

### B. Large-Scale Simulations

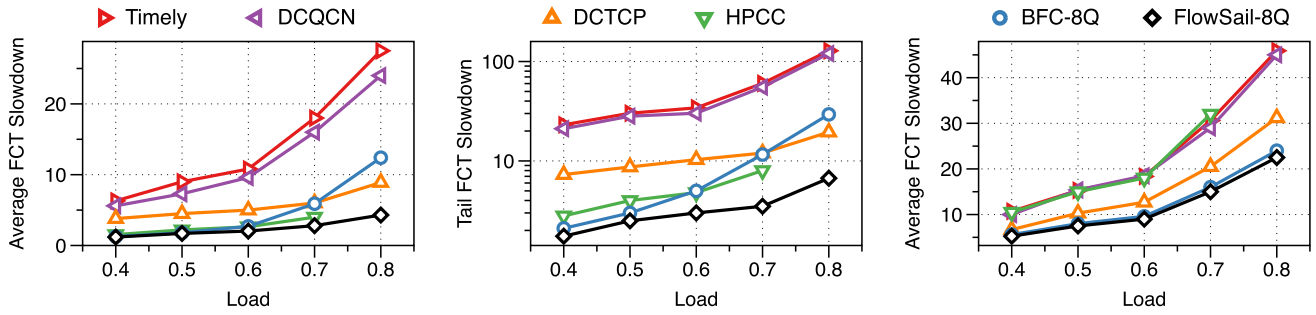
**Network Topology.** We simulate a 3-layer fat-tree topology, composed of 64 top-of-the-rack (ToR) switches, 64 Spine switches, 64 Core switches, and 1024 servers, with an oversubscription ratio of 2:1. Each link is 100Gbps and has a propagation delay of 1$\mu$s, yielding a 1-Hop RTT of 2$\mu$s. The switches use a standard shared memory model with a total buffer of 12MB, and use the Equal-Cost Multi-Path (ECMP) as load balancing and Go-Back-N for retransmission.

**Workloads and Metrics.** We construct three realistic workloads including Web Search [9], Web Server [28] and Google RPC [29]. Specifically, the first two workloads comprise a mix of large and short flows, around 70% and 20% of flows under 10KB, respectively. The Google RPC workload is predominantly composed of short flows (97% of flows are less than 10KB). The synthesized flows follow a Poisson process with sources and destinations randomly selected. We measure different loads, where X% denotes X% load on the core links. We supplement these with a 50-to-1 incast traffic, with flow sizes randomly selected between 50KB to 200KB and a total load of (0.1*X)%. Our primary metric is FCT slowdown, supplemented by FCT and queue length.

**Comparisons.** We compare FLOWSAIL with four end-to-end transport protocols: DCQCN [10], Timely [31], HPCC [12], and DCTCP [9], configuring their parameters according to their papers. By default, end-to-end CC is evaluated with PFC enabled. We also examine the state-of-the-art FC scheme, BFC. We primarily evaluate BFC and FLOWSAIL, each equipped with 8 queues, denoted as BFC-8Q and FLOWSAIL-8Q. A comprehensive evaluation of BFC and FLOWSAIL with different queue numbers is conducted in §VI-C.

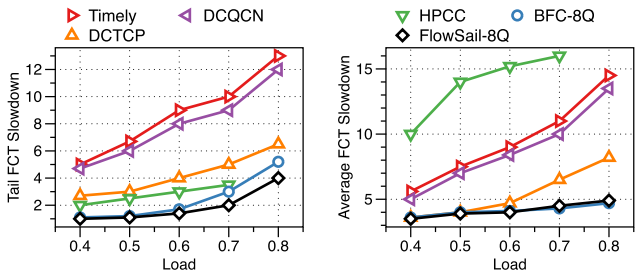
**Performance.** We examine FLOWSAIL against comparisons under a Web Server distribution with a load varying from 0.4 to 0.8, with and without incast traffic. Evaluations under Web Search and Google RPC are detailed in §VI-C.

