Efficient Headroom Allocation With Two-Level Flow Control for Lossless Datacenter Networks

Danfeng Shan, Jinchao Ma, Yunguang Li, Boxuan Hu, Tong Zhang, Yazhe Tang, Hao Li, Jinyu Wang, and Peng Zhang

Abstract—**In datacenters, lossless network is very attractive as it can achieve ultra-low latency. In commodity Ethernet, lossless forwarding is achieved by hop-by-hop Priority-based Flow Control (PFC). To avoid buffer overflow, PFC-enabled switches need to reserve some buffer as *headroom*, absorbing in-flight packets during the delay for backpressure messages to take effect. However, with the growing link speed in production networks, the buffer becomes increasingly insufficient, and the headroom can occupy a considerable fraction of buffer. As a result, the remaining buffer for absorbing normal traffic bursts is significantly squeezed, leading to frequent PFC messages that degrade the network performance. Worse yet, we find that the current static and queue-independent headroom allocation scheme is quite inefficient, resulting in significant buffer wastage. In light of this, we propose Dynamic and Shared Headroom allocation scheme (DSH), which dynamically allocates headroom to congested queues and enables sharing of allocated headroom among different queues. To achieve this, DSH first introduces port-level flow control, which performs flow control at the granularity of individual ports, guaranteeing lossless forwarding with a small fraction of per-port headroom. With this lossless guarantee, the switch is liberated for dynamic headroom adjustment. DSH dynamically allocates per-queue headroom based on the congestion status of each queue. Meanwhile, DSH preserves the queue-level flow control to protect the non-congested queues from being paused by congested queues, ensuring performance isolation on buffer sharing. Extensive experiments show that DSH can reduce the flow completion time by up to ∼78.8%.**

Index Terms—**Priority-based flow control, bursty traffic, buffer management.**

Received 23 September 2024; revised 10 April 2025; accepted 30 July 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor J. S. Sun. This work was supported in part by the National Natural Science Foundation of China under Grant 62372363, Grant 62472219, Grant 62132007, Grant 62172323, and Grant 62272382; in part by the National Key Laboratory of Advanced Communication Networks Fund under Project SCX24641X001; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20242038; and in part by China Post-Doctoral Science Foundation under Grant 2024T171163. The preliminary version of this paper was published in the Proceedings of the 2023 IEEE ICDCS [DOI: 10.1109/ICDCS57875.2023.00019]. *(Corresponding author: Tong Zhang.)*

Danfeng Shan, Jinchao Ma, Yunguang Li, Boxuan Hu, Yazhe Tang, Hao Li, Jinyu Wang, and Peng Zhang are with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: dfshan@xjtu.edu.cn; majc2002@stu.xjtu.edu.cn; yunguangli1@outlook.com; huboxuan2004@gmail.com; yztang@xjtu.edu.cn; hao.li@xjtu.edu.cn; jinyu.wang@xjtu.edu.cn; p-zhang@xjtu.edu.cn).

Tong Zhang is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 211106, China (e-mail: zhangt@nuaa.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TON.2025.3596437, provided by the authors.

Digital Object Identifier 10.1109/TON.2025.3596437

I. Introduction

LOSSLESS network is increasingly attractive in datacenters as it can provide ultra-low latency for applications [2], [3], [4], [5], [6]. In commodity Ethernet, lossless transmission is achieved by the hop-by-hop Priority-based Flow Control (PFC) mechanism. To avoid packet dropping, a PFC-enabled switch sends a PAUSE frame to its upstream device when its buffer is about to overflow. Upon receiving the PAUSE frame, the upstream device holds back the packet transmission. To prevent buffer overflow, a fraction of buffer should be reserved as *headroom* to absorb in-flight packets before the PAUSE frame takes effect. However, in large-scale networks, PFC messages can result in serious performance issues, such as head-of-line blocking, congestion spreading, collateral damage, and even deadlocks [3], [5], [7], [8], [9], [10], [11], [12], [13],. Therefore, it is a common belief that PFC should be triggered as infrequently as possible. Ideally, PFC should only serve as a backup method to ensure lossless packet transmissions.

However, due to the recent industrial trends, it is more and more likely that datacenter networks (DCNs) suffer from frequent PFC messages. Specifically, the link speed of DCN has rapidly grown from 1Gbps to 40Gbps/100Gbps and will continuously increase to 400Gbps in the near future [3], [14]. The amount of required headroom is increasingly large as it is positively related to the link speed. Unfortunately, the memory size in the switching chip cannot keep pace with the increasing link capacity [15], [16], [17]. This is because datacenter switches usually employ on-chip memory for high-speed and low-latency access, and the memory size is limited by the chip area and cost. Specifically, the buffer size (related to switching capacity) has decreased by 4× over the past decade (§III-A). Under these trends, a considerable amount of buffer should be reserved as headroom, significantly squeezing the buffer space for accommodating normal traffic. As a result, the queue length can easily hit the PFC pause threshold, leading to frequent generation of PFC messages. Although recent advances in end-to-end congestion control (CC) algorithms [2], [3], [4], [5] can help keep persistent buffer occupancy low, they cannot completely tackle this issue. This is because end-to-end CC takes at least 1 round-trip time (RTT) to respond to traffic changes, unable to control over short-term congestion events, which are very common in DCNs. Specifically, studies have shown that most flows will be finished within 1 RTT in future DCNs [18], [19] and most congestion events will be caused by sub-RTT traffic bursts [20]. Within 1 RTT, it is the buffer management scheme that determines whether PFC messages can be avoided.

Given the considerable headroom requirements, we find that the current headroom allocation scheme, referred to as Static and Independent Headroom (SIH) allocation in this paper, is quite inefficient in headroom utilization. Our experiments show that 75% of headroom remains unused 99% of the time even when the network load reaches 90% (§III-B). We analyze the root causes of this inefficiency through experiments, and find that the underlying reason lies in SIH's *static* and *queue-independent* nature. Specifically, SIH reserves a *fixed* amount of headroom *independently* for each ingress queue on every port. However, this approach is quite inefficient due to the following two reasons.

(1) The actual required headroom is dynamic to the traffic characteristics. The required headroom is determined by the traffic arrival and departure rates at ingress queues, which vary over time. Despite this, SIH allocates a fixed amount of headroom for each queue. To avoid buffer overflow under any circumstances, the static manner entails SIH allocating headroom based on the worst-case scenario, which, however, rarely occurs, resulting in significant wasted headroom buffer most of the time.

(2) The ingress queues on the same port naturally share the uplink capacity. The traffic arriving rate to an ingress queue is less than the link capacity as long as other ingress queues on the same port are active. However, SIH fails to take advantage of this sharing property. Instead, it reserves headroom buffer independently for each ingress queue, resulting in wasted headroom buffer.

Furthermore, our experiment results demonstrate that SIH's inefficiency cannot be fundamentally addressed by simply reducing the reserved headroom size (§III-C). This is because the worst case, although rarely, does occur. As a result, current approaches, although enhancing headroom sharing by oversubscribing the headroom buffer [21], [22], can result in the risk of packet loss, which is unacceptable for RoCE transport with Go-back-N loss recovery. Consequently, a brand-new headroom allocation scheme is needed to radically resolve this issue.

In light of these observations, we propose *Dynamic and Shared Headroom* allocation scheme (DSH) (§IV) to improve the headroom efficiency without risking packet loss. Our key idea is that the headroom allocation is closely correlated to the flow control. The current PFC mechanism performs flow control independently for each queue, requiring the headroom to be statically reserved per queue independently. To enable headroom sharing, the flow control should be performed across different queues jointly. Based on this idea, DSH incorporates *port-level* flow control besides *queue-level* flow control. The *port-level* flow control performs port-wise flow control operations, which enables DSH to guarantee lossless forwarding with a minimal amount of reserved headroom. It is based on the observation that different ingress queues in the same port naturally share the common uplink capacity. Thus, to avoid packet overflow, there is no need to independently reserve headroom for each ingress queue. Instead, DSH only needs to reserve headroom for each port and make the ingress queues on the same port share the reserved headroom. If any queue at a port starts to occupy the headroom, the queue generates a port-wise flow control message to pause the entire upstream port. This, however, sacrifices the isolation across different queues. Thus, DSH retains the queue-level flow control and makes port-level flow control merely as a backup measure against packet loss.

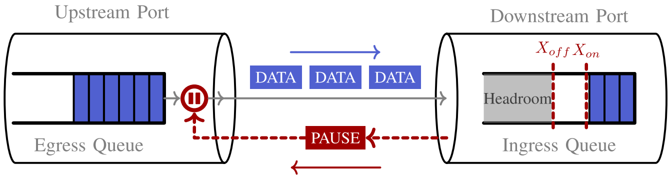


Fig. 1. Hop-by-hop priority-based flow control.

The *queue-level* flow control performs queue-wise flow control actions, similar to PFC. It is motivated by the observation that not all queues require headroom simultaneously. Therefore, DSH *dynamically* allocates headroom to a queue only when it becomes congested. This ensures that headroom is provisioned only when necessary, preventing wastes. Furthermore, the amount of headroom allocated is dynamically adjusted based on traffic characteristics, aligning with the actual demands of congested queues and reducing buffer waste. Additionally, DSH allows headroom to be *shared* among multiple ingress queues, leveraging statistical multiplexing to improve buffer utilization efficiency.

We evaluate DSH with extensive ns-3 simulations (§V). The microbenchmarks show that DSH can effectively mitigate the impairments (including collateral damage and deadlock) induced by PFC, while guaranteeing performance isolation among different traffic classes (§V-A). Our large-scale simulations show that DSH reduces the flow completion time by up to ∼78.8% for short fan-in flows and up to ∼45.2% for other flows (§V-B).

The rest of the paper is organized as follows. §II introduces the background of PFC and switch buffer. §III discusses the problem of the current headroom allocation scheme. Next, §IV describes the design of DSH. In §V, we present the evaluations of DSH. Finally, we discuss some related work in §VI and conclude in §VII.

II. Background

In this section, we introduce the background of PFC and switch buffer.

A. Priority-Based Flow Control

Ethernet-based datacenter networks rely on Priority-based Flow Control (PFC) [23] to guarantee lossless packet forwarding. PFC is a hop-by-hop flow control mechanism (as shown in Fig. 1). In a PFC-enabled switch, once the length of an ingress queue exceeds a preset threshold (i.e., X_{off}), the switch sends a PAUSE frame to the upstream device. Upon receiving the PAUSE frame, the upstream device suspends the packet transmission for the duration specified by the PAUSE frame. When the ingress queue length falls below another threshold (i.e., X_{on}), the device sends a PAUSE frame with zero duration (which we refer to as *RESUME frame*) to the upstream device, which recovers the packet transmission.

To prevent packet dropping, X_{off} should be set conservatively. This is because it takes time for the PAUSE frame to arrive at the upstream device and take effect. To prevent buffer overflow, enough buffer beyond X_{off} should be reserved to accommodate in-flight packets during this time. The reserved buffer beyond X_{off} is called *buffer headroom*.

In the PFC standard [23], traffic can be classified into 8 priority classes. Each priority class is mapped to a separate queue, with packets of different priority classes placed into their respective queues. The PFC messages carry the priority information and can pause/resume one or more traffic classes.

