Revisiting Flow Control in Node-Centric Datacenter Networks

Peirui Cao, Rui Ning, Guangyu Zhao, Zhaochen Zhang, Chang Liu, Yunzhuo Liu, Rui Li, Chengyuan Huang, *Member, IEEE*, Tao Sun, Zhiqiang Li, Guihai Chen, *Fellow, IEEE*, Baochun Li, *Fellow, IEEE*, and Chen Tian, *Senior Member, IEEE*

***Abstract*—Node-centric Data Centers (NDCs) are highly flexible, cost-efficient, and failure-resilient, and have gained growing popularity in recent years. However, RDMA technology used in NDC still faces challenges, including high retransmission overhead, Head-of-Line Blocking (HoLB) and deadlock problems. Existing solutions for traditional data centers cannot simultaneously address these issues due to the unique topology and server transmission characteristics of NDC. In this paper, we propose a per-port flow control named PortFC for NDC. PortFC addresses the above problems through the designs of a Pause/Resume control signal, a per-port queue allocation method, an egress-detecting per-port flow control mechanism, and a server-aware queue scheduling method. Our evaluation shows that PortFC is free from retransmission, capable of eliminating HoLB and avoiding deadlocks. PortFC achieves 1.7-8.0 times higher throughput and reduces latency by 11.7%-87.7% compared to the state-of-the-art lossy RDMA based on IRN and the lossless RDMA method based on PFC. In particular, PortFC still demonstrates good performance in a Rail-only NDC with heterogeneous bandwidth domains.**

***Index Terms*—Node-centric data center, data center networks, flow control.**

I. Introduction

NODE-CENTRIC data centers (NDCs) [1], [2], [3], [4], [5], [6], [7], [8] have gained significant attention recently due to their high flexibility, low construction costs, and strong resilience to handle network node and link failures. Specifically, NDC network architectures based on the BCube [6] topology, such as the Rail-only [1] and Alibaba HPN [8] architectures, have demonstrated great potential in supporting the most focused-on applications at present, e.g., the training of Large Language Models (LLMs). As a result, constructing high-performance and stable networks on such architectures has become a focal point for many researchers and service providers.

Remote Direct Memory Access (RDMA) over Converged Ethernet Version 2 (RoCEv2) offloads network stacks into hardware and lays the foundation for constructing networks with both high throughput and low latency. However, with the rapid increase in link bandwidth, the throughput of devices cannot match that of links, and RDMA networks face a high risk of packet loss, leading to severe performance degradation. Even with a low packet drop rate (0.4%), the application-level goodput can degrade to zero [9]. This problem is further highlighted in BCube architectures, where low-end switches with smaller buffers are often chosen for higher cost-efficiency.

There are two general approaches to cope with RDMA packet loss. One is to perform loss recovery when packet loss occurs, as exemplified by the state-of-the-art method IRN [10]. IRN enhances network performance and reduces unnecessary queuing without relying on Priority Flow Control (PFC) by implementing efficient loss recovery mechanisms and Bandwidth-Delay Product flow control (BDP-FC) in the NIC. This design is referred to as lossy RDMA. In contrast, the another approach is lossless RDMA, which avoids packet loss by enabling PFC. However, neither approach truly brings RDMA to its full potential under BCube, as they suffer from the following key limitations:

- **High retransmission overhead.** Loss recovery methods like IRN rely on a timeout threshold to determine packet loss and trigger retransmission. However, an appropriate timeout threshold is often variable and difficult to obtain, particularly in BCube where the relay path is more complex. A small threshold can decrease latency but causes large amounts of spurious retransmissions that degrade network throughput. In contrast, a large threshold reduces spurious retransmissions but leads to high retransmission latency.
- **Unnecessary Head-of-Line Blocking (HoLB).** PFC ensures a lossless RDMA network but faces severe HoLB problems. HoLB problems occur when a flow at the front of a queue experiences a delay or is blocked, causing all subsequent flows in the same queue but to different destinations to be delayed as well. Existing methods that alleviate HoLB problems focus on more general network architectures where the destinations of flows in a queue can vary greatly. However, the intrinsic architecture of BCube greatly reduces such possibilities, making it possible to further reduce HoLB problems by appropriately scheduling the queues. Existing methods ignore this feature of BCube and suffer from unnecessary HoLB.
- **Network paralysis caused by deadlock.** In BCube, nodes also participate in relay forwarding, increasing the complexity of the network paths. Additionally, switch

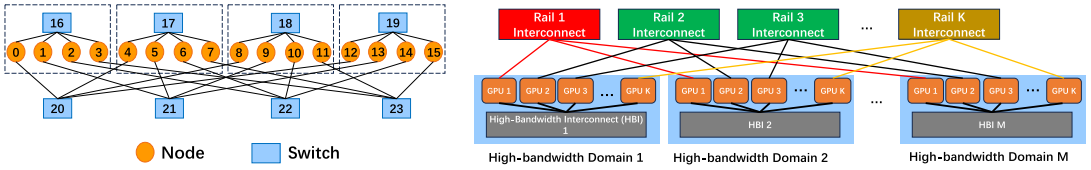
Received 25 August 2025; revised 28 December 2025 and 16 February 2026; accepted 24 March 2026; approved by IEEE TRANSACTIONS ON NETWORKING Editor P. P. C. Lee. Date of publication 27 March 2026; date of current version 9 April 2026. This work was supported in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant JYB2025XDXM118; in part by NSFC under Grant 62325205 and Grant U25B2035; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20243053; and in part by Nanjing University-China Mobile Communications Group Company Ltd., Joint Institute. An earlier version of this paper appeared in ICS 2025 (https://dl.acm.org/doi/10.1145/3721145.3725749). *(Corresponding author: Zhaochen Zhang.)*

Peirui Cao, Rui Ning, Zhaochen Zhang, Chang Liu, Yunzhuo Liu, Rui Li, Chengyuan Huang, Guihai Chen, and Chen Tian are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: zhaochen.zhang@outlook.com).

Guangyu Zhao, Tao Sun, and Zhiqiang Li are with China Mobile Research Institute, Beijing 100053, China.

Baochun Li is with the University of Toronto, Toronto, ON M5S 3E3, Canada.

Digital Object Identifier 10.1109/TON.2026.3678334



(a) BCube topology.

(b) Rail-only topology [1]. The GPU can be regarded as a node, and the high-bandwidth interconnect can be regarded as a switch.