*(i) Web Server workload with incast.* We evaluate the average and tail ($99^{th}$ percentile) FCT slowdown for short flows (< 10KB) and average FCT slowdown for large flows (> 100KB). As depicted in Fig. 11, FLOWSAIL significantly outperforms BFC and all end-to-end CCs in terms of latency for short flows and throughput for large flows. Specifically, FLOWSAIL achieves a remarkable reduction of up to 19×, 17.9×, 3.2×, 2.2×, 4.3× in tail latency for short flows compared to Timely, DCQCN, DCTCP, HPCC, and BFC, respectively (Fig. 11b). This performance enhancement is attributed to FLOWSAIL's precise control over congested




(a) Average FCT slowdown for short flow (b) Tail FCT slowdown for short flow (c) Average FCT slowdown for large flow

Fig. 11. Performance under Web Server distribution with different load (X%) and (0.1*X)% 100-to-1 incast traffic.




(a) Tail slowdown for short flows (b) Avg slowdown for large flows

Fig. 12. Performance under Web Server distribution without incast traffic.

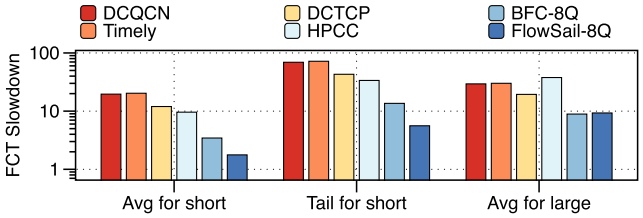



Fig. 13. Performance under Google RPC distribution with 60% load and 6% incast.

flows, preventing unfair blocking and pausing of innocent flows. Furthermore, the performance advantage of FLOWSAIL increases with increasing load. HPCC becomes unstable at 80%, and thus we exclude its value at that load.

*(ii) Web Server workload without incast.* We repeat the previous experiment in Fig. 11 without the incast traffic and present results in Fig. 12. Without incast traffic, the performance gap between FLOWSAIL and other schemes narrows as the bursty congestion and contention to physical queue issues mitigate. Despite this, FLOWSAIL outperforms all other schemes, *e.g.*, achieving up to 3.1×, 3×, 1.6×, 1.75×, 1.3× lower tail latency for short flows compared with Timely, DCQCN, DCTCP, HPCC, and BFC, respectively (Fig. 11b). FLOWSAIL also maintains higher throughput for large flows, such as up to 2.2× higher and comparable throughput compared with HPCC and BFC, respectively. Note that the poor performance of HPCC on large flow is due to HPCC explicitly trading off throughput for latency and INT header consuming extra bandwidth, as also mentioned in the HPCC paper [12].

### C. FLOWSAIL *Deep Dive*

**Performance under Google RPC distribution.** We evaluate FLOWSAIL under the Google RPC distribution, which is set to a 60% load with a 6% incast traffic. We compare FLOWSAIL against all the aforementioned approaches, and calculate the average and tail latency for short flows (< 3KB, about 92%) and average latency for large flows (> 100KB). As Fig. 13 reveals, FLOWSAIL outperforms all comparisons. For instance, FLOWSAIL achieves 12.8×, 12.3×, 7.6×, 6.1×, 2.4× lower tail latency for short flows compared to Timely, DCQCN, DCTCP, HPCC, and BFC, respectively. Furthermore, for large flows, FLOWSAIL sustains a higher throughput against all end-to-end CCs, such as 4× against HPCC. Note that while FLOWSAIL's throughput for large flows is 5% lower compared to BFC, this is due to BFC allowing large flows to seize capacity from other innocent flows within the same queue. In contrast, FLOWSAIL prevents this unfair seizure by accurate per-flow level identification and isolation.

**Performance under Web Search distribution.** We first evaluate FLOWSAIL under the Web Search distribution with a fixed 60% load and 6% incast traffic, and then measure the throughput of large flows with various traffic loads.