B. Buffer Architecture in PFC-Enabled Switches

On a switching chip, the packets waiting to be transmitted are stored in a packet buffer. Today's commodity high-speed switching chips usually employ on-chip shared memory for high-bandwidth and low-latency packet access [3], [14], [24], [25], [26], [27], [28], [30], [31]. To improve buffer efficiency, the scarce memory is dynamically shared among all queues. Shared-memory switches typically employ output-queueing, where packets are placed into queues dedicated to their egress ports (i.e., egress queues). Output-queued switches do not contain physical ingress queues. To support PFC, which triggers PAUSE/RESUME messages based on ingress queue lengths, switches record the length of each *logical* ingress queue with a counter [3]. Furthermore, in modern switch architectures, shared buffer allocation for lossless traffic is managed from the ingress perspective, i.e., the switch allocates buffer space to ingress queues. When the length of an ingress queue exceeds its allocated buffer, incoming packets are not admitted into the corresponding buffer and must be paused.

As shown in Fig. 2, the buffer from ingress perspective is partitioned into three segments [28], [32], [33].

- **Private pool:** buffer space reserved for each queue, which guarantees each queue's minimum buffer resource.
- **Shared pool:** buffer space shared among all queues.
- **Headroom pool:** buffer space reserved for each queue, which absorbs in-flight packets after sending PAUSE frames.

In addition to lossless traffic, lossy traffic may also be present in datacenter networks [34]. To isolate these two traffic types, the buffer is typically partitioned into two separate pools: a lossless pool and a lossy pool. Unlike lossless traffic, whose buffer is ingress-allocated, the buffer for lossy traffic is managed from the egress perspective. Specifically, packets are dropped when the egress queue length exceeds the allocated buffer capacity.

C. Buffer Allocation From Ingress Perspective

The switching chip utilizes a Memory Management Unit (MMU) to allocate the buffer to arriving packets. The sizes of the private pool and headroom pool are explicitly configured. The remaining buffer serves as shared buffer.

There is no explicit rule specifying how to configure the private pool size. Nevertheless, the amount of private pool is relatively small (e.g., 16% in Arista 7050 × 3 switches [29]).

Different from the private pool, the size of the headroom pool should be carefully configured to prevent packet loss. This is because it takes some delay for a PAUSE frame to take effect, and the MMU needs to reserve enough headroom to absorb the arriving traffic during this delay. According to [2], [3], [35], and [36], the headroom size for each ingress queue (denoted by ϕ) is given by

$$\Phi = 2\left(C \cdot D_{prop} + L_{MTU}\right) + 3840\text{B} \tag{1}$$

where C is the capacity of the upstream link, D_{prop} is the propagation delay of the upstream link, and L_{MTU} is the length of an MTU-sized packet. The rationale of such a setting is as follows. The delay for PFC pause to take effect comprises the following five parts:

① Waiting delay: A port may be busy transmitting another packet when a PAUSE frame is generated. The PAUSE frame needs to wait for the transmission to be finished. In the worst case, the port just begins to transmit the first bit of an MTU-sized packet, and thus the PAUSE frame needs to wait for L_{MTU}/C time.

② Propagation delay (of PAUSE frame): It takes D_{prop} time for the PAUSE frame to arrive at the upstream device. D_{prop} depends on the cable length and propagation speed of signals. In datacenters, the distance between two connected switches can be as large as 300 meters [3]. For single-mode fibers, the speed of light is 65% of that in a vacuum. As a result, the propagation delay is ~1.5μs.

③ Processing delay: It takes some time for the switch to process the PAUSE frame and stop the transmission. The PFC definition has capped this time to 3840B/C [35].

④ Response delay: When the upstream port decides to execute the pause action, it might be sending another packet. In the worst case, the switch just begins to transmit the first bit of an MTU-sized packet. Thus, it takes L_{MTU}/C for the pause action to truly take effect.

⑤ Propagation delay (of the last packet): When the upstream device stops sending packets, there are still some in-flight packets on the link, which should also be absorbed by the headroom. It takes another D_{prop} time for the last sent packet to arrive at the downstream switch.

Combining the above five parts results in Eq. 1.

The shared buffer is available to all queues. MMU utilizes a buffer management scheme to ensure fair and efficient allocation of the shared buffer among all queues. Among various buffer management schemes, Dynamic Threshold (DT) is the most common one on commodity switching chips [3], [5], [24], [25], [26], [29], [32], [37], [38], [39],.

DT uses a threshold to restrict the shared buffer occupancy of each queue. The threshold is dynamically adjusted according to the remaining buffer size. Specifically, let $T(t)$ denote the threshold at time t, $\omega_s^{i,j}(t)$ denote the amount of shared buffer occupation for queue j in port i at time t, and B_s denote the shared buffer size. The threshold is given by

$$T(t) = \alpha \cdot \left(B_s - \sum_{i=1}^{N_p} \sum_{j=1}^{N_q} \omega_s^{i,j}(t) \right) \quad (2)$$

where α is a control parameter, N_p denotes the number of ports, and N_q denotes the number of queues per port. The intuition behind DT is as follows. When the network is less congested, the buffer occupancy is low and the remaining buffer size is high. DT adjusts the threshold to a higher value, which allows each queue to occupy more buffer, making the buffer efficiently used. On the contrary, when the network is more congested, the buffer occupancy is high and thus the remaining buffer size is low. DT adjusts the threshold to a lower value, which restricts the buffer usage of each queue, ensuring fair sharing of the buffer among different queues.

Algorithm 1 Packet Placement

```
if $\omega_p^{i,j}(t) + packet.length < \rho$ then
    ▷ Place the packet into the private buffer pool ◁
    $\omega_p^{i,j}(t) \leftarrow \omega_p^{i,j}(t) + packet.length$
else if $\omega_s^{i,j}(t) + packet.length < T(t)$ then
    ▷ Place the packet into the shared buffer pool ◁
    $\omega_s^{i,j}(t) \leftarrow \omega_s^{i,j}(t) + packet.length$
else if $\omega_h^{i,j}(t) + packet.length < \Phi$ then
    ▷ Place the packet into the headroom buffer pool ◁
    $\omega_h^{i,j}(t) \leftarrow \omega_h^{i,j}(t) + packet.length$
else
    ▷ Drop the packet ◁
```

With PFC enabled, MMU monitors the ingress queue lengths and decides where to place each arriving packet. The overall workflow is outlined in Algorithm 1, and the used notations are summarized in TABLE I. Generally, MMU places packet in the order of private pool, shared pool, and headroom pool, selecting the first pool with sufficient free space.

Besides, MMU generates PFC PAUSE messages to upstream devices based on the amount of shared buffer occupancy (i.e., $\omega_s^{i,j}(t)$) and X_{off}/X_{on} thresholds. With DT, the X_{off}/X_{on} thresholds are dynamic, namely $X_{off} = T(t)$ and $X_{on} = X_{off} - \delta$. Each ingress queue transitions between two states: ON and OFF state, as illustrated in Fig. 3. In the ON state, the ingress queue turns to the OFF state when the occupancy of shared pool becomes higher than X_{off}. Meanwhile, a PFC PAUSE frame is generated to the upstream device. In the OFF state, the ingress queue turns into the ON state when the headroom pool is empty and the occupancy of shared pool drops below X_{on}. Meanwhile, a PFC RESUME frame is generated to the upstream device.

III. Motivation

In this section, we present the problem of the current headroom allocation scheme.

A. Headroom Occupies Considerable Memory

It is expected that most of the memory should serve as shared buffer to absorb bursty traffic without triggering PFC messages. However, with the current buffer allocation scheme, the headroom buffer occupies considerable memory, which can significantly squeeze "footroom" buffer[1] and result in frequent PFC messages.

TABLE I
Key Notions

Notion	Description
$T(t)$	Threshold of shared buffer occupancy
α	Parameter of DT
B_s	Size of shared pool
B_p	Size of private pool
Φ	Required headroom size per ingress queue
B_h	Total headroom size with SIH
ρ	Size of private buffer per ingress queue
(i,j)	Ingress queue j at port i
$\omega_p^{i,j}(t)$	Occupancy of private pool for queue (i,j)
$\omega_s^{i,j}(t)$	Occupancy of shared pool for queue (i,j)
$\omega_h^{i,j}(t)$	Occupancy of headroom pool for queue (i,j)
N_p	Number of ports in the switch
N_q	Number of queues per port
B_i	Size of insurance headroom with DSH

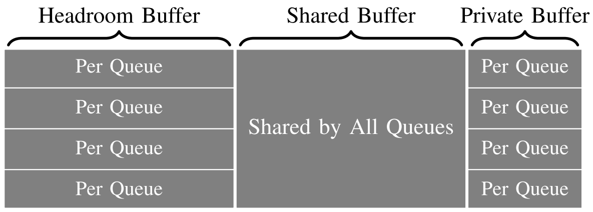


Fig. 2. Buffer partition in a PFC-enabled switch.

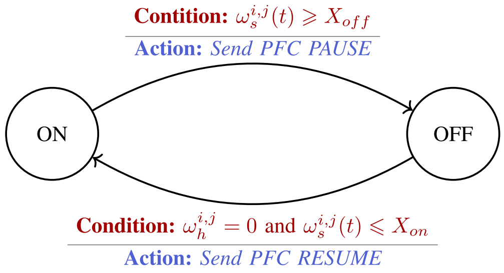

Contition: $\omega_s^{i,j}(t) \geqslant X_{off}$
Action: Send PFC PAUSE
ON
OFF
Condition: $\omega_h^{i,j} = 0$ and $\omega_s^{i,j}(t) \leqslant X_{on}$
Action: Send PFC RESUME

Fig. 3. State transition diagram of PFC.

Specifically, the current buffer allocation scheme *independently* reserves a *static* headroom for every ingress queue [2], [3]. Assume that each ingress queue requires Φ headroom. The total headroom size (denoted by B_h) is given by

$$B_h = N_p \times N_q \times \Phi \quad (3)$$

where N_p is the number of ingress ports, N_q is the number of queues per port, and Φ is given by Eq. 1.

With this method, MMU has to allocate worst-case headroom for every ingress queue, and *headroom can occupy a large fraction of memory*. For example, Broadcom Trident2 switching chip contains 12MB memory. It has 32 40GbE ports (i.e., $N_p = 32$ and $C = 40$Gbps). For each port, the PFC

[1] For convenience, we define footroom as the buffer other than headroom.

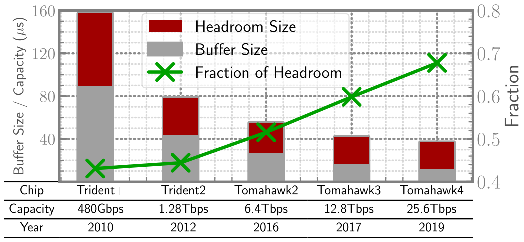


Fig. 4. Trends of buffer in Broadcom's switching chips.

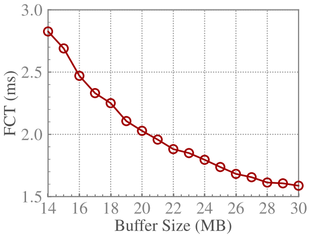


Fig. 5. FCT vs. buffer size.