Fig. 1. Network topology examples of Node-centric data center (NDC).

queues are influenced by the next two-hop switches and nodes, not just by adjacent switches. This makes existing solutions ineffective in truly resolving deadlocks in BCube, which can lead to network paralysis.

To bring RDMA to its full potential under BCube architectures, this paper presents PortFC, a per port flow control method that simultaneously achieves high throughput and low latency. PortFC addresses the key limitations through the following designs: First, PortFC enables lossless RDMA by using a Pause/Resume signal-based flow control mechanism similar to PFC and is free from retransmission. Second, by designing a per-port queue allocation (§III-B) to match the NDC topology (§III-A) and egress-detecting per-port flow control method (§III-C) to solve the HoLB problems, PortFC guarantees low latency and high throughput. Third, PortFC proposes a deadlock-free strategy (§III-D) that utilizes queue scheduling combined with the node-aware feature that nodes can also act as forwarding nodes to eliminate potential deadlock loops in NDC. Finally, we propose asymmetric traffic engineering (§III-E) for domains with heterogeneous bandwidth as a supplement for special NDCs like Rail-only.

We implement PortFC (§IV) on Tofino2 switch [11] and DPDK [12], and our evaluation (§V) shows that PortFC achieves the following performance:

- **Deadlock-free and HoLB-free.** Our evaluation on the NS3 simulator shows that PortFC successfully avoids deadlock and HoLB problems in BCube.
- **High throughput and low latency.** The throughput of PortFC outperforms IRN, Go-Back-N and improved PFC design by 1.7-2.3 times and 2.4-21.6 times, 1.9-8.0 times, respectively. Compared to IRN, Go-Back-N and improved PFC design, PortFC reduces the Flow Completion Time (FCT) and tail latency by 11.7%-69.2%, 58.4%-97.9% and 19.8%-87.7%, respectively.
- **Scalability.** PortFC demonstrates significant performance gains across different topology sizes, various traffic patterns in the simulation experiments.

II. Background and Motivation

A. Bring NDC Into Focus

NDCs [1], [2], [3], [4], [5], [6], [7], [8] enjoy several key advantages and have gained significant attention in recent years. Firstly, NDC provides well flexibility for data center migration, deployment, and scalability. Secondly, NDC supports common communication patterns efficiently and offers higher bottleneck throughput with lower deployment costs, as it requires fewer switches and has lower performance demands on them. Lastly, NDC demonstrates greater stability in handling switch and node failures due to its multiple equivalent paths and switch layers, ensuring good transmission performance even during failures.

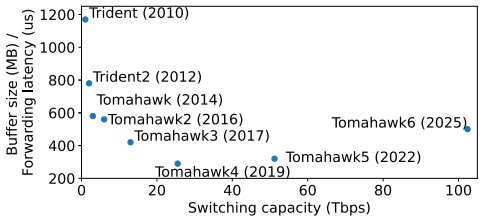


Fig. 2. The trends in switch chip architecture.

BCube [6] is a highly regarded topology within NDC network architectures, as shown in Figure 1(a). It allows nodes to participate in forwarding, and it can be modularly and recursively expanded. This results in low construction costs, strong resilience to node failures, and ease of migration due to its modular structure.

These representative advantages have led to its widespread application in recent years. For instance, the currently popular GPU training network Rail-only architecture [1], [4] is based on BCube principles, where GPU nodes can be likened to nodes, and rail interconnects and high-bandwidth interconnects (using NVLink [13] and NVSwitch [14]) can be likened to switches, as shown in Figure 1(b). Both BCube and Rail-only architectures utilize nodes or GPUs to handle a portion of the transmission responsibilities, in addition to switches. From the perspective of the modeling level, the difference between the two is that the bandwidths of the scale-up and scale-out parts in the Rail-only model are not equal. The latest Alibaba HPN [8] architecture is also capable of combining multiple Rail-only topologies. Therefore, this paper constructs an NDC structure (§III-A) capable of representing topologies such as Rail-only and BCube.

While these architectures (such as NVIDIA GPU fabrics [13], Huawei UB-Mesh [15], Alibaba HPN [8]) are often viewed as rail-optimized switch-centric designs, we categorize them as node-centric because the multi-homed nodes actively participate in cross-rail traffic coordination and bridge the scale-up and scale-out domains.

B. The Trends in Switch Chip Architecture

Starting in 2025, single-port 400 Gbps and 800 Gbps have become the current standard [16], [17]. However, the growth rate of network device buffers is far from keeping pace with the increase in port bandwidth [18]. The latest Tomahawk 6 has a switching capacity (total port bandwidth) of 102.4 Tbps, a buffer size of 250 MB, and a forwarding latency on the order of 500 ns. As shown in Figure 2, the ratio of switch buffer capacity to its forwarding latency is difficult to scale continuously in the same way as the switching capacity.

This development trend in network devices indicates that traffic within the network is becoming more bursty and harder to control. However, the commercial off-the-shelf (COTS)

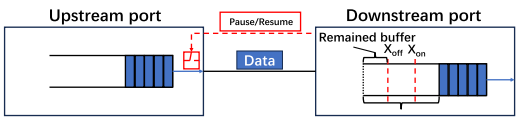
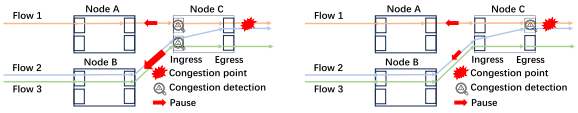


Fig. 3. The PFC mechanism.



(a) Ingress detection. (b) Engress detection.

Fig. 4. HoLB examples in BCube.

micro switches used in BCube networks typically lack large buffers. As a result, BCube networks are prone to packet loss due to switch buffer overflow when facing increasingly bursty traffic. Packet loss often leads to higher retransmission delay, thereby negatively impacting the user experience at the application layer. Furthermore, in common traffic patterns within data centers, such as computational traffic and storage traffic, there are many-to-one incast scenarios [19]. Shallow-buffer switches are also highly susceptible to buffer exhaustion in a short time during large-scale incast traffic, leading to packet loss.

C. Deploy RDMA Over Ethernet in NDC

1) NDC Based on RoCEv2 Networks: Large vendors [3], [9], [20], [21] have begun large-scale deployment of RDMA in data centers to achieve better network performance. Since IB technology is closed-source and many already deployed data centers are based on Ethernet networks [22], there have been advanced RoCE designs, e.g., Resilient RoCE [23], IRN [10] etc., that could work with a lossy network. However, supporting RoCE in lossy networks requires handling packet retransmission using timeouts, selective acknowledgments, etc., which may not only complicate the NIC design but also hurt network latency and throughput performance. As a result, lossy RDMA may not be able to substitute lossless RDMA in all cases. Therefore, we focus on NDC networks with lossless RoCEv2 (RDMA over Converged Ethernet v2) [24] and conduct extensive experiments to confirm that our design outperforms lossy RDMA designs in NDC networks.