*(ii) Web Search distribution with a fixed load.* The Web Search distribution is characterized by a large number of concurrent long-running flows, resulting in the contention of queues with short flows. Thus, BFC exhibits poor tail latency for short flows, as the presence of long-running flows degrades the short flows in the same queue. In this context, reducing the rates of long-running flows greatly benefits short flows, leading to lower latency for DCQCN and DCTCP compared to BFC. In contrast, FLOWSAIL outperforms DCQCN and DCTCP by achieving a 2.7× reduction in latency of short flows; this is because FLOWSAIL's fine-grained control prevents innocent pausing and blocking of short flows. As explained later, FLOWSAIL could potentially decrease throughput for large flows, compared to BFC. Therefore, we recommend using FLOWSAIL in conjunction with end-to-end CC to simultaneously handle the transient bursts and long-running flows. Fig. 14 illustrates that CC+FLOWSAIL yields the best performance.

*(ii) Throughput with various loads.* We vary the average load from 0.4 to 0.8 and measure the average FCT slowdown for large flows. As shown in Fig. 15, FLOWSAIL experiences reduced throughput for large flows across all load levels. This

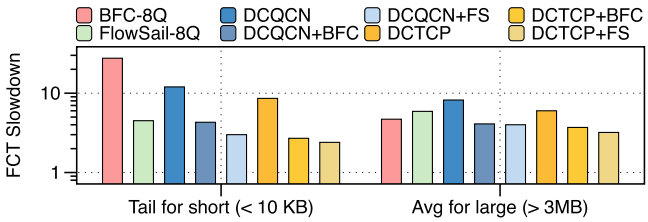



Fig. 14. Performance under Web Search distribution with 60% load and 6% incast.

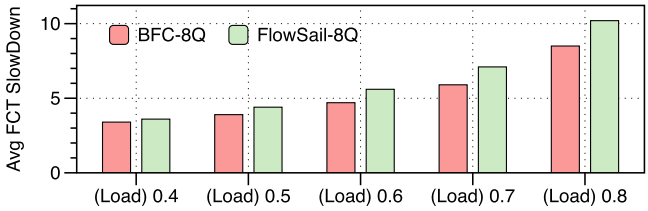
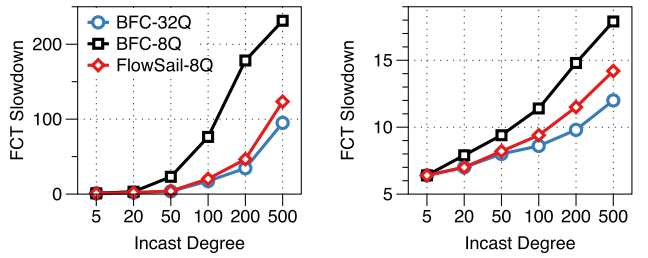



Fig. 15. Average throughput for large flows (> 3MB) under Web Search distribution with various loads.




(a) Tail slowdown for short flows (b) Avg slowdown for large flows

Fig. 16. Performance under Web Server distribution with a fixed 60% load and varying incast degrees.

is because large flows in BFC could unfairly occupy buffer space, leading to improved throughput. In contrast, FLOWSAIL ensures fairness through isolation. Notably, FLOWSAIL actually compresses congested flows' space rather than that of large flows, so there is no absolute trade-off between throughput and latency. To illustrate, the throughput gap between FLOWSAIL and BFC in the Web Server distribution (Fig. 11c) is minimal. However, in Web Search, where large flows dominate, a higher percentage of large flows experience congestion, resulting in more significant degradation.

**Impact of incast degree.** We compare the performance of FLOWSAIL-8Q against BFC-8Q and BFC with 32 queues (BFC-32Q), under a Web Server distribution with a 60% fixed load and varying incast degrees. The results shown in Fig. 16 show FLOWSAIL is more resilient to incast degrees compared to BFC; FLOWSAIL achieves comparable performance with 4× fewer queues (8 *vs.* 32). Under the same number of queues, FLOWSAIL-8Q attains 3.0×∼3.8× lower tail latency for short flows when the incast degree exceeds 100, and 1.3× higher throughput for large flows, compared to BFC-8Q.