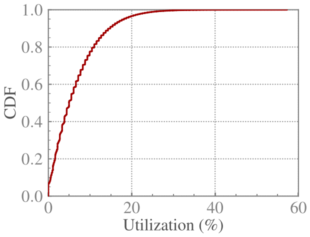


Fig. 6. Headroom utilization.

standard supports 8 queues (i.e., $N_q = 8$). Assume that the MTU is 1500B (i.e., $L_{MTU} = 1500\text{B}$) and $D_{prop} = 1.5\mu\text{s}$, MMU needs to allocate ∼5.33MB memory for headroom buffer in total, which occupies 44.4% of total memory.

With the growing link capacity, this situation gets worse. The link speed in production DCN has grown from 1Gbps to 40Gbps and 100Gbps in the past decade [5], [14] and continues to grow. With higher link speed, MMU needs to allocate more headroom to avoid buffer overflow. However, the buffer size is limited by the chip area and cost, and thus cannot scale with the switching capacity [15], [16], [17]. As a result, *the fraction of required headroom becomes increasingly large*, significantly squeezing the footroom buffer. Fig. 4 depicts the trend of buffer size and the fraction of required headroom in Broadcom's switching chips, where we assume the presence of eight traffic classes, as specified in IEEE 802.1Qbb [23], with all classes enabled.

The switch buffer size per unit of capacity has decreased by 4× in the last decade (from 157μs to 37μs), while the fraction of required headroom has increased by 56% (from 43% to 67%).

Without enough "footroom" buffer, PFC messages can be frequently triggered, which may result in serious performance impairments (e.g., head-of-line blocking, congestion spreading, collateral damage) and even network deadlocks.

To quantitatively demonstrate the performance degradation brought by inadequate buffer, we perform a large-scale ns-3 simulation. We build a 16 × 16 leaf-spine topology with 256 servers. Each link has 100Gbps capacity and 2μs latency. Each switch is equipped with a 16MB buffer, emulating the Broadcom Tomahawk switch chip [40] (more details in §V-B). The headroom size is configured according to Eq. (1) and (3). The congestion control algorithm is PowerTCP [41], which can effectively keep persistent queue length low. We use the widely-used web search workload [30] to generate realistic DCN traffic. The total network load is 90%. Fig. 5 shows the average flow completion time (FCT) with different buffer sizes. The FCT with 14MB buffer is 78.1% worse than that with 30MB buffer.

To alleviate this problem, current network operators have to restrict the number of priority queues [3]. However, this ad hoc approach can aggravate the head-of-line blocking problem, as different services cannot be isolated and the congestion of one point can spread to the entire network. Furthermore, lots of studies [18], [42], [43], [44], [45] have shown that multiple service queues can greatly improve the network performance. Restricting the number of queues prevents the network applications from benefiting from them.

B. Current Headroom Allocation Scheme Is Inefficient

Despite the increasingly large fraction of headroom, the current *static and independent* headroom allocation scheme (referred to as SIH) is still quite inefficient and wasteful. To quantitatively illustrate this issue, we conduct a simulation with the same settings as before, except that the congestion control algorithm is DCQCN [2], which induces a higher buffer occupancy. To examine the headroom efficiency, we extract the local maximum values of headroom occupancy, which indicates the actual required headroom size. Fig. 6 shows that the headroom utilization is only 4.96% at the median and 25.33% at the 99th percentile, indicating that the headroom buffer is significantly over-allocated most of the time.

Next, we analyze the underlying reason behind SIH's inefficiency. To ensure lossless forwarding, SIH allocates the headroom buffer based on the wost-case scenario, which has two assumptions:

- **Assumption 1.** All ingress queues need to occupy the headroom simultaneously.
- **Assumption 2.** During the time for the PAUSE frame to take effect, the length of each ingress queue grows at line rate, i.e., the ingress queue is continuously receiving traffic at line rate without being drained.

However, we find that the above two assumptions are not valid most of the time due to the following three observations.

Observation 1: *Not all queues need to occupy headroom.* An ingress queue needs to occupy headroom only when it gets congested and its queue length exceeds the X_{off} threshold. In reality, it is unlikely that all queues are congested at the same time [21], i.e., Assumption 1 is typically not valid. Despite this, SIH *independently* allocates worst-case headroom size for all ingress queues. As a result, most headroom buffer keeps unused. Fig. 7 shows the distribution of the number

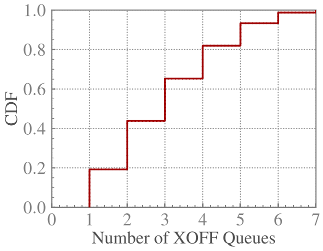


Fig. 7. Number of queues in XOFF state.

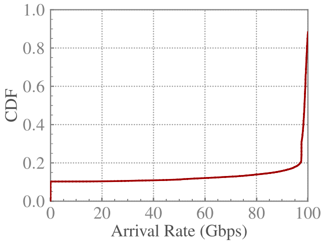


Fig. 8. Traffic arrival rate to the XOFF queue (1 CoS, no sharing).

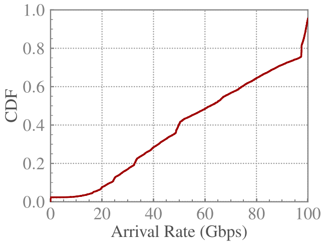


Fig. 9. Traffic arrival rate to the XOFF queue (7 CoSes, sharing).

of queues in the OFF state when the headroom buffer is being occupied (the experiment settings are the same as above). We can observe that, in over 60% cases, fewer than half of the queues require occupying the headroom, indicating that over 50% of the headroom is over-allocated more often than not.

Observation 2: *Traffic heading for different ingress queues at the same port shares the uplink capacity.* In a port, all ingress queues are connected to the same uplink and thus the traffic heading for them shares the link capacity. When a traffic class of a port needs to be paused, its traffic arriving rate should be lower than C as long as other traffic classes also have in-flight packets on the uplink. In this case, Assumption 2 is not valid. Fig. 8 and Fig. 9 show the distribution of the traffic arrival rates to an XOFF queue with flows classified into 1 queue and 7 queues, respectively. With a single queue exclusively using the uplink capacity (Fig. 8), the traffic arrival rate can approach the line rate most of the time. In comparison, with multiple queues sharing the uplink capacity, the traffic arrival rate to an XOFF queue drops to $\sim$60Gbps at the median (note that the line rate is 100Gbps).

Observation3: *The headroom buffer is being drained during the OFF state.* When determining the headroom requirement (§II-C), it is assumed that the traffic arrives at line rate (denoted by C), causing the headroom occupancy to increase at a rate of C. However, actually, the increasing rate of headroom occupancy is also related to the traffic departure rate. Assuming that the traffic arrival rate towards an ingress queue is C and the traffic departure rate is d, then the actual increasing rate of headroom occupancy is $C - d$. In reality, $d > 0$ unless *all* egress queues holding the traffic are being paused. As a result, the buffer occupancy of the headroom pool may grow at a rate lower than C, and Assumption 2 is not valid. Fig. 10 shows that the traffic departure rate is over 20Gbps at the median, indicating that the required headroom is over-allocated in half of the cases.

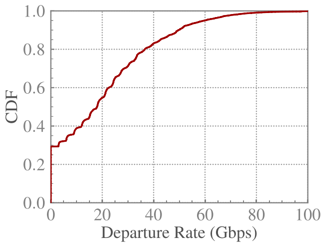


Fig. 10. Traffic departure rate from the XOFF queue.

In sum, *the underlying reason behind SIH's inefficiency lies in its static and queue-independent nature.* Specifically, the actual required headroom is dynamic to the traffic characteristics. The static nature entails SIH allocating the worst-case headroom to avoid buffer overflow. The worst-case, however, rarely occurs. Furthermore, SIH independently allocates headroom for all ingress queues, regardless of the fact that the ingress queues at a port naturally share the uplink capacity.

C. Why Not Simply Reduce the Headroom Size

Given the inefficiency of SIH, a natural approach is to oversubscribe the headroom size [21], [22], i.e., reserving less headroom than the required amount B_h. However, this approach, although improving the headroom efficiency, can bring about the risk of packet loss. As shown in Fig. 7, Fig. 9, and Fig. 10, the worst case, although rarely, indeed occurs occasionally. In these cases, the oversubscribed headroom is insufficient to avoid buffer overflow, leading to packet loss, which is unacceptable for the RoCE transport, because packet losses can significantly hurt the transmission performance due to the "Go-back-N" strategy. In sum, *the inefficiency of SIH is inherent to its static and queue-independent nature*, and cannot be resolved by simply adjusting the headroom buffer size. Thus, we choose to explore another headroom allocation scheme.

IV. DSH Design

To address the inefficiency problem of SIH, we propose *Dynamic and Shared Headroom (DSH)* allocation, which aims to efficiently allocate headroom while ensuring no congestion loss. In this section, we first explain the key ideas behind DSH in §IV-A. Then we present the details of our design in §IV-B, §IV-C, §IV-D, and §IV-E. Finally, we discuss some issues of DSH on various scenarios in §IV-F.

A. Key Ideas

DSH utilizes two ideas to achieve efficient headroom allocation while ensuring no congestion loss.

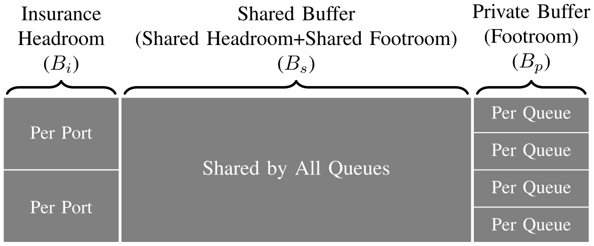

Insurance Headroom (B_i)
Shared Buffer (Shared Headroom+Shared Footroom) (B_s)
Private Buffer (Footroom) (B_p)
Per Port
Per Port
Shared by All Queues
Per Queue
Per Queue
Per Queue
Per Queue

Fig. 11. Buffer partition with DSH.

(1) DSH proactively reserves a small amount of buffer as insurance headroom to avoid packet loss. Based on Observation 2, different ingress queues in the same port share the uplink capacity. Thus, to avoid buffer overflow under any circumstances, there is no need to allocate Φ headroom for each ingress queue. Rather, we only need to reserve Φ buffer for each port. In this way, the amount of reserved headroom is significantly reduced. However, the benefit comes at the cost of performance isolation. When any ingress queue in a port starts to occupy the insurance buffer, the entire upstream port is paused. This compromises performance isolation among different traffic classes. Thus, we also have the following mechanism to make the port-wise pause rarely triggered, only serving as an insurance measure.

(2) DSH dynamically allocates the headroom to congested queues and makes the allocated headroom shared among different ingress queues. Based on Observation 1, an ingress queue needs to occupy headroom only when it is congested. Therefore, instead of reserving headroom for uncongested queues, DSH allocates headroom only when a queue becomes congested.

Furthermore, Observations 2 and 3 suggest that the actual headroom requirement varies with traffic characteristics. Therefore, instead of allocating a fixed amount of headroom for a congested queue, DSH dynamically adjusts the headroom allocation based on observed traffic characteristics. In addition, since the allocated headroom may not always be fully utilized, DSH allows the headroom buffer to be shared across ingress queues. In this way, DSH can take advantage of statistical multiplexing to improve the buffer efficiency.