However, current lossless RDMA algorithms are mainly designed for switch-centric tree topologies and are not suitable for NDC. Current research on NDC primarily focuses on traffic scheduling optimization, routing algorithm design [25], [26], fault diagnosis [27], [28] and has yet to fully explore how to implement refined flow control strategies at the link layer to directly ensure the effective operation of lossless networks.

2) Head-of-Line Blocking Issues in NDC: PFC [29] is a flow control algorithm that supports RoCEv2 in a lossless network. As Figure 3 shows, PFC sets X_{OFF} and X_{ON} as threshold values for the ingress port queue of the switch. When the length of the ingress port queue exceeds X_{OFF}, the downstream receiver sends a Pause message to halt the transmission from the upstream sender. Conversely, once the length of the ingress port queue falls below X_{ON}, the receiver sends a Resume message to restart the transmission from the sender. When the headroom (the remained buffer between the total buffer size and X_{OFF}) is greater than one BDP, the switch buffer will not overflow. PFC causes serious Head-of-Line Blocking (HoLB) when different flows share the same nodes.

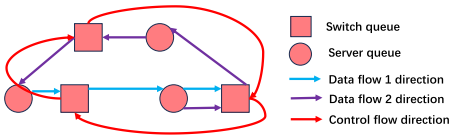


Fig. 5. A deadlock example in NDC.

As shown in Figure 4(a), the congestion point affects both flow 1 and flow 2 simultaneously. Traditional PFC mechanisms conduct congestion detection at ingress ports of nodes, which cannot recognize which specific flow is congested; hence, Pause frames are triggered to pause transmission at the ports of node A and node B. Consequently, the victim flow 3 will also be paused. Although the number of switch layers in the NDC is low, the Pause frame can still be transmitted to another switch through relay node nodes.

BFC [30] performs per-hop, per-flow flow control at an overly fine granularity and does not incorporate semantic-level queue classification. Such excessively fine-grained backpressure control leads to high management complexity in NDCs, and as the number of concurrent flows or port demands increases, it suffers from queue contention, state overhead, and deadlock. TCD [31] and ACC [32] target PFC-enabled networks and mitigate the impact of HoLB after it occurs. However, PFC can introduce deadlocks in NDC, making these approaches unsuitable for NDCs.

If we change the congestion detection from ingress ports to egress ports, we can conveniently mark all flows at the egress ports and then simply send Pause frames with the congestion information to the upstream switches. Simultaneously, we can use the priority queue to distinguish and manage victim flow 3 and congested flow 2, as shown in Figure 4(b). Unfortunately, trivial queue allocations will lead to unbearable resource costs. Fortunately, we can utilize the NDC topology features (§III-B) with the penetrate node to synchronize congestion information to multi-hop switches and conduct per-port flow control (§III-C), requiring only a limited number of queues to achieve high performance.

3) Existing Works Fail to Resolve Deadlock in NDC: Under the Pause/Resume mechanism of PFC, due to the presence of loops in the network, multiple flows may enter the same queue, causing congestion, which then propagates hop-by-hop through the network, eventually forming a pause loop, which is called Circular Buffer Dependency (CBD). Then, every port is waiting for the congestion to be alleviated at the next hop, resulting in a situation where each node in the loop cannot send packets, leading to a deadlock.

In NDC topologies, there is still no effective solution to date. Previous works about deadlock prevention can be divided into two groups. One is to restrict the method of routing in order to avoid appearance of CBD [9], [33], [34]. The other is to manage buffers in the queue more properly to avoid deadlock [35]. However, the former causes waste of bandwidth in data center and reduce performance, which is unacceptable. The latter always creates many priority levels, which is expensive for practice. Both of them cannot obtain desirable outcomes in NDC. Figure 5 illustrates a potential deadlock scenario in a two-dimensional NDC. The figure assumes the presence of two four-hop[1] flows in the network, with the flow control algorithm

[1]No matter how the topology of the NDC changes, or what types of GPUs, nodes, and switches are used, any traffic that passes through a NIC once is considered to have traversed one hop.

deployed at the switches, and the nodes merely forwarding flow control information received from downstream switches. These two flows form a loop across six nodes in the network. When a queue at one of the switches within this loop becomes congested, as shown by the red arrow, the pause messages form a loop with the host's forwarding Every switch queue within this loop is in a paused state, waiting for the downstream switch queue to empty, thus causing the deadlock issue.

Takeaway: To the best of our knowledge, no prior work has shown that simply enabling the above two classes of solutions together can simultaneously eliminate both HoLB and deadlock in PFC. For example, TCD [31] distinguishes port states based on the duration for which a port is paused to mitigate HoLB, a mechanism that cannot be directly incorporated into GFC [36]. BFC [30] and Pyrrha [37] redesign flow-control protocols to avoid HoLB. However, BFC cannot avoid deadlock in NDC networks. Moreover, these schemes theoretically require an unbounded number of queues, and their performance degrades significantly when the number of available queues is limited. PortFC, by contrast, is specifically designed for NDC networks and mitigates both HoLB and deadlock using a fixed number of queues. In contrast, PortFC employs a unified flow-control protocol design that mitigates both HoLB and deadlock.

D. Goals

To address the aforementioned features and challenges, we propose PortFC to meet the following goals. To the best of our knowledge, PortFC is the first approach to implement per-port flow control using distinct switch/node egress-port queue allocations to achieve a high-bandwidth, low-latency, deadlock-free NDC network.

G1: Retransmission-free. The PortFC needs to dynamically adjust upstream node transmission based on the buffering capacity of downstream switches. It is important to minimize the occupancy of switch buffers while ensuring that the transmission from upstream nodes does not cause the buffers of downstream switches to overflow. This approach not only enables lossless data transmission at the link layer but also effectively reduces the queuing delay of packets during transmission by utilizing short queue strategies, thereby optimizing overall network performance and efficiency.

G2: HoLB-free. PortFC should be designed to exploit the intrinsic feature of NDC architectures to eliminate HoLB problem, while minimizing the utilization of switch hardware resources.