**Impact of the number of available queues.** The experiment in Fig. 16 is repeated with a fixed 50-to-1 incast degree, but the number of queues is varied from 2 to 128. We measure the average FCT for all flows. As Fig. 17 reveals, FLOWSAIL is more resilient to the number of queues compared to BFC, achieving satisfactory performance even with a minimum of

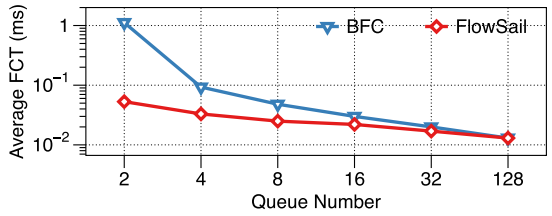



Fig. 17. Performance under Web Server distribution with a varying number of available queues.

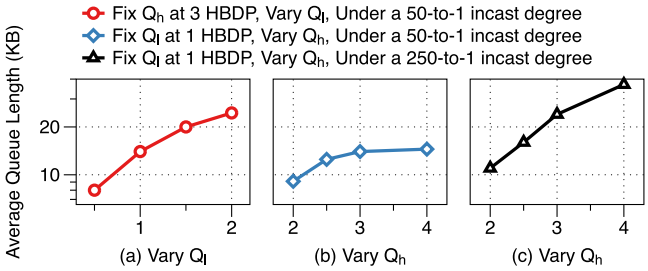



Fig. 18. Average queue length with varying pausing thresholds.

2 queues. For instance, FLOWSAIL experiences a more gentle degradation compared to BFC when the number of queues reduces and achieves up to 20× lower FCT than BFC at 2 queues. Note that configurations with different queue numbers possess the same transmitting capacity per port, so the performance degradation arises from the severe interference between flows within the same queue.

**Parameter sensitivity.** We conduct evaluations to ascertain how the two pausing thresholds $Q_h$ and $Q_l$ influence the performance of FLOWSAIL. First, we fix $Q_h$ at 3 Hop-BDP (HBDP) and vary $Q_l$ (Fig. 18a), then, we fix $Q_l$ at 1 HBDP and adjust $Q_h$ (Fig. 18b), under a 50-to-1 incast degree. The results suggest that $Q_l$ primarily impacts the control of queue length, with a smaller $Q_l$ leading to a shorter queue length. Once $Q_l$ effectively manages congestion, $Q_h$ has little effect; for instance, no queue length difference occurs between 3 and 4 HBDP $Q_h$. Nevertheless, $Q_h$ remains necessary as a safeguard in extreme cases. For instance, in Fig. 18c, which illustrates an extreme 250-to-1 incast degree, $Q_h$ significantly influences queue length control. It is crucial to note that the continuous shallow (or even zero) queue length can lead to under-utilization of bandwidth [12], [13], [16]. Hence, we set the values of $Q_l$ and $Q_h$ to 1 and 3 HBDP respectively, rather than other lower values.

**Compatibility with priority-based scheduling.** As mentioned in §IV-D, FLOWSAIL is inherently compatible with priority-based scheduling. We conducted an experiment to demonstrate the performance of combining FLOWSAIL with priority-based scheduling. We configure a port with 16 queues and statically allocate these queues into 4 priority groups (4 queues per group). The switch manages the groups using priority-based scheduling. Each queue group performs fair scheduling and independently employs FLOWSAIL as its FC mechanism. We adopted the Web Server as the traffic workload and configured each priority level with a 15% load.