B. Buffer Organization

Fig. 11 shows the buffer organization with DSH. In addition to the traditional buffer partitions, DSH further divides the headroom into two parts: shared headroom and insurance headroom. The insurance headroom is statically reserved for each port to guarantee against buffer overflow. The shared headroom is dynamically allocated as needed and shared among different ingress queues.

Furthermore, as both shared headroom and shared footroom are dynamically shared and allocated, DSH integrates them into a single segment, collectively called shared buffer. Such a design brings two benefits: (1) It facilitates the implementation of DSH on switching chips, as the buffer partition is the same as the existing one on commodity switching chips. (2) It improves the buffer utilization. By integrating two kinds of buffer, both headroom and footroom share the same piece of buffer, increasing the degree of statistical multiplexing. As a result, the buffer is utilized more efficiently.

C. Estimation of the Actual Required Headroom Per Queue

Based on Observations 2 and 3, the required headroom for each ingress queue depends on traffic characteristics. Instead of allocating the worst-case headroom (i.e., Φ), DSH estimates headroom requirement based on traffic arrival and departure rates. Furthermore, to avoid port-level pauses caused by insufficient headroom, DSH employs conservative headroom estimation.

Specifically, the actual required headroom per ingress queue can be given by

$$\phi = \int_0^D g(t)\,\mathrm{d}t \tag{4}$$

where $g(t) = \frac{\mathrm{d}}{\mathrm{d}t}q(t)$ is the gradient of queue length, and D is the delay for the pause message to take effect.

Algorithm 2 Estimating ϕ

```
function UPON PACKET ARRIVAL
    $g \leftarrow \frac{\Delta q}{\Delta t}$                                ▷ Queue length gradient
    $v \leftarrow |g_{avg} - g|$                             ▷ Deviation
    $g_{avg} \leftarrow (1 - w_g) \cdot g_{avg} + w_g \cdot g$         ▷ Average gradient
    $v_{avg} \leftarrow (1 - w_v) \cdot v_{avg} + w_v \cdot v$         ▷ Mean deviation
    $\hat{g} \leftarrow g_{avg} + k \cdot v_{avg}$          ▷ Conservatively estimated gradient
    $\hat{\phi} \leftarrow \hat{g} \times D$
```

As the headroom size must be determined beforehand, DSH predicts $g(t)$ and ϕ using a conservative algorithm outlined in Algorithm 2, which favors overestimation over underestimation to prevent port-wise pause. First, the instantaneous queue length gradient $g = \frac{\Delta q}{\Delta t}$ is calculated upon every packet arrival. To mitigate noise effects, we apply Exponential Weighted Moving Average (EWMA) to smooth the gradient. The average queue length gradient (g_{avg}) is updated upon each packet arrival as

$$g_{avg} = (1 - w_g) \cdot g_{avg} + w_g \cdot g \tag{5}$$

where w_g weights the instantaneous gradient against the historical average.

To prevent underestimation of the queue length gradient, DSH incorporates variance using an approach similar to Retransmission Timeout (RTO) maintenance [46]. On every packet arrival, the mean deviation v_{avg} is updated as

$$v_{avg} = (1 - w_v) \cdot v_{avg} + w_v \cdot |g_{avg} - g| \tag{6}$$

Considering the variation, DSH estimates the queue length gradient as

$$\hat{g} = g_{avg} + k \cdot v_{avg} \tag{7}$$

where k controls sensitivity to variations.

After estimating the queue length gradient $\hat{g}$, the headroom is then given by

$$\hat{\phi} = \hat{g} \times D \tag{8}$$

The conceptual design involves computationally intensive operations (e.g., division, multiplication, floating-point arithmetic), which are infeasible for high-speed switches like

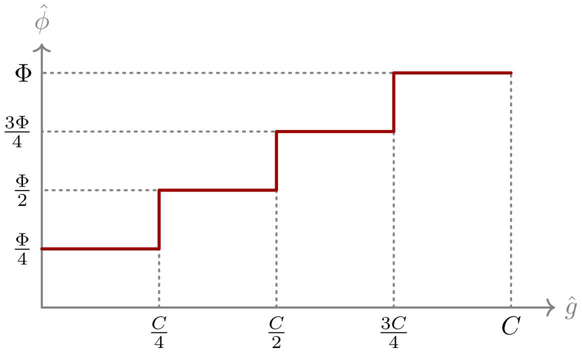


Fig. 12. Mapping function from queue length gradient to $\hat{\phi}$.

Tofino [47] due to nanosecond-level packet processing constraints. To facilitate implementation, DSH employs several techniques to minimize computational overhead.

Algorithm 3 Estimating ϕ in Practice

```
function On Packet Arrival
    $g \leftarrow 2^{\log_2(\Delta q) - \log_2(\Delta t)}$
    $v \leftarrow |g_{avg} - g|$
    $g_{avg} \leftarrow g_{avg} - w_g \cdot v$
    $v_{avg} \leftarrow v_{avg} - w_v \cdot (v_{avg} - v)$
    $\hat{g} \leftarrow g_{avg} + k \cdot v_{avg}$
    Compare $\hat{g}$ to thresholds gets $\hat{\phi}$
```

To eliminate division operations, DSH implements division using logarithm and exponent functions [48], [49]. As demonstrated in Algorithm 3, $\frac{\Delta q}{\Delta t}$ can be reformulated as $2^{\log_2(\Delta q) - \log_2(\Delta t)}$. The logarithm and exponent can be implemented using lookup table [48].

To avoid multiplication, DSH configures the parameters w_g, w_v, and k as powers of two, allowing multiplication to be replaced with bit shifting operations. However, this method cannot be applied to the computation of Eq. 8, as neither $\hat{g}$ nor D necessarily equals a power of two. To avoid multiplication, DSH utilizes a mapping function to convert $\hat{g}$ to $\hat{\phi}$, as illustrated in Fig. 12. This mapping function can be implemented as a lookup table with pre-calculated values, thereby eliminating real-time multiplication computations.

Overall, Algorithm 3 presents the estimation of the required headroom in practice.

D. Buffer Allocation and Management

The allocation of the private buffer remains unchanged. For insurance headroom, DSH statically reserves some memory for each ingress port. Assume that there are N_p ports, the insurance headroom size (denoted by B_i) is given by

$$B_i = N_p \times \Phi \tag{9}$$

where Φ is given by Eq. 1.

The remaining memory serves as shared buffer. DSH adopts DT to dynamically allocate shared buffer for each ingress queue. This is because DT has been widely used in commodity switching chips for decades, proven to be adaptive and efficient while simple to be implemented. Specifically, DSH uses a threshold $T(t)$ to restrict the buffer occupancy (including both shared headroom and shared footroom) for each ingress queue. The threshold $T(t)$ is dynamically adjusted based on the remaining shared buffer, given by Eq. 2. Note that the amount of shared buffer occupancy (i.e., $\omega_s^{i,j}(t)$) in Eq.2 includes the buffer occupancy of both shared headroom and shared footroom with DSH.

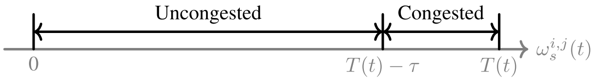


Fig. 13. An ingress queue can be considered as congested when the buffer occupancy approaches $T(t)$.

After allocating the total buffer space for both shared headroom and shared footroom, the next step is to determine the portion of the buffer designated for shared headroom. The key idea is to allocate $\hat{\phi}$ headroom to an ingress queue only when it gets congested. This is implemented using a simple mechanism based on the observation that, *for a congested queue, its shared buffer occupancy — represented as $\omega_s^{i,j}(t)$ — should approach its buffer allocation $T(t)$*. In contrast, for a non-congested queue, $\omega_s^{i,j}(t)$ remains significantly below $T(t)$, as illustrated in Fig. 13. Therefore, DSH allocates headroom to a queue only if its queue length exceeds $T(t) - \tau$, where τ denotes the margin to the buffer allocation. This allows real-time congestion detection using a comparator and a subtractor, which are both cheap and fast.

Next, we analyze how to set τ. On the one hand, τ should not be too large, otherwise the headroom is allocated to a queue prematurely when it is not heavily congested and unlikely to occupy headroom in the near future. On the other hand, τ should not be too small, otherwise DSH cannot allocate enough shared headroom from the free buffer. Based on the above two considerations, we find that $\hat{\phi}$ is an appropriate choice for τ. $\hat{\phi}$ is not large and is an estimation of the required headroom for an ingress queue. DSH can likely allocate $\hat{\phi}$ headroom for the queue when the buffer occupancy reaches $T(t) - \hat{\phi}$.

Algorithm 4 Setting τ on Packet Arrival

```
if $packet.queue\_id \neq prev\_queue\_id$ then
    $start\_time \leftarrow current\_time$
    $prev\_queue\_id \leftarrow packet.queue\_id$
    $\tau \leftarrow \hat{\phi}$
else if $current\_time - start\_time > window\_length$ then
    $\tau \leftarrow 0$
```

There is still an issue that DSH may waste up to 50% of the headroom with only one queue per port has traversing traffic. In this case, only one queue per port requires headroom, while DSH may allocate $\Phi + \hat{\phi}$ headroom per port (i.e., Φ insurance headroom + $\hat{\phi}$ shared headroom). To further improve the efficiency, DSH set τ to 0 in this special case. This adjustment can be achieved through a simple algorithm shown in Algorithm 4. The basic idea is that, if all packets are heading for the same queue within a time window, DSH sets $\tau = 0$. The length of the time window is 10ms.

E. Flow Control

We now demonstrate how flow control works with the above buffer allocation scheme. DSH has two types of flow control mechanisms to guarantee against packet loss while ensuring performance isolation: queue-level flow control and port-level flow control.

The *queue-level flow control* is similar to the PFC mechanism. Specifically, a PFC PAUSE frame is sent to the upstream port when the amount of shared buffer occupancy for an ingress queue becomes higher than the pause threshold (denoted by X_{qoff}). Arriving at the upstream port, the PFC PAUSE frame will suspend the packet transmission of the corresponding traffic class. Furthermore, when the amount of shared buffer occupancy for an ingress queue falls below a resume threshold (denoted by X_{qon}), a PFC RESUME frame is sent to the upstream port to recover the packet transmission of the corresponding traffic class.

The only difference is the setting of X_{qoff} threshold. With DSH, the $X_{qoff}(t)$ threshold is set as

$$X_{qoff}(t) = T(t) - \tau \tag{10}$$

where $\tau = \hat{\phi}$ with multiple queues active in each port and $\tau = 0$ otherwise. The settings of resume threshold is the same as PFC. Specifically, the resume threshold X_{qon} is slightly lower than the pause threshold, namely, $X_{qon} = X_{qoff} - \delta_q$. For example, according to [2], δ_q can be set to 2 MTUs.

Only queue-level flow control is not enough to avoid buffer overflow, as the shared headroom is dynamically allocated as needed rather than statically reserved beforehand, and thus DSH cannot guarantee that every ingress queue can get $\hat{\phi}$ headroom when becoming congested. Consequently, DSH also contains port-level flow control to avoid buffer overflow under any circumstances.

The *port-level flow control* is triggered when the total occupancy of shared footroom and headroom of *all* queues in a port becomes higher than a pause threshold (denoted by X_{poff}). When triggered, the ingress port sends a port-level PAUSE frame to the upstream port. Arriving at the upstream port, the port-level PAUSE frame will suspend the packet transmissions of *all* traffic classes. Furthermore, when the total occupancy of shared footroom and headroom in a port falls below a resume threshold (denoted by X_{pon}), a port-level RESUME frame is sent to the upstream port to cancel the suspension.