G3: Deadlock-free. The design of PortFC should avoid deadlocks. In a NDC network, nodes can perform routing functions to forward packets to other nodes. It is necessary to design a flow control algorithm that ensures control messages do not create deadlocks in the network.

E. Motivation

Specifically, in NDCs, nodes act as both endpoints and intermediate forwarding units. Traditional flow control schemes often lead to CBD deadlocks and severe HoLB because they fail to distinguish between traffic destined for the local node and traffic requiring further relay. PortFC is motivated by the necessity to decouple these distinct traffic types to ensure lossless transmission within high-radix NDC topologies. Essentially, PortFC implements a topology-aware flow control mechanism that categorizes traffic based on its structural path requirements.

III. Design

A. NDC Structure

It is necessary to briefly introduce the construction method of the general NDC Data Center network. $NDC(n,k),(k \geq 1)$ is a recursively defined structure, consisting of $nNDC(n,k-1)$ units and n^k switches, each with n ports. Each node in $NDC(n,k)$ has $k+1$ ports, which are connected sequentially to $k+1$ switches from layer 0 to layer k. According to the above definition, the number of nodes in $NDC(n,k)$ is n^{k+1}, and the number of switches is $(k+1)n^k$. These switches are distributed across layers 0 to k, with n^k switches at layer $k-1$. As shown in Figure 6, $NDC(4,0)$ within the dashed box consists of one 4-port switch and four nodes, and four $NDC(4,0)$ units are connected through four 4-port layer-one switches to form $NDC(4,1)$. It is noteworthy that nodes located in the same position in each $NDC(4,0)$ are connected through the same layer-one switch.

Commercial electric switches can currently achieve 128 ports, which is already capable of supporting a $k = 1$ NDC with 16,384 nodes. Moreover, by leveraging the reconfigurable capabilities of Optical Circuit Switches (OCS), We can directly combine multiple NDCs into a very large scale without increasing the level. To maintain the general formulation of $NDC(n,k)$, the design retains k as a variable.

Instead of ingress port queue detection, which is used by PFC algorithm, the flow control in this paper detects congestion based on the egress port queue. Detecting network congestion at the egress port to control congested traffic can alleviate the HoLB issue associated with ingress port flow control algorithms. Specifically, the queue allocation scheme of the NDC per-port flow control algorithm designed in this paper is divided into two types: switch port queue allocation and node port queue allocation.

B. Per-Port Queue Allocations

1) Two Types of Queues for Switch Ports: In the NDC, the adjacent nodes of a switch are all nodes, allowing the traffic flowing through the switch to be divided into two categories. The first category is traffic that needs to be forwarded by the next-hop node because the next-hop node is not the destination node. The second is traffic reaching the destination node at the next hop.

As shown in the bottom right corner of Figure 6, there are several flows: $F11$, $F12$, $F13$, $F14$ from node 11, $F15$ from node 3 and $F16$ from node 7. Considering the simple case where the next hops of flows are their destinations respectively, and $F11$ and $F16$ should be queued in the same queue $Q3$ as node 15 is the next hop of switch 23. For the other flows, the situation is relatively more complex. Assume all ports and queues are indexed starting from 0 on the left-hand side. In the next switch where the flows will be forwarded, $F12$ and $F13$ will be forwarded from port 0 and port 1 of switch 19 respectively, while $F14$ and $F15$ will be forwarded from port 2 of switch 19. Corresponding to the forwarding ports, $F12$, $F13$ are queued in $Q0$, $Q1$ respectively, and $F14$, $F15$ are both queued in $Q2$. Moreover, there's a high-priority queue (HQ) for storing some control messages in the network (like ACK and CNP messages). HQ enables control information to be preferentially transmitted in the network without being affected by flow control, achieving lower latency. Specifically, when there are packets in HQ, the scheduler will prioritize sending them. Since the control information occupies little bandwidth, it hardly blocks the transmission of data packets.

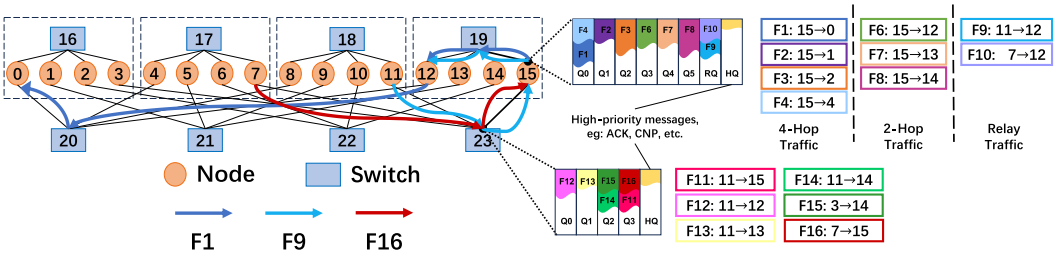


Fig. 6. The NDC structure and example of queue allocation on the port of switches and nodes.

The queue allocation strategy of the NDC per-port flow control algorithm at the switch port is based on which egress port the packet will go through at the next-2-hop switch (since the next hop connected by the NDC switch is a node). The packet is placed into the queue corresponding to the egress port of the next-2-hop switch at the current switch port. If the packet reaches its destination at the next hop, it is placed into a separate queue. The number of queues is allocated based on the number of ports on the downstream switch, which ensures that NDC per-port flow control uses minimal hardware resources on the switch.

For an $NDC(n,k)$ topology, where each switch has n ports, only $n-1$ queues are needed at each switch port to store packets destined for the $n-1$ possible egress ports of the next-hop switch. Additionally, only 1 queue is needed to store packets that reach the destination node at the next hop. If needed, 1 high-priority queue can also be added. In summary, in the design of the per-port flow control (§III-C), each port of a switch in the $NDC(n,k)$ topology only requires $n+1$ queues.

2) Three Types of Queues for Node Ports: In the NDC topology, there are three types of traffic at the node ports. 1) **The four-hop traffic** originates from the node itself. 2) **The two-hop traffic** originates from the node itself. 3) **Node-forwarding traffic (relay traffic)** is traffic originating from other nodes, which needs to be forwarded by the current node. For these three types of traffic, the per-port flow control algorithm provides different queue allocation schemes. In an $NDC(n,k)$ topology, each switch has n ports. The four-hop traffic originating from the node will be placed into queues numbered from 0 to $(n-2)$ based on the egress port number of the next-hop switch (with $n-1$ possible values). The two-hop traffic originating from the node will be placed into queues numbered from $(n-1)$ to $(2n-3)$ based on the egress port number of the next-hop switch (with $n-1$ possible values). The third type of traffic, which needs to be forwarded by the node, will be placed into a 'relay queue' numbered $(2n-2)$.