We measured the average FCT slowdown for traffic at each priority level and present the results in Fig. 19. The

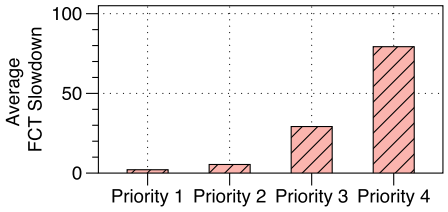



Fig. 19. Average FCT slowdown for each priority level's traffic under Web Server distribution.

results demonstrate that traffic of different priority levels experiences varying FCT slowdowns. Specifically, the highest and lowest priority traffic suffer the lowest and largest slowdowns, respectively. Note that the degradation across different priority levels follows an exponential-like decrease due to the inherent feature of priority-based scheduling. In summary, the results clearly indicate that FLOWSAIL can coexist with priority-based scheduling and maintain varying levels of QoS.

## VII. LIMITATION AND DISCUSSION

**HoL blocking among congested flows.** As previously explained, FLOWSAIL utilizes a single reserved queue for all congested flows at a port and ensures in-order delivery through Order Mark Matching. Consequently, the potential for HoL blocking exists among congested flows, as a congested flow may be required to wait for preceding congested flows to locate a matching Order Mark. However, we contend that this has a negligible impact on overall performance, as evidenced by earlier experiments wherein FLOWSAIL demonstrated superior average FCTs. We believe that a more fine-grained scheme designed specifically for congested flows could eliminate this issue, and we leave this as one of our future works.

**Optimization with statistical counter.** FLOWSAIL necessitates flow size counting, and probabilistic counters, such as sketch [45], hold the potential to significantly alleviate the memory overhead associated with flow size counting. Nonetheless, it is essential to acknowledge that some sketch methods may introduce inaccuracies in flow size counting; this could pose challenges to our existing FLOWSAIL design, given its current reliance on precise flow size counting. Consequently, the integration of sketch methods necessitates non-trivial adaptation, and we leave this as one of our future works.

**Deadlock Prevention.** As FLOWSAIL continues to use the stopping and starting of a queue as its control action, it is not immune to deadlock. Similar to PFC, FLOWSAIL is also vulnerable to deadlocks when there are cyclic buffer dependencies (CBD). There are plenty of solutions for deadlock prevention in PFC scenarios, such as CBD prevention and traffic rerouting [21], [46], [47]. These existing solutions can be applied directly to FLOWSAIL without system redesign.

## VIII. RELATED WORKS

Firstly, numerous studies have attempted to address PFC's limitations and attempted to address its limitations [21], [23], [48]. However, these PFC-based approaches inherit the static mapping feature of PFC. That is, the mapping between traffic and queues is statically configured using constant packet tags. In contrast, FLOWSAIL dynamically maps flows to available queues and implements flow isolation when necessary, providing a more flexible solution. Another approach, TCD [49], aims to tackle the interference between flow control and end-to-end CCs, which does not resolve the feedback delay issue inherent in end-to-end transports. Although we primarily discuss PFC in this paper, it is worth noting that CBFC [18] suffers from similar coarse-grained granularity issues.

Secondly, proactive CCs, such as Homa [29], NDP [50], Aeolus [14], and ExpressPass [51], enable receivers to allocate credits to senders in advance of data transmission. However, proactive CCs struggle either with wasting the first RTT or risking congestion reoccurrence. In contrast, per-hop FC utilizes switches to enable accurate and timely control decisions.

Finally, some studies propose various queue scheduling methods to mitigate the negative impact of large flows on the latency of short flows. Examples include fair queuing approximation [52] and priority-based scheduling [42]. Switch buffer managements, such as ABM [16], leverage different switch buffer characteristics to determine the admission threshold for each queue. These methods alone cannot effectively reduce buffer occupancy and are orthogonal to FLOWSAIL.

## IX. CONCLUSION

In this paper, we introduce FLOWSAIL, an innovative flow control scheme that enables fine-grained control at the per-flow level without the requirement of per-flow queues. The core of FLOWSAIL is to effectively approximate the ideal FC's behavior at both the congested port and upstream port through its key design components. Extensive prototype experiments and simulations demonstrate the superior performance of FLOWSAIL compared with the state-of-the-art FC scheme, BFC, and end-to-end CCs.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive suggestions.