The $X_{poff}(t)$ threshold is set as

$$X_{poff}(t) = N_q \times T(t) \tag{11}$$

Similar to the queue-level flow control, the resume threshold X_{pon} is slightly lower than the pause threshold, namely, $X_{pon} = X_{poff} - \delta_p$, where δ_p can be set to 2 MTUs based on [2].

The intuition of the above equation is as follows. DSH allocates $T(t)$ buffer as footroom and headroom for each ingress queue. Thus, for all ingress queues in a port, the total allocated buffer is $N_q \times T(t)$. The rationale behind the intuition is that DSH allows the ingress queues in the same port to share the allocated buffer, especially headroom. Specifically, by restricting the port-level buffer occupancy (rather than queue-level), a congested queue can occupy the headroom allocated to other queues (in the same port) if it has used up its allocated headroom. As the traffic heading for the ingress queues in the same port naturally shares the capacity of uplink, port-level buffer share is both efficient and fair. In this way, not only can DSH utilize the shared buffer more efficiently, but also the port-level flow control can be less triggered.

F. Discussions

The port-level flow control can be implemented using PFC, without introducing new flow control messages. The PFC frame contains both priority enable and time vectors, each comprising eight fields that support pausing and resuming eight priorities independently. Port-level PAUSE/RESUME messages can be realized by setting all fields in both priority and time vectors accordingly. Furthermore, to achieve effective port-level flow control, we must establish conditions for generating these port-level PAUSE/RESUME messages. The pause threshold (X_{poff}) is calculated as $N_q \cdot T(t)$, where N_q represents the number of queues per port. This calculation requires a multiplier with N_q and $T(t)$ as inputs. Notably, as is often the case, when N_q is a power of two, the calculation can be simplified using only a shift register. The resume threshold (X_{pon}) is defined as $X_{pon} - \delta_p$, where δ_p is a configurable parameter. Computing this threshold requires a subtractor with X_{pon} and δ_p as inputs.

In the scenarios where lossless RDMA traffic coexists with legacy lossy TCP traffic, competition for buffer resources may arise. When both traffic types require buffer space, TCP traffic could potentially impact RDMA performance. Nevertheless, this interference occurs in the footroom rather than the headroom [34], [50]. Since DSH specifically targets headroom allocation for lossless traffic, addressing this interference falls outside DSH's scope.

Certain congestion control algorithms, such as DCQCN [2], require sufficient footroom buffer to maintain high link utilization [51]. By improving headroom efficiency, DSH enables more footroom space for congestion control mechanisms, thereby facilitating higher throughput performance.

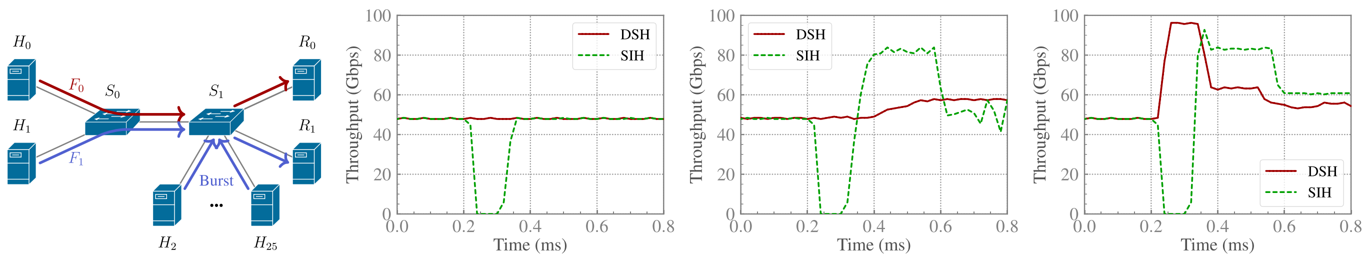
DSH's performance benefits increase with the number of traffic classes (i.e., more queues per port). In practical deployments with fewer enabled traffic classes, DSH's advantages may be somewhat diminished. Nevertheless, DSH's efficiency is never worse than SIH. In the extreme case where only one queue is enabled, DSH cannot dynamically share headroom across queues. In this scenario, the headroom allocated by DSH is Φ per port, equivalent to SIH.

V. Evaluation

In this section, we evaluate DSH's performance with both testbed experiments and ns-3 simulations [52].

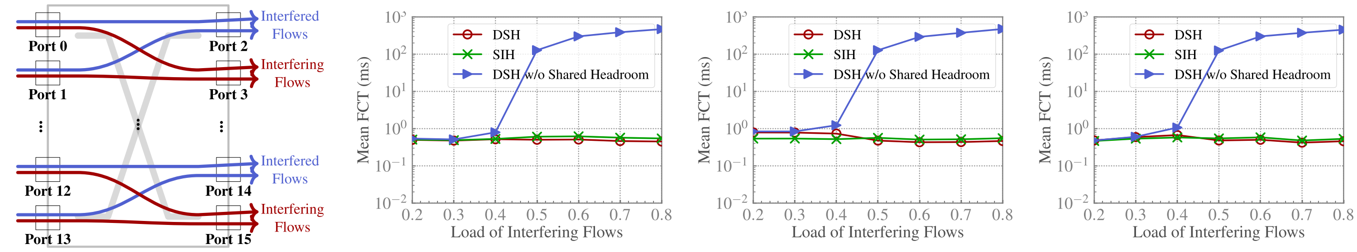
A. Microbenchmarks

In this part, we evaluate DSH's basic performance with carefully constructed microbenchmarks based on ns-3 simulations. We emulate the Broadcom Tomahawk switching chip,



(a) Scenario (b) Throughput of F_0 (w/o CC) (c) Throughput of F_0 (DCQCN) (d) Throughput of F_0 (PowerTCP)

Fig. 14. [Simulation] Mitigation of collateral damage.



(a) Scenario (b) Mean FCT (w/o CC) (c) Mean FCT (DCQCN) (d) Mean FCT (PowerTCP)

Fig. 15. [Simulation] Performance isolation between different queues at the same port.

which has 32 100Gbps ports and 16MB shared memory. Each port has 8 queues. One queue is reserved for ACK packets and PAUSE/RESUME messages, assigned to the highest priority. Other seven queues are scheduled by the DWRR algorithm with a quantum of 1600B. The link delay is 2μs and thus $\Phi = 60000$B. The total headroom size for SIH is $60000\text{B} \times 32 \times 7 = 12.8$MB. The private buffer size is 672KB (3KB for each DWRR-scheduled queue). For DT, α is set to 1/16 according to [3]. For DSH, we set $w_g = w_v = 0.25$ and $k = 4$. The X_{qon}/X_{pon} threshold is the same as the X_{qoff}/X_{poff} threshold.

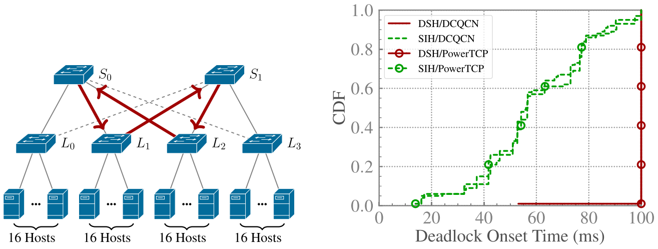
In this part, we evaluate DSH's ability to mitigate collateral damage with ns-3 simulations. We consider the same scenario as above, which is shown in Fig. 14(a). All links are 100Gbps and the propagation delay is 2μs. Two long-lived flows, F_0 and F_1, are sending traffic from H_0 and H_1 to R_0 and R_1, respectively. After their throughputs reach 50Gbps, H_2-H_{25} generate 24 concurrent fan-in flows to R_1. Each flow has a size of 64KB, which is smaller than a BDP and thus the fan-in traffic is uncontrolled by congestion control algorithms.

Fig. 14 shows the throughput of F_0. We can observe similar results as testbed experiments that DSH can effectively avoid performance degradation for F_0. Furthermore, we find that the state-of-the-art congestion control algorithms (Fig. 14(c) and Fig. 14(d)) are not able to avoid the collateral damage. This is because end-to-end congestion control requires at least 1 RTT to react to traffic changes. Within 1 RTT, it is the buffer management scheme that determines whether PFC messages can be avoided.

DSH incorporates port-level flow control. As a result, a single congested queue can potentially pause the entire port. In this part, we evaluate the performance isolation of DSH among different queues at the same port. We consider a scenario shown in Fig. 15(a). A switch is connected to 16 hosts with 100Gbps/2μs links. We divide these hosts and the connected switch ports into 4 equivalent groups. In each group, we generate two kinds of flows: interfered flows and interfering flows. We evaluate the impact of interfering flows on interfered flows. Take group 1 as an example, the interfered flows are from Port 0/Port 1 to Port 2. Flows are generated one by one and flow arrivals follow a Poisson process with an average load of 0.2. Different from interfered flows, we generate two interfering flows simultaneously, transmitting 64KB data from Port 0 and Port 1 to Port 3, respectively. As a result, the interfering flows certainly induce buffer occupancy. The load of interfering flows varies from 0.2 to 0.8 in order to demonstrate their impact on interfered flows. These two kinds of flows are classified into different traffic classes, with interfered flows classified into Class of Service (CoS) 1 and interfering flows randomly classified into CoS 2-7. Other settings remain unchanged. Ideally, the performance of interfered flows should not be affected by the interfering flows.

Fig. 15 shows the mean FCT of the interfered flows without congestion control (Fig. 15(b)), with DCQCN (Fig. 15(c)), and with PowerTCP (Fig. 15(d)). To show the effectiveness of shared headroom in improving performance isolation, we compare DSH's performance with SIH and DSH without shared headroom. We observe that the performance of DSH is comparable to that of SIH, with the FCTs of interfered flows unaffected by the interfering flows. Furthermore, without shared headroom, DSH's performance is significantly degraded when the load of interfering flows becomes higher than 0.5. These observations validate that the shared headroom can effectively ensure performance isolation among different queues at the same port.

One impairment brought by PFC messages is deadlock [3], [7], [8], [9], [12], which is a serious problem as it can make



(a) Scenario

(b) CDF of deadlock onset time

Fig. 16. [Simulaiton] Deadlock avoidance.

a large part of the network unusable. In this part, we evaluate the ability of DSH to avoid deadlock.

We consider a topology shown in Fig. 16(a), which is a leaf-spine topology with two link failures marked with dashed lines (i.e., S_0-L_3 and S_1-L_0). In the topology, there are two spine switches (S_0 and S_1) and four leaf switches ($L_0 - L_3$). Each leaf switch is connected to 16 hosts with 100Gbps downlinks, and connected to two spine switches with 400Gbps uplinks. The link delay is 2μs. We generate fan-in flows, which are from L_0 to L_3, from L_3 to L_0, from L_1 to L_2, and from L_2 to L_1, respectively. As a result, there is a cyclic buffer dependency marked as red lines: $S_0 \rightarrow L_1 \rightarrow S_1 \rightarrow L_2 \rightarrow S_0$. The fan-in ratio (i.e., the number of flows) ranges from 1 to 15. The flow size is randomly chosen based on the Hadoop workload [31], and flow arrivals follow a Poisson process. The network load is 0.5 at the downlinks of each leaf switch. Each scheme is tested 100 times and each simulation lasts for 100ms.