In the top right corner of Figure 6, an example of queue allocation at switch ports is presented. Flows are categorized into three types as mentioned before. Four-hop traffic includes $F1$, $F2$, $F3$, and $F4$. For example, $F1$ traverses switch 19, node 12, switch 20 and finally reaches node 0, and the other flows in this type follow a similar path. Two-hop traffic consists of $F6$, $F7$, and $F8$, which pass through switch 19 and then reach their respective destinations. Node-forwarding traffic contains $F9$ and $F10$, flowing only via node 15.

For the queue allocation at the port of node 15 connected to switch 19, for four-hop traffic, since $F1$ and $F4$ are forwarded through port 0 of switch 19, they are queued in $Q0$. $F2$ and $F3$ are queued in $Q1$ and $Q2$ respectively. For two-hop traffic, $F6$, $F7$, and $F8$ are queued in $Q3$, $Q4$, and $Q5$ respectively as they are forwarded through port 0, port 1, and port 2 of switch 19 respectively. For the relay flows $F9$ and $F10$, they are both queued in RQ. Analogous to switch queue allocation, HQ is used to store high-priority packets.

From the above example, it can be seen that the queue allocation on the node is slightly more complex than on the switch. This more refined classification makes it more convenient for per-port flow control algorithms to manage traffic (§III-C), resulting in finer granularity and better control effects. Even so, in an $NDC(n,k)$ topology, the number of port queues on the node is still linearly proportional to the number of switch ports. Specifically, in the $NDC(n,k)$ topology, the traffic originating from the node can be divided into $k+1$ types according to the number of hops, with each type requiring $n-1$ queues (corresponding to $n-1$ possible egress port choices for the next hop) for storage, one queue as a relay queue to hold packets that need to be forwarded by the node, and one high-priority queue. In summary, in an $NDC(n,k)$ topology, a node port requires a total of $(k+1)\times(n-1)+2$ queues (where k is generally less than or equal to 4). Current commercial switches can generally support dozens or even more than one hundred queues [30], [38], [39], so the queue allocation scheme designed in this paper can still support large NDC networks and has good scalability.

C. Per-Port Flow Control

In designing the per-port flow control algorithm for NDC, congestion detection occurs at the switch's egress port. Based on §II-C.2 analysis, compared to ingress-port-based congestion detection algorithms, egress-port-based ones effectively mitigate HoLB caused by the impact of congested traffic on non-congested traffic.

This section will detail the control logic of per-port flow control in the NDC network through three parts: congestion detection and mitigation at switch ports, node handling of flow control messages, and switch handling of flow control messages.

1) Congestion Detection and Mitigation: In §III-B.1, the allocation of two types of queues on the switch is described. Thus, in the per-port flow control algorithm of NDC, two sets of congestion detection and mitigation thresholds are defined.

Inter-Hop Forwarding Queue (IHFQ): For the queues in the switch's egress ports that store the traffic for the next two hops, which pass through different egress ports of the switch ($n-1$ queues, numbered from 0 to $n-2$), the congestion threshold X_{off} and the congestion mitigation threshold X_{on} are set. Destination-Direct Queue (DDQ): DDQ stores traffic where the next hop is the destination node (1 queue, numbered $n-1$), the congestion threshold $\hat{X}_{off}$ and the congestion mitigation threshold $\hat{X}_{on}$ are set.

Algorithm 1 Packet Reception Operation at the Switch Egress Port in $NDC(n,k)$

```
1: qIdx = GetQIdx(packet)
2: if CheckBufferAddmission(packet.size) == false then
3:     Buffer overflow, drop this packet.
4: else
5:     qlen = 0
6:     if qIdx < n − 1 then
7:         idx = 0
8:         while idx < n − 1 do
9:             qlen = qlen + queueLen[idx]
10:            idx + +
11:        end while
12:        if upstreamIsPaused == false & qlen >= X_off
           then
13:            upstreamIsPaused = true
14:            for All other ports do
15:                PauseFrame=GenFrame(Pause, qIdx)
16:                SendToUpstream(PauseFrame)
17:            end for
18:        end if
19:    else if qIdx == n − 1 then
20:        qlen = queueLen[n − 1]
21:        if upstreamIsPaused==false & qlen ≥ X̂_off then
22:            upstreamIsPaused = true
23:            for All other ports do
24:                PauseFrame=GenFrame(Pause, qIdx)
25:                SendToUpstream(PauseFrame)
26: end for
27:        end if
28:    else
29:        return.
30:    end if
31: end if
```

Algorithm 2 Packet Transmission Operation at the Switch Egress Port in $NDC(n,k)$

```
1: queueLen[qIdx] = queueLen[qIdx] − packet.size
2: qlen = 0
3: if qIdx < n − 1 then
4:     idx = 0
5:     while idx < n − 1 do
6:         qlen = qlen + queueLen[idx]
7:         idx + +
8:     end while
9:     if upstreamIsPaused == true & qlen <= X_on then
10:        upstreamIsPaused = false
11:        for All other ports do
12:            ResumeFrame=GenFrame(Resume, qIdx, portIdx)
13:            SendToUpstream(ResumeFrame)
14:        end for
15:    end if
16: else if qIdx == n − 1 then
17:    qlen = queueLen[n − 1]
18:    if upstreamIsPaused == true & qlen <= X̂_on then
19:        upstreamIsPaused = false
20:        for All other ports do
21:            ResumeFrame=GenFrame(Resume, qIdx, portIdx)
22:            SendToUpstream(ResumeFrame)
23:        end for
24:    end if
25: else
26:    return.
27: end if
```

Detection of congestion occurrence on switches. When a packet arrives at the egress port of a switch, congestion detection is performed. The detailed logic is shown in Algorithm 1. Lines 4 to 18 of Algorithm 1 present the processing logic when a packet enters the IHFQ in $NDC(n,k)$. In this paper, the length of these $n-1$ queues and whether they exceed the congestion threshold are used as the criteria for determining whether the port is congested. When a packet is received at the egress port of a switch and enters the IHFQ, the lengths of $n-1$ queues are checked to see if they exceed the congestion threshold X_{off}. If they do, a pause message (containing the queue number and the congested port number) is sent upstream to the ports connected to other ports, pausing the queues corresponding to the congested port at the port of upstream node.