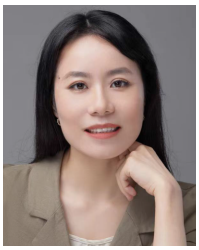
## REFERENCES

[1] M. Li et al., "Communication efficient distributed machine learning with the parameter server," in *Proc. NIPS*, vol. 27, 2014.
[2] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," *IEEE Trans. Cloud Comput.*, 2022.
[3] Z. Wang et al., "SRNIC: A scalable architecture for RDMANICs," in *Proc. USENIX NSDI*, 2023, pp. 1–14.
[4] W. Li et al., "Gleam: An RDMA-accelerated multicast protocol for datacenter networks," 2023, *arXiv:2307.14074*.
[5] W. Bai et al., "Information-agnostic flow scheduling for commodity data centers," in *Proc. USENIX NSDI*, 2015, pp. 455–468.
[6] *Google Cloud Platform*. [Online]. Available: https://cloud.google.com
[7] J. Davidson et al., "The YouTube video recommendation system," in *Proc. 4th ACM Conf. Recommender Syst.*, Barcelona, Spain, Sep. 2010, pp. 293–296.
[8] W. Li et al., "Cepheus: Accelerating datacenter applications with high-performance roce-capable multicast," in *Proc. IEEE HPCA*, Aug. 2024, pp. 908–921.
[9] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, Aug. 2010, pp. 63–74.
[10] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, 2015.