Fig. 16(b) shows the CDF of deadlock onset time. With SIH, deadlock occurs for all simulations either with DCQCN [2] or PowerTCP [41]. In comparison, DSH can avoid 96% of the deadlocks with DCQCN and all deadlocks with PowerTCP. This is because DSH can leave more "footroom" to absorb bursty traffic, avoiding PFC messages.

B. Large-Scale Simulations

In this part, we evaluate DSH in a large-scale DCN topology. The simulations are conducted on a Huawei Taishan 200 server, equipped with two Kunpeng 920 CPUs and 192GB of memory.

We build a leaf-spine topology with 16 leaf switches, 16 spine switches, and 256 servers. Each leaf switch is connected to 16 servers with 100Gbps downlinks and 16 spine switches with 100Gbps uplinks, forming a full-bisection network. The link delay is 2μs and thus the base RTT is 16μs across the spine. We employ ECMP for load balancing.

We emulate the Broadcom Tomahawk switching chip. The settings are the same as those in previous simulations.

We consider three end-to-end congestion control algorithms: DCQCN [2], HPCC [5], and PowerTCP [41]. We use the default parameter settings in their open-source simulations.

We generate two kinds of traffic: background traffic and bursty fan-in traffic. The background traffic follows a one-to-one pattern. The sender and receiver are randomly chosen. The flow sizes are generated according to a web search workload [30]. The fan-in traffic follows a many-to-one pattern, where 16 senders simultaneously transmit 64KB data to the same receiver. The senders are randomly chosen and are in different racks from the receiver. Flow arrivals follow a Poisson process. The total network load is 0.9. The flows are randomly classified into CoS 1-7. CoS 0 is reserved for ACK packets and PFC PAUSE/RESUME frames.

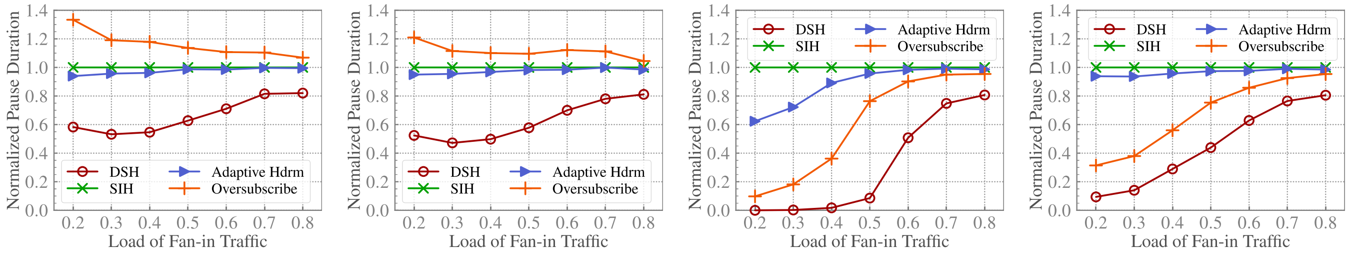
We compare DSH with three alternative headroom allocation schemes. (1) SIH: The de facto headroom allocation scheme used in practice, where each queue is allocated a fixed amount of headroom [2], [3], namely, $\Phi = 60,000$B. The total headroom pool size is 12.8MB. (2) Adaptive PFC Headroom: A mechanism proposed in IEEE P802.1Qdt [53] that automatically determines the headroom required for each queue through precise link delay measurements [54]. (3) Headroom Oversubscription: An approach adopted by Microsoft [21] and SONiC [22] to improve headroom efficiency by allocating less headroom than SIH. In our evaluation, we configure an oversubscription ratio of 80%.

Fig. 17 shows the total pause duration with different congestion control algorithms. For clear comparisons, we normalize each pause duration to the value achieved by SIH. With more footroom, DSH can significantly reduce the PFC messages, especially with less fan-in traffic. Specifically, without congestion control, DSH can reduce the total pause duration by ~18.0%-46.8%. With DCQCN, DSH can reduce the total pause duration by ~18.8%-52.8%. With HPCC and PowerTCP, DSH can more effectively reduce the pause duration, i.e., reducing the total pause duration by ~19.3%-99.9% and ~19.4%-90.6%, respectively. This is because HPCC and PowerTCP is more effective in keeping the persistent queue length low. As a result, the PFC messages are mainly triggered by bursty fan-in traffic whose flow size are within 1 BDP, and the transmission is not controlled by end-to-end congestion control algorithms. In such cases, buffer management plays a more important role in reducing PFC messages.

Adaptive PFC headroom marginally reduces PFC pause duration by accurately determining Φ. However, as analyzed in §III-B, headroom inefficiency primarily stems from the static and queue-independent nature of SIH. Thus, the improvements achieved by adaptive PFC headroom are limited.

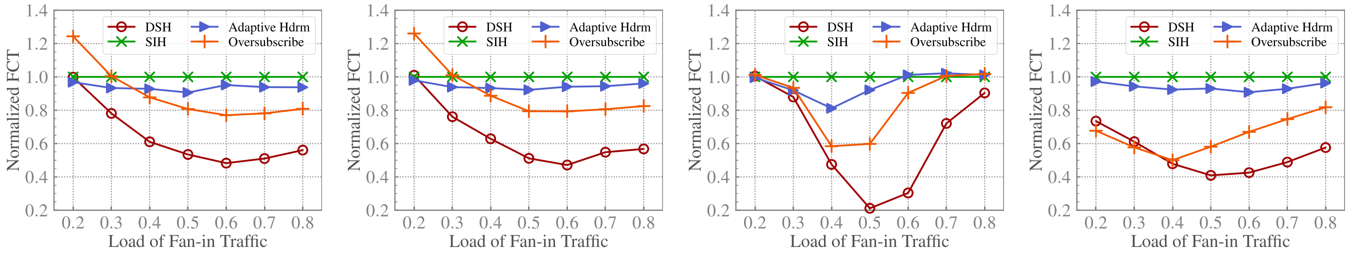
Headroom oversubscription can even deteriorate the PFC pause duration without congestion control or with DCQCN. The reason is that, unlike HPCC and PowerTCP that maintain low persistent buffer occupancy, non-congestion-control or DCQCN can introduce high buffer occupancy from both bursty short flows and long flows. In these scenarios, insufficient headroom due to oversubscription leads to packet drops and subsequent retransmissions, introducing additional traffic that exacerbates congestion. Consequently, PFC pause duration increases. For instance, we find that, with a background traffic load of 0.7 and fan-in traffic load of 0.2, the overall traffic volume of headroom oversubscription is ~29% larger than that of SIH with DCQCN.

Fig. 18 and Fig. 19 show the normalized flow completion time (FCT) of fan-in traffic and background traffic, respectively. The results show that both fan-in flows can benefit



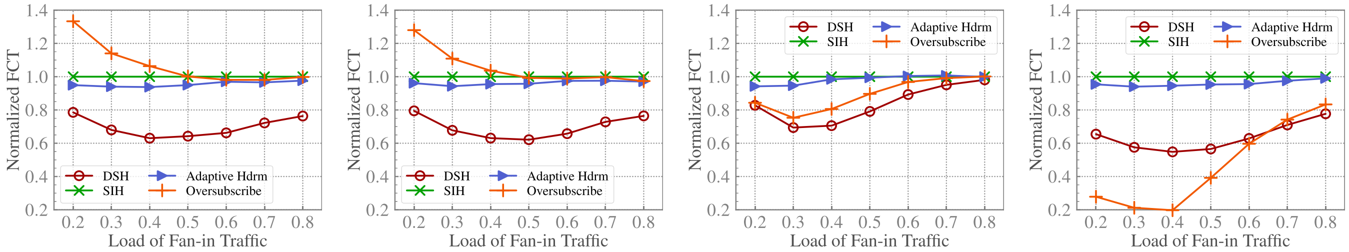
(a) Without congestion control (b) DCQCN (c) HPCC (d) PowerTCP

Fig. 17. Total pause duration in large-scale simulations.



(a) Without congestion control (b) DCQCN (c) HPCC (d) PowerTCP

Fig. 18. Average FCTs of fan-in traffic in large-scale simulations.



(a) Without congestion control (b) DCQCN (c) HPCC (d) PowerTCP

Fig. 19. Average FCTs of background traffic in large-scale simulations.

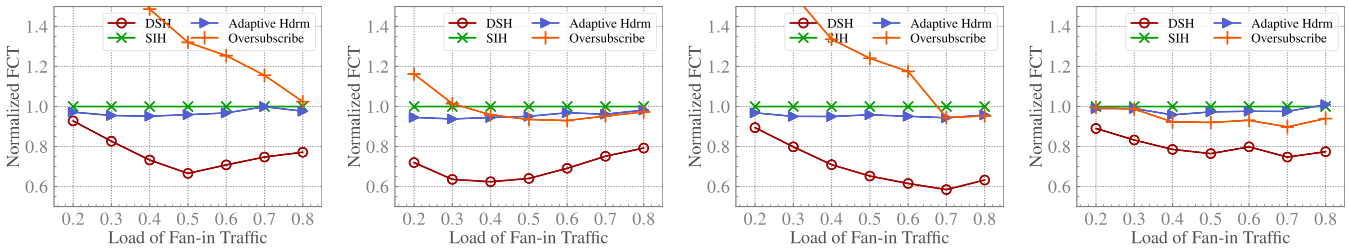
from DSH. Without congestion control, DSH can reduce the average FCTs of fan-in traffic and background traffic by up to ∼51.7% and ∼36.9%, respectively. With DCQCN, DSH can reduce the average FCTs of fan-in traffic and background traffic by up to ∼52.9% and ∼37.9%, respectively. With HPCC, DSH can reduce the average FCTs of fan-in traffic and background traffic by up to ∼78.8% and ∼30.6%, respectively. With PowerTCP, DSH can reduce the average FCTs of fan-in traffic and background traffic by up to ∼59.1% and ∼45.2%, respectively. This is because DSH can provide more footroom to absorb transient bursty traffic and avoid PFC messages.

In comparison, adaptive PFC headroom yields only modest improvements in flow completion time (FCT). Specifically, with DCQCN, FCT improvements for fan-in traffic and background traffic are within ∼8% and ∼6%, respectively. This modest improvement occurs because adaptive PFC headroom only focuses on determining the worst-case headroom requirement, without addressing the fundamental inefficiency of static and queue-independent allocation.

Headroom oversubscription may extend the average FCTs of fan-in and background traffic by up to ∼24.3% and ∼33.3%, respectively. This performance degradation results from packet drops due to insufficient headroom. As analyzed in §III-C, although rare, the worst-case scenario does occur. Consequently, simple oversubscription of the headroom pool can result in packet loss, significantly extending FCT since RoCE NICs typically employ Go-back-N loss recovery. The experiment results substantiate our analysis in §III-C.