Similar to the operation when the IHFQ receives a packet, the logic for detecting whether congestion has occurred when a packet is received by the DDQ, which stores packets whose next hop is the destination, is given in lines 19 to 27 of Algorithm 1. Additionally, the flow control algorithm proposed in this paper does not control the transmission of the highest priority. Since such packets are usually few, if a packet belongs to the highest priority, congestion detection is not performed for the highest-priority queue, as shown in line 29 of Algorithm 1.

Detection of congestion mitigation on switches. When a queue at the egress port of a switch is about to send a packet, the congestion state of the switch needs to be reassessed, checking whether the current queue length meets the condition for congestion mitigation. Lines 3 to 15 of Algorithm 2 present the logic for determining congestion mitigation when sending a packet from the IHFQ. This determination is based on whether the lengths of the $n-1$ queues are less than the given threshold X_{on}. Similar to the congestion mitigation and sending congestion mitigation messages to upstream nodes for the IHFQ, lines 16 to 24 of Algorithm 2 provide the logic for handling congestion mitigation for the DDQ on the switch. Similarly, no flow control logic is applied to the highest-priority packets when they are sent.

The variable *upstreamIsPaused* is used to determine whether it is necessary to send a control message upstream, which ensures that the same congestion or congestion mitigation information is not repeatedly reported to the upstream nodes.

2) Handling Messages at Nodes: Since there are two types of queues on the switch that can generate flow control messages, there are corresponding operations to receive these two types of flow control messages on the upstream node. First, the node parses the content of the flow control message to determine the congested (or congestion-mitigated) port number downstream, the queue number causing the congestion (or congestion mitigation), and the type of control message (Pause or Resume). As shown in lines 4 to 11 of Algorithm 3, when a node receives a flow control message generated by the IHFQ on the switch, it sets the corresponding state for its

Algorithm 3 Handling Flow Control Messages Received by the Switch in $NDC(n,k)$

```
1:  congestedQIdx = frame.qIdx
2:  congestedPortIdx = frame.portIdx
3:  type = frame.type
4:  if congestedQIdx < n − 1 then
5:      qIdx = congestedPortIdx
6:      if type == Pause then
7:          queueState[qIdx] = Paused
8:      else
9:          queueState[qIdx] = Resume
10:         queue[qIdx].TriggerSend()
11:     end if
12: else if congestedQIdx == n − 1 then
13:     offset = n − 1
14:     qIdx = congestedPortIdx + offset
15:     if type == Pause then
16:         queueState[qIdx] = Paused
17:     else
18:         queueState[qIdx] = Resume
19:         queue[qIdx].TriggerSend()
20:     end if
21:     for All other ports do
22:         RelayFrameToUpstream(frame)
23:     end for
24: else
25:     assert error.
26: end if
```

own first $n-1$ queues based on the type of control message for the queue numbered $congestedQIdx$. Because in the NDC topology, traffic is classified into four-hop traffic and two-hop traffic. When a node receives a control message from the IHFQ on a downstream switch, it means that the four-hop traffic sent by the node may continue to congest the downstream port numbered $congestedPortIdx$, so the local queue corresponding to the congested downstream port should be paused.

If the node receives a flow control message from the DDQ on the downstream switch, it indicates that the queue on the downstream switch destined for the next-hop node is congested. This traffic may be two-hop traffic originating from the node or four-hop traffic from a further upstream point. Thus, the corresponding queue numbered $congestedPortIdx$ in the DDQ on the node (numbered from $(n-1)$ to $(2n-3)$) should be paused since the packets in this queue will aggravate the congestion in the downstream port. As shown in line 14, an offset of $n-1$ must be added to the queue number indicated by $congestedPortIdx$. The above logic corresponds to lines 12 to 20 in Algorithm 3. Additionally, lines 21 to 23 show that the node needs to propagate the flow control message further upstream to control the effect of the four-hop traffic sent from the upstream on the congested port. If the node does not forward this pause frame, the upstream switch may keep sending packets to the node, resulting in the accumulation of packets on the node, causing more queuing delay and thus affecting the throughput rate.

Observing the node's response to congestion in the two types of queues on the downstream switch, it can be noted that congestion caused by traffic in the IHFQ on the switch (traffic that needs to be forwarded by the next-hop node) will only pause the corresponding queue on the upstream node of that switch. In contrast, congestion caused by traffic in the DDQ on the switch (queues storing traffic for which the next-hop node is the destination node) will pause both the corresponding queue on the upstream node and the corresponding queue on the upstream switch. Since new congestion is not detected at the node, in the flow control algorithm designed in this paper, congestion will at most propagate three hops in the network.

Algorithm 4 Handling Flow Control Messages Received by the Switch in $NDC(n,k)$

```
1: congestedPortIdx = frame.portIdx
2: type = frame.type
3: qIdx = congestedPortIdx
4: if type == Pause then
5:     queueState[qIdx] = Paused
6: else
7:     queueState[qIdx] = Resume
8:     queue[qIdx].TriggerSend()
9: end if
```

3) Handling Messages at Switches: The processing of control messages received by the switch is relatively straightforward, as described in Algorithm 4. When the switch receives a flow control message, it first parses the port information of the downstream switch that is congested (or congestion-mitigated) and the type of the flow control message (Pause or Resume). Based on the type of the flow control message, the switch then pauses or resumes the sending operation for the queue corresponding to the congested port.

D. Deadlock-Free Strategy

The example of deadlock in NDC, as Figure 5 shows, is basically discussed in §II-C.3. This paper analyzes the root cause of the NDC deadlock scenario: Not only switch-forwarded traffic, but also node-forwarded traffic in NDC, can form CBD loops, causing deadlocks. Unlike traditional PFC, where all traffic at both switches and nodes is placed in a single queue, this paper designs a queue allocation strategy and flow control logic that successfully break the flow control loop, effectively avoiding the aforementioned deadlock issue. Unlike improved PFC [40], the queue allocation policy proposed in this paper can be summarized as follows. At the switch, traffic is divided into two types: traffic that needs to be forwarded by the next-hop node and traffic for which the next-hop node is the final destination. These two types of traffic are placed into two different queues, denoted as A type queues and B type queues, respectively.