[11] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 514–528.
[12] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM SIGCOMM*, 2019, pp. 44–58.
[13] P. Goyal et al., "Backpressure flow control," in *Proc. USENIX NSDI*, 2022, pp. 779–805.
[14] S. Hu et al., "Aeolus: A building block for proactive transport in datacenters," in *Proc. ACM SIGCOMM*, 2020, pp. 422–434.
[15] S. Abdous, E. Sharafzadeh, and S. Ghorbani, "Burst-tolerant datacenter networks with vertigo," in *Proc. 17th Int. Conf. Emerg. Netw. exp. Technol.*, 2021, p. 1–15.
[16] V. Addanki et al., "ABM: Active buffer management in datacenters," in *Proc. ACM SIGCOMM*, 2022, pp. 36–52.
[17] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: Saving (DC)TCP for high-speed extremely shallow-buffered datacenters," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 2007–2016.
[18] *Infiniband Architecture Volume 1, General Specifications, Release 1.2.1*. [Online]. Available: http://www.infinibandta.org/specs
[19] *Supplement To InfiniBand Architecture Specification Volume 1 Release 1.2.2 Annex A17: RoCEv2 (IP Routable RoCE)*. [Online]. Available: http://www.infinibandta.org/specs
[20] *IEEE 802.1 Qbb Priority-based Flow Control*. [Online]. Available: https://1.ieee802.org/dcb/802-1qbb/
[21] S. Hu et al., "Tagger: Practical PFC deadlock prevention in data center networks," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2017, pp. 451–463.
[22] J. Hu et al., "Load balancing in PFC-enabled datacenter networks," in *Proc. ACM APNet*, 2022.
[23] C. Tian et al., "P-PFC: Reducing tail latency with predictive PFC in lossless data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1447–1459, Jun. 2020.
[24] W. Li, C. Zeng, J. Hu, and K. Chen, "Scaling switch-driven flow control with aquarius," in *Proc. 7th Asia–Pacific Workshop Netw.*, Jun. 2023, pp. 81–87.
[25] N. Kung and R. Morris, "Credit-based flow control for ATM networks," *IEEE Netw.*, vol. 9, no. 2, pp. 40–48, Jul. 1995.
[26] W. Li et al., "Towards fine-grained and practical flow control for datacenter networks," in *Proc. IEEE ICNP*, Jul. 2023.
[27] *Edge-Core Networks*. [Online]. Available: https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335
[28] A. Roy et al., "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, 2015, pp. 123–137.
[29] B. Montazeri et al., "HOMA: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM*, 2018, pp. 221–235.
[30] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster, and T. Edsall, "RoCC: Robust congestion control for RDMA," in *Proc. 16th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2020, pp. 17–30.
[31] R. Mittal et al., "Timely: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.
[32] *NS-3*. [Online]. Available: https://www.nsnam.org/
[33] W. Bai, K. Chen, S. Hu, K. Tan, and Y. Xiong, "Congestion control for high-speed extremely shallow-buffered datacenter networks," in *Proc. 1st Asia–Pacific Workshop Netw.*, Aug. 2017, pp. 29–35.
[34] *High-Density 25/100 Gigabit Ethernet StrataXGS Tomahawk Ethernet Switch Series*. [Online]. Available: https://www.cavium.com/
[35] Y. Pan et al., "Support ECN in multi-queue datacenter networks via per-port marking with selective blindness," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 33–42.
[36] Z. Yu et al., "Programmable packet scheduling with a single queue," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 179–193.
[37] Y. Lu et al., "One more queue is enough: Minimizing flow completion time with explicit priority notification," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.
[38] *Cavium XPliant*. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56960
[39] *Intel Tofino 2*. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.htm
[40] W. Bai et al., "Empowering Azure storage with RDMA," in *Proc. USENIX NSDI*, 2023, pp. 49–67.
[41] P. E. McKenney, "Stochastic fairness queueing," in *Proc. IEEE INFOCOM*, Jun. 1990, pp. 733–734.
[42] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.
[43] J. Sonchack et al., "TurboFlow: Information rich flow record generation on commodity switches," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–16.
[44] D. Zhou et al., "Scalable, high performance Ethernet forwarding with cuckooswitch," in *Proc. ACM CoNEXT*, 2013, pp. 97–108.
[45] C. Hu et al., "DISCO: Memory efficient and accurate flow statistics for network measurement," in *Proc. IEEE ICDCS*, Feb. 2010, pp. 665–674.
[46] K. Qian et al., "Gentle flow control: Avoiding deadlock in lossless networks," in *Proc. ACM SIGCOMM*, 2019, pp. 75–89.
[47] C. Guo et al., "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 202–215.
[48] W. Cheng et al., "Re-architecting congestion management in lossless Ethernet," in *Proc. USENIX NSDI*, 2020, pp. 19–36.
[49] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 370–383.
[50] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2017, pp. 29–42.
[51] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2017, pp. 239–252.
[52] N. K. Sharma et al., "Approximating fair queueing on reconfigurable switches," in *Proc. USENIX NSDI*, 2018, pp. 1–16.
[53] K. Xu et al., "TACC: A full-stack cloud computing infrastructure for machine learning tasks," 2021, *arXiv:2110.01556*.

**Wenxue Li** (Student Member, IEEE) received the B.S. degree from Zhejiang University in 2021. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, advised by Prof. Kai Chen. Her research interests include datacenter networking, in-network acceleration, and machine learning systems.

**Chaoliang Zeng** received the B.S. degree from the University of Science and Technology of China in 2018 and the Ph.D. degree from The Hong Kong University of Science and Technology (HKUST) in 2023, supervised by Prof. Kai Chen. His research interests are datacenter networking, hardware acceleration, and machine learning systems.



**Jinbin Hu** (Member, IEEE) received the Ph.D. degree in computer science from Central South University, China, in 2020. She is currently a Post-Doctoral Researcher with the Computer Science and Engineering Department, The Hong Kong University of Science and Technology, advised by Prof. Kai Chen. Her current research interests are in the area of datacenter networks and distributed systems.

**Kai Chen** (Senior Member, IEEE) received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is currently a Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong. His current research interests include data center networking, AI-centric networking, and machine learning systems.