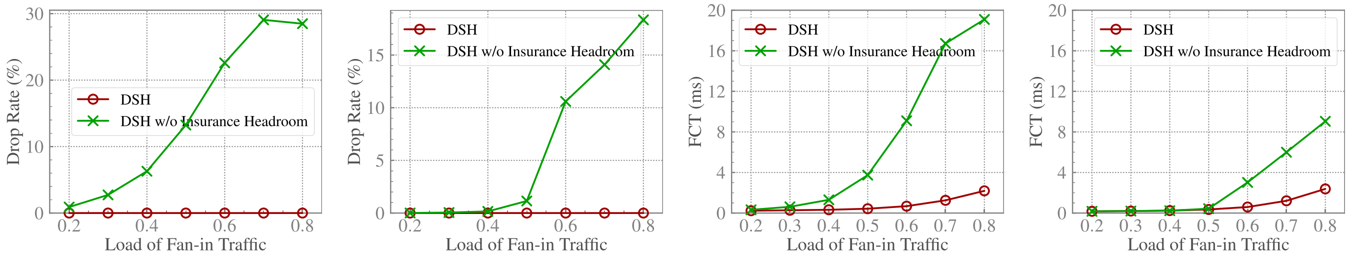
Furthermore, we evaluate DSH across different DCN applications. Besides the web search workload, we also consider other three realistic workloads: a data mining workload [55], a cache workload [31], a Hadoop workload [31], and a storage workload [56]. Other settings remain unchanged. Fig. 20 shows the FCT of background traffic across different workloads with DCQCN. The results confirm that DSH can improve FCT with different DCN workloads.

The insurance headroom is essential to ensure lossless forwarding, because the dynamically allocated shared headroom may be insufficient to accommodate in-flight packets. In this part, we evaluate the effectiveness of insurance headroom. As shown in Fig. 21(a) and Fig. 21(b), without insurance headroom, the packets can be dropped. In comparison, with a small amount of insurance headroom, DSH can ensure lossless



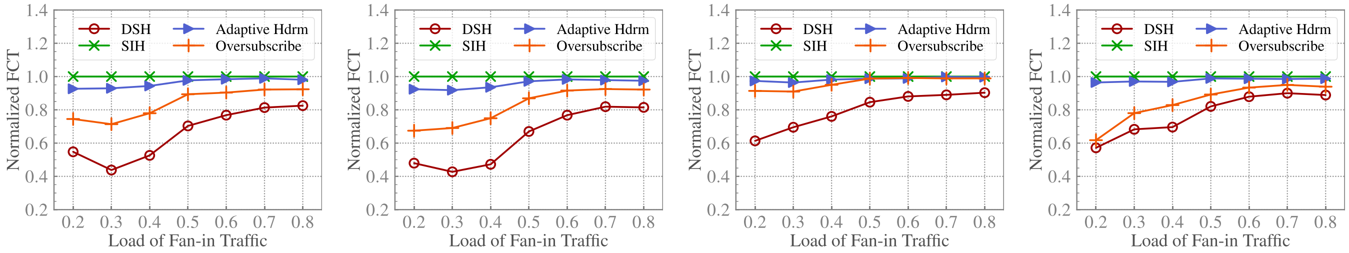
(a) Data mining (b) Cache (c) Hadoop (d) Storage

Fig. 20. Average FCTs across different workloads (transport: DCQCN).



(a) Drop rate with DCQCN (b) Drop rate with PowerTCP (c) Average FCT with DCQCN (d) Average FCT with PowerTCP

Fig. 21. Effectiveness of insurance headroom.



(a) Fan-in Traffic (DCQCN) (b) Fan-in Traffic (PowerTCP) (c) Background Traffic (DCQCN) (d) Backgrnd Traffic (PowerTCP)

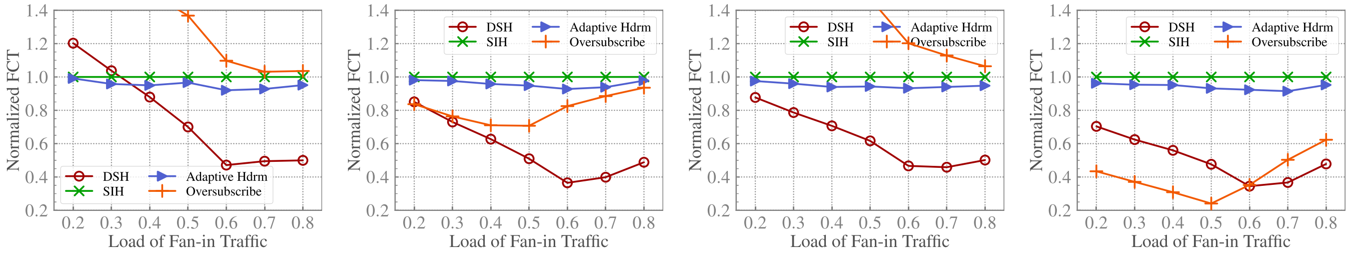
Fig. 22. Average FCTs with high burst traffic.

forwarding under any traffic load. Additionally, Fig. 21(c) and Fig. 21(d) show that the insurance headroom can significantly improve the FCT of background traffic, especially under high traffic load.

To evaluate DSH under high burst conditions, we generate 1024-to-1 fan-in traffic, in which 1024 flows — from 255 senders to a single receiver — are generated at a time. Each flow transmits 8KB data. Other settings remain unchanged. Fig. 22 demonstrates that DSH outperforms all alternative schemes. With DCQCN, DSH improves average FCTs of fan-in and background traffic by up to ∼56.2% and ∼38.6%, respectively. With PowerTCP, DSH improves average FCTs of fan-in and background traffic by up to ∼57.3% and ∼42.8%, respectively.

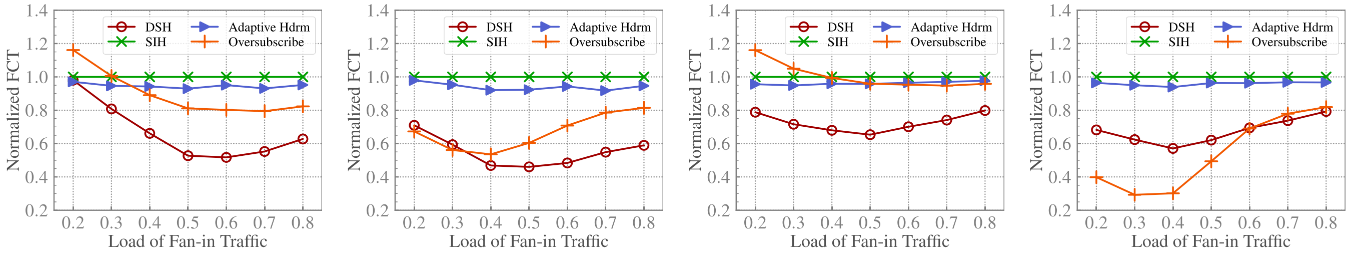
In practical datacenter environments, traffic distribution can be asymmetric due to diverse applications across hosts. To assess DSH under such conditions, we divide hosts into four groups: (1) Group 1: The hosts under leaf switches 0-3, which are generating both background traffic and fan-in traffic. The total traffic load is 0.9, with the load of fan-in traffic varying from 0.2 to 0.8. (2) Group 2: The hosts under leaf switches 4-7, which are generating background traffic with a load of 0.9. (3) Group 3: The hosts under leaf switches 8-11, which are generating background traffic with a load of 0.6. (4) Group 4: The hosts under leaf switches 12-15, which are generating background traffic with a load of 0.3. Other settings remain unchanged. Fig. 23 demonstrates that DSH significantly improves FCTs across most scenarios. With DCQCN, DSH improves average FCTs of fan-in and background traffic by up to ∼52.9% and ∼54.1%, respectively. With PowerTCP, DSH improves average FCTs of fan-in and background traffic by up to ∼63.3% and ∼63.5%, respectively. We also observe that, with DCQCN, DSH can incur worse FCT when the load of fan-in traffic is 0.2. Upon further investigation, we confirmed that this performance degradation stems from elevated buffer occupancy generated by background traffic — an issue that end-to-end congestion control algorithms are expected to mitigate. Since this phenomenon relates to congestion control effectiveness rather than headroom allocation, addressing it falls beyond DSH's scope.

In this part, we consider a scenario that the application can concurrently generate multiple flows. To simulate such applications, we increase the concurrency of background traffic by generating 8 flows at a time. Other settings remain unchanged.



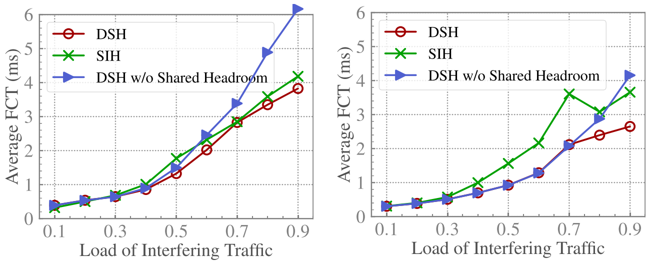
(a) Fan-in Traffic (DCQCN) (b) Fan-in Traffic (PowerTCP) (c) Background Traffic (DCQCN) (d) Backgrnd Traffic (PowerTCP)

Fig. 23. Average FCTs with asymmetric traffic distribution.



(a) Fan-in Traffic (DCQCN) (b) Fan-in Traffic (PowerTCP) (c) Background Traffic (DCQCN) (d) Backgrnd Traffic (PowerTCP)

Fig. 24. Average FCTs with high-concurrency application.



(a) DCQCN (b) PowerTCP

Fig. 25. Performance isolation in large-scale simulations.

Fig. 24 shows that DSH consistently improves the FCT performance for both fan-in and background traffic, regardless of whether DCQCN or PowerTCP is used. Specifically, with DCQCN, DSH can improve the average FCT of fan-in and background traffic by up to ∼48.2% and ∼34.7%, respectively. With PowerTCP, DSH can improve the average FCT of fan-in and background traffic by up to ∼53.9% and ∼42.9%, respectively.

With different queues sharing headroom in DSH, the performance of different traffic classes can be potentially affected. To evaluate how DSH affects performance isolation, we create two kinds of traffic: interfering traffic and interfered traffic. The interfering traffic is randomly classified into CoS 2-7. The interfered traffic is classified into CoS 1. Both kinds of traffic follow a one-to-one pattern, i.e., the source and destination are randomly chosen. Flow arrivals follow a Poisson process and flow sizes are randomly chosen based on the web-search workload [30]. To show the impact of interfering traffic on interfered traffic, we fix the load of interfered traffic at 0.1 and vary the load of interfering traffic from 0.1 to 0.9.

Fig. 25 shows the average FCT of interfered traffic as the load of interfering traffic varies. DSH can achieve comparable performance to SIH, which isolates different queues with dedicated headroom. This result indicates that DSH can effectively ensure performance isolation. Besides, without shared headroom, DSH's performance is significantly degraded under high traffic load, demonstrating the effectiveness of shared headroom in ensuring performance isolation.

VI. Related Work

In recent years, several literatures [34], [50], and [57] point out the issue of isolation between RDMA and TCP traffic when sharing the buffer. L2BM [50] allocates more buffer to ingress queues with higher draining rate, enabling better burst absorption and reducing PFC messages. Reverie [34] consolidates the admission controls of RDMA traffic at ingress and TCP traffic at egress, and applies a low-pass filter to queue lengths in admission control, helping absorb transient traffic bursts. BRT [57] allocates buffer space separately for RDMA and TCP traffic, based on the number of persistent long queues and total dequeue rates. In comparison, DSH only focuses on buffer management for RDMA traffic and is orthogonal to these approaches.