At the node, the traffic types are slightly more complex and divided into three categories: four-hop traffic originating from the node, two-hop traffic originating from the node, and traffic from other nodes that needs to be forwarded by the current node. These three types of traffic are placed into three different queues, denoted as a, b, and c, respectively.

$$switch_i Q_B \geq \hat{X}_{off} \Longrightarrow Pause(node_{i-1} Q_b) \quad Pause(switch_{i-2} Q_A) \tag{1}$$

$$switch_i Q_A \geq X_{off} \Longrightarrow Pause(node_{i-1} Q_a) \tag{2}$$

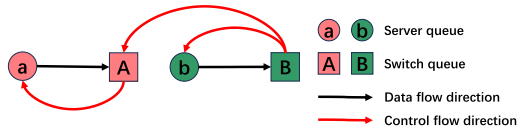


Fig. 7. Avoiding deadlocks in NDC.

The congestion propagation path after detecting congestion at the switch egress port is briefly illustrated in Figure 7. As described in §III-C.2, when congestion is detected in an A type switch queue, the signal will be propagated to an a type queue in previous-hop node. In the other case, when congestion occurs in a B type queue, the congestion message will be transmitted to a b type queue in previous-hop node and an A type queue in switch respectively. Moreover, we can describe the process of congestion propagation more formally with two equations. Equation 1 and Equation 2 show the situations where upstream queues are paused after congestion is detected at the switch. As shown in Equation 1, one scenario where congestion occurs at the egress port of the i^{th} hop switch is when its B type queue becomes congested. This congestion will pause the b type queue at the $i-1$ hop node and the A type queue at the $i-2$ hop switch. Assuming the A type queue of the $i-2$ hop switch also becomes congested due to being paused, according to Equation 2, this congestion will further pause the a type queue at the $i-3$ hop node. Therefore, when the B type queue of the i^{th} hop switch becomes congested, the congestion propagates at most three hops to the $i-3$ hop node before pausing. Another scenario is described in Equation 2, where the A type queue at the egress port of the i^{th} hop switch becomes congested. In this case, the congestion only propagates to the previous hop, the $i-1$ hop node, and does not spread further upstream. Combining the above analysis, it can be seen that the queue allocation strategy and congestion control logic designed in this paper effectively prevent the continuous propagation of congestion, thereby preventing deadlock.

Necessity for Deadlock Prevention: The "aggressive" mechanism of pausing both the upstream node and switch is a deliberate design to break the CBD. In NDC topologies, node-forwarded traffic can form loops; by limiting the propagation of congestion to at most three hops via this mechanism, we successfully break the potential flow control loops and ensure the network is deadlock-free.

Minimal Impact due to Queue Isolation: The concern about bandwidth utilization is mitigated by our per-port queue allocation. Because Destination-Direct Queue (DDQ) traffic is physically separated from Inter-Hop Forwarding Queue (IHFQ) traffic at the switch egress, pausing the congested DDQ path does not affect the line-rate forwarding of other non-congested relay traffic.

E. Asymmetric Traffic Engineering for Heterogeneous Bandwidth Domains

The above design is adaptive to arbitrary NDC topology. In this section, we propose asymmetric traffic engineering for domains with heterogeneous bandwidth as a supplement for special NDCs like Rail-only [1]. As shown in Figure 1(b), we can regard the GPU as a node. For inter-GPU communication, GPUs within a node are interconnected through high-bandwidth, short-range links, such as NVLink [41] or NVSwitch [42], forming what is referred to as the "high-bandwidth domain" [1]. Across nodes, GPUs are linked via high-performance RDMA networks, forming the "network domain". These two domains connect GPUs to form GPU-centric clusters.

From the perspective of the modeling level, the difference between the two is that the different bandwidths of two domain in the Rail-only model are not equal. Even if the traffic with the same volume passes through two domains with different bandwidths, the probability of congestion, HoLB, and deadlock are also different. But if we simply let more traffic go through the high-bandwidth domain, it will also cause this domain to become a performance bottleneck.

Algorithm 5 Asymmetric Traffic Engineering

Input: A asymmetric threshold value p. D_{ij} denotes traffic demands from source i to destination j.

Output: The proportion of traffic from source i to destination j that takes the first hop through the high-bandwidth domain: w_{ij}^H, the proportion of traffic from source i to destination j that takes the first hop through the network domain: w_{ij}^N.

while unallocated D_{ij} **do**
  Randomly assign a flow to w_{ij}^H or w_{ij}^N.
  Detect the inflight traffic volume T_i^N from source i to the first-hop network domain and T_i^H from source i to the first-hop high-bandwidth domain.
  if $T_i^H \geq pB_H$ **then**
    // Reroute the overflowing traffic from the first-hop high-bandwidth domain to the first-hop network domain. $w_{ij}^H = \frac{pB_H}{T_i^H}$.
    $w_{ij}^N = \frac{w_{ij}^N p + (T_i^H - pB_H)}{T_i^N}$.
  end if
end while
Return w_{ij}^N, w_{ij}^H.

Therefore, We additionally introduce an early warning asymmetric threshold for the high-bandwidth domain to assist in solving the unfairness problem of traffic engineering towards the high-bandwidth domain. In Algorithm 5, Asymmetric Traffic Engineering (ATE) operates by iteratively processing unallocated traffic demands D_{ij} from source i to destination j. The capacity of the High-bandwidth domain is B_H, and the capacity of the network domain is B_N. The algorithm initially makes a probabilistic decision to route traffic through either the high-bandwidth domain (w_{ij}^H) or the standard network domain (w_{ij}^N). Concurrently, ATE continuously monitors the cumulative traffic volume T_i^H and T_i^N for each source i, representing the total inflight traffic directed to the high-bandwidth and network domains, respectively. If $T_i^H \geq pB_H$, indicating potential congestion in the high-bandwidth domain, the algorithm dynamically reroute the overflowing traffic to the network domain. This iterative process ensures that traffic is adaptively distributed based on real-time conditions, preventing oversubscription of high-bandwidth resources while maximizing their utilization. The algorithm terminates when all traffic demands are allocated, returning the optimized proportions w_{ij}^H and w_{ij}^N that minimize congestion across both domains.

ATE introduces a proactive yet simple mechanism to balance traffic across asymmetric bandwidth domains, avoiding the inefficiencies of static routing approaches. By dynamically

adjusting routing decisions based on real-time traffic volumes and a configurable threshold p, the algorithm achieves a near-optimal tradeoff between utilizing high-bandwidth domains and preventing their overutilization, making it particularly effective for NDC networks with heterogeneous bandwidth domains. The selection of p can be determined based on the hardware performance degradation attributes of the actual high-bandwidth domain, and it is generally set to 0.9.