Another line of work focuses on buffer management for footroom or lossy traffic. EDT [58], FAB [59], TDT [60], and Protean [49] allocate more buffer to bursty traffic to enhance burst absorption capability. NDT [61] employs deep reinforcement learning to dynamically adjust the parameters of DT. ABM [62] considers both buffer occupancy and drain rate to achieve isolation, burst tolerance, and bounded buffer drain time. Credence [63] enhances buffer management by incorporating machine-learning-based predictions of future packet arrivals. Occamy [64] achieves agile adjustment of buffer allocation through preemption. SPFC [65] allocates more footroom buffer to victim ports to avoid throughput loss. These buffer management schemes are designed for lossy traffic or footroom allocation, and thus are not applicable to the headroom allocation.

VII. Conclusion

In datacenter networks, PFC-enabled switches need to reserve some buffer as *headroom* to avoid buffer overflow. However, with the growing link speed, the buffer becomes increasingly inadequate, and the headroom occupies a considerable fraction of buffer, significantly squeezing the buffer space for accommodating normal traffic. As a result, PFC messages can be frequently generated, bringing about serious performance impairments. In this paper, we argue that the current static and queue-independent headroom allocation scheme is inherently inefficient. We propose Dynamic and Shared Headroom (DSH) allocation scheme, which dynamically allocates headroom to congested queues and enables allocated headroom to be shared among different queues. Extensive simulations show that DSH can significantly reduce the PFC messages and improve the network performance.

References

[1] D. Shan et al., “Less is more: Dynamic and shared headroom allocation in PFC-enabled datacenter networks,” in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2023, pp. 591–602.

[2] Y. Zhu et al., “Congestion control for large-scale RDMA deployments,” in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 523–536.

[3] C. Guo et al., “RDMA over commodity Ethernet at scale,” in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 202–215.

[4] R. Mittal et al., “TIMELY: RTT-based congestion control for the datacenter,” in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 537–550.

[5] Y. Li et al., “HPCC: High precision congestion control,” in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 44–58.

[6] Z. He et al., “MasQ: RDMA for virtual private cloud,” in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 1–14.

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

[8] S. Hu et al., “Tagger: Practical PFC deadlock prevention in data center networks,” *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 889–902, Apr. 2019.

[9] K. Qian, W. Cheng, T. Zhang, and F. Ren, “Gentle flow control: Avoiding deadlock in lossless networks,” in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 75–89.

[10] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, “Re-architecting congestion management in lossless Ethernet,” in *Proc. USENIX NSDI*, 2020, pp. 19–36.

[11] 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.

[12] X. C. Wu and T. S. Eugene Ng, “Detecting and resolving PFC deadlocks with ITSY entirely in the data plane,” in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 1928–1937.

[13] J. Hu, C. Zeng, Z. Wang, H. Xu, J. Huang, and K. Chen, “Load balancing in PFC-enabled datacenter networks,” in *Proc. 6th Asia–Pacific Workshop Netw.*, Jul. 2022, pp. 21–28.

[14] A. Singh et al., “Jupiter rising: A decade of clos topologies and centralized control in Google’s datacenter network,” in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 183–197.

[15] 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.

[16] G. Zeng, J. Qiu, Y. Yuan, H. Liu, and K. Chen, “FlashPass: Proactive congestion control for shallow-buffered WAN,” in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–12.

[17] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, “Backpressure flow control,” in *Proc. Workshop Buffer Sizing*, Dec. 2019, pp. 1–3.

[18] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, “Homa: A receiver-driven low-latency transport protocol using network priorities,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 221–235.

[19] S. Hu et al., “Aeolus: A building block for proactive transport in datacenters,” in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, Jul. 2020, pp. 422–434.

[20] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, “High-resolution measurement of data center microbursts,” in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 78–85.

[21] W. Bai et al., “Empowering Azure storage with RDMA,” in *Proc. USENIX NSDI*, 2023, pp. 49–67.

[22] *SONiC Command Line Interface Guide*. Accessed: Aug. 17, 2025. [Online]. Available: https://github.com/sonic-net/sonic-utilities/blob/master/doc/Command-Reference.md#dynamic-buffer-management

[23] Priority-Based Flow Control, Standard 802.1Qbb, 2011. [Online]. Available: https://1.ieee802.org/dcb/802-1qbb

[24] S. Das and R. Sankar, “Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications,” Broadcomreport, Palo Alto, CA, USA, Tech. Rep., 2012.

[25] (2014). *Congestion Management and Buffering in Data Center Networks*. [Online]. Available: http://learn.extremenetworks.com/rs/extreme/images/Congestion-Management-and-Buffering-wp.pdf

[26] A. Arcilla and T. Palmer. (2019). *Broadcom Trident 3 Platform Performance Analysis*. [Online]. Available: https://docs.broadcom.com/doc/12395356

[27] B. Wheeler. (2019). *Tomahawk 4 Switch First to 25.6TBPS*. [Online]. Available: https://docs.broadcom.com/doc/12398014

[28] (2014). *Cisco Nexus 9300 Platform Buffer and Queuing Architecture*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-732452.pdf

[29] (2013). *Arista 7050X3 Series Switch Architecture*. [Online]. Available: https://www.arista.com/assets/data/pdf/Whitepapers/7050X3ArchitectureWP.pdf

[30] M. Alizadeh et al., “Data center TCP (DCTCP),” in *Proc. ACM SIGCOMM Conf.*, Aug. 2010, pp. 63–74.

[31] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the social network’s (datacenter) network,” in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 123–137.

[32] (2021). *BCM88800 Traffic Management Architecture*. [Online]. Available: https://docs.broadcom.com/doc/88800-DG1-PUB

[33] NVIDIA.(2022). *How to Configure Mellanox Spectrum Switch for Lossless RoCE*. [Online]. Available: https://enterprise-support.nvidia.com/s/article/howto-configure-mellanox-spectrum-switch-for-lossless-roce

[34] V. Addanki, W. Bai, S. Schmid, and M. Apostolaki, “Reverie: Low pass filter-based switch buffer sharing for datacenters with RDMA and TCP traffic,” in *Proc. USENIX NSDI*, 2024, pp. 651–668.

[35] (2015). *Priority Flow Control: Build Reliable Layer 2 Infrastructure*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/whitepaperc11-542809.pdf

[36] *Proposal for Priority Based Flow Control*. Accessed: Aug. 17, 2025. [Online]. Available: https://www.ieee802.org/1/files/public/docs2008/bb-pelissier-pfc-proposal-0508.pdf

[37] Mellanox. (2018). *Understanding the Alpha Parameter in the Buffer Configuration of Mellanox Spectrum Switches*. [Online]. Available: https://support.mellanox.com/s/article/howto-configure-mellanox-spectrum-switch-for-lossless-roce

[38] *Cisco Nexus 9000 Series NX-OS Quality of Service Configuration Guide*. Accessed: Aug. 17, 2025. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/6-x/qos/configuration/guide/b_Cisco_Nexus_9000_Series_NX-OS_Quality_of_Service_Configuration_Guide.pdf

[39] Y. He, N. Batta, and I. Gashinsky, “Understanding switch buffer utilization in CLOS data center fabric,” in *Proc. Workshop Buffer Sizing*, 2019, pp. 1–3.

[40] *Tomahawk*. Accessed: Aug. 17, 2025. [Online]. Available: https://people.ucsc.edu/∼warner/Bufs/tomahawk

[41] V. Addanki, O. Michel, and S. Schmid, “PowerTCP: Pushing the performance limits of datacenter networks,” in *Proc. USENIX NSDI*, 2021, pp. 51–70.

[42] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, “Information-agnostic flow scheduling for commodity data centers,” in *Proc. USENIX NSDI*, 2015, pp. 455–468.

[43] M. P. Grosvenor et al., “Queues don’t matter when you can JUMP them!,” in *Proc. USENIX NSDI*, 2015, pp. 1–14.

[44] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, “Approximating fair queueing on reconfigurable switches,” in *Proc. USENIX NSDI*, 2018, pp. 1–16.

[45] L. Chen, J. Lingys, K. Chen, and F. Liu, “AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 191–205.

[46] V. Jacobson, “Congestion avoidance and control,” in *ACM SIGCOMM*, Aug. 1988, pp. 314–329.

[47] *Intel Tofino*. Accessed: Aug. 17, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[48] N. K. Sharma et al., “Evaluating the power of flexible packet processing for network resource allocation,” in *Proc. USENIX NSDI*, 2017, pp. 67–82.

[49] H. Almasi, R. Vardekar, and B. Vamanan, “Protean: Adaptive management of shared-memory in datacenter switches,” in *Proc. IEEE Conf. Comput. Commun.*, May 2023, pp. 1–10.

[50] Y. Liu, J. Han, K. Xue, R. Li, and J. Li, “L2BM: Switch buffer management for hybrid traffic in data center networks,” in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2023, pp. 1–11.

[51] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, “ACC: Automatic ECN tuning for high-speed datacenter networks,” in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 384–397.

[52] *Ns-3*. Accessed: Aug. 17, 2025. [Online]. Available: https://www.nsnam.org/

[53] Priority-Based Flow Control Enhancements, Standard P802.1Qdt, 2024. [Online]. Available: https://1.ieee802.org/tsn/802-1qdt/

[54] Adaptive PFC Headroom and PTP, Standard IEEE 802.1, 2021. [Online]. Available: https://www.ieee802.org/1/files/public/docs2021/new-lv-adaptive-pfc-headroom-and-PTP-0602-v03.pdf

[55] A. Greenberg et al., “VL2: A scalable and flexible data center network,” in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 51–62.

[56] *Alibaba Storage Flow Size Distribution*. Accessed: Aug. 17, 2025. [Online]. Available: https://github.com/alibaba-edu/High-Precision-Congestion-Control/tree/master/traffic_gen

[57] S. Zhang et al., “BRT: Buffer management for RDMA/TCP mix-flows in datacenter networks,” *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 4, pp. 4146–4160, Aug. 2024.

[58] D. Shan, W. Jiang, and F. Ren, “Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches,” in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 118–126.

[59] M. Apostolaki, L. Vanbever, and M. Ghobadi, “FAB: Toward flow-aware buffer sharing on programmable switches,” in *Proc. Workshop Buffer Sizing*, Dec. 2019, pp. 1–6.

[60] S. Huang, M. Wang, and Y. Cui, “Traffic-aware buffer management in shared memory switches,” in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[61] M. Wang, S. Huang, Y. Cui, W. Wang, and Z. Liu, “Learning buffer management policies for shared memory switches,” in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 730–739.

[62] V. Addanki, M. Apostolaki, M. Ghobadi, S. Schmid, and L. Vanbever, “ABM: Active buffer management in datacenters,” in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 36–52.

[63] V. Addanki, M. Pacut, and S. Schmid, “Credence: Augmenting datacenter switch buffer sharing with ML predictions,” in *Proc. USENIX NSDI*, 2024, pp. 613–634.

[64] D. Shan et al., “Occamy: A preemptive buffer management for on-chip shared-memory switches,” in *Proc. 20th Eur. Conf. Comput. Syst.*, Mar. 2025, pp. 1365–1382.

[65] H. Huang et al., “Re-architecting buffer management in lossless Ethernet,” *IEEE/ACM Trans. Netw.*, vol. 32, no. 6, pp. 4749–4764, Dec. 2024.