ATE only acts on the first-hop selection of traffic and does not require precise measurement; instead, it only relies on rough planning based on the threshold of the queue, seamlessly integrating with PortFC's original per-port flow control strategy.

F. Mathematical Analysis

HoLB-Free Guarantee: In an $NDC(n,k)$ topology, $n+1$ queues at the switch and $(k+1)(n-1)+2$ queues at the node are sufficient to eliminate HoLB.

HoLB occurs when traffic destined for a non-congested port is blocked by congested traffic sharing the same queue. In PortFC, for a switch with n ports:We allocate $n-1$ Inter-Hop Forwarding Queues (IHFQ) to store traffic destined for the $n-1$ possible egress ports of the next-hop switch.We allocate 1 Destination-Direct Queue (DDQ) for traffic terminating at the next-hop node. Let Q_{dir} be the set of egress directions. Since each direction $d \in Q_{dir}$ is mapped to a unique physical queue q_d, a pause signal S_d acting on q_d has no impact on $q_{d'}$ where $d \neq d'$. Thus, the topology-aware isolation ensures that no victim flow is delayed by a congested flow destined for a different path.

Deadlock-Free Guarantee: The specific queue segmentation at nodes effectively breaks all CBD.

PortFC categorizes node traffic into three distinct types: (a) four-hop traffic, (b) two-hop traffic, and (c) relay traffic. As defined in Equation 1 and 2. This segmentation ensures that congestion signals propagate at most three hops and strictly terminate. Because a flow always moves through a sequence of queues with decreasing hop-potential, no cyclic dependency can form, thus preventing deadlock.

Mathematical Analysis of ATE: To justify the fairness and efficiency of the Asymmetric Traffic Engineering (ATE) mechanism, we provide a formal analysis of its bandwidth allocation logic.

Proportional Fairness in ATE: Consider a node i with two potential egress domains: the High-Bandwidth Domain (HBD) with capacity B_H, and the Network Domain (ND) with capacity B_N. Let T_i^H be the real-time inflight traffic in HBD. ATE aims to achieve proportional fairness such that the traffic distribution ratio $\gamma = \frac{Flow_{HBD}}{Flow_{ND}}$ is positively correlated with the physical capacity ratio $\frac{B_H}{B_N}$.

Convergence of Dynamic Load Balancing: Given a utilization threshold $p \in (0,1)$, the iterative weight adjustment in Algorithm 5 converges to a stable equilibrium that maximizes HBD utilization while preventing ND saturation.

The routing weight for HBD at time step $t+1$, denoted as $w_{ij}^H(t+1)$, is updated based on the feedback of inflight traffic $T_i^H(t)$:

$$w_{ij}^H(t+1) = w_{ij}^H(t) \cdot \frac{p \cdot B_H}{T_i^H(t)} \tag{3}$$

When the system reaches an equilibrium state where $w_{ij}^H(t+1) = w_{ij}^H(t)$, it implies that $T_i^H(t) = p \cdot B_H$. If the $T_i^H(t)$ in the denominator momentarily drops to zero (e.g., under sparse MoE traffic), the update would be ill-defined. In our implementation, when $(T_i^H(t) = 0)$, we skip the update and keep the previous weight.

If the total traffic demand D exceeds the effective capacity of HBD ($D > p \cdot B_H$), the overflow traffic $\Delta T = D - p \cdot B_H$ is automatically rerouted to the ND via the weight $w_{ij}^N = 1 - w_{ij}^H$. This ensures that:

1) HBD is utilized to its maximum efficient throughput ($p \cdot B_H$) to leverage high-speed interconnects.
2) ND acts as a secondary buffer for bursty traffic, preventing the hardware degradation typically associated with oversubscribing heterogeneous links.

Thus, the bandwidth is allocated proportionally to the domain's real-time capability, ensuring that no domain becomes a premature bottleneck.

The threshold p (typically set to 0.9) acts as a safety margin (headroom) to absorb transient bursts. This closed-loop feedback mechanism transforms ATE from a simple heuristic into a robust control system, ensuring stable performance in rail-only and other heterogeneous NDC topologies.

IV. Implementation

We implemented the PortFC prototype on Tofino2, a state-of-the-art programmable switch [11] ASIC with Reconfigurable Match Table (RMT) architecture. A packet in Tofino2 first traverses the ingress pipeline, followed by the traffic manager (TM) and finally the egress pipeline. Each pipeline has multiple stages, each capable of doing stateful packet operations. Ingress/egress ports are statically assigned to pipelines. In this section, we briefly describe the key modules of the prototype.

A. Switch Queue Management

Queue Assignment Manager: An arriving packet first undergoes a standard processing procedure, including parsing and control. Henceforth, the packet enters the queue assignment manager. Our queue assignment manager assigns queues to different traffic according to the destination of flows, which have been parsed during the ingress pipeline. In detail, we leverage the address characteristics of NDC rather than relying on static forwarding tables for routing. For packets destined for hosts within the same node, they can be easily identified by the same subnet, while for packets destined for hosts outside the node, we determine the corresponding queue number by calculating the IP address modulo the node size. By utilizing NDC's address properties, our approach both helps to save switch storage space and avoids potential single-point failures that may arise from fixed routing tables.

Queue Depth Gatherer: We need queue depth information in the ingress pipeline for pausing and resuming. With an inbuilt feature *Ghost Thread* tailored for this task in Tofino2, the traffic manager can communicate the queue depth information for all the queues in the switch to all the ingress pipelines without consuming any additional ingress cycles or bandwidth.

Signal Packet Emitter: When the queue depth triggers the Pause or Resume threshold, the signal packet emitter leverages the packet trigger functionality to construct signal packets, such as Pause and Resume. Upon receiving a Pause or Resume, the module engages Tofino2's AFC (Advanced Flow Control) mechanism to pause (or resume) the queue.

B. Node Queue Management

We also implemented the PortFC prototype on DPDK, a set of libraries and drivers for fast packet processing. With the

zero-copy ability of DPDK, we can implement extremely fast user-defined data plain forwarding program. In this section, we briefly depict the node-side prototype of PortFC.

According to the DPDK [12] documentation, the maximum queue number under optimal NIC performance for each port is 32, while our design only requires 4 queues on each port. For queue assigner, unlike the switch configuration mentioned earlier, we classify traffic based on both its source and destination on the node side. For the signal packet responder, we pause or resume the appropriate queue according to the contents of the signal packet.

C. Hardware Justification

Queue Resources: For a typical $NDC(n,k)$, the number of required queues per port is only $(k+1)\times(n-1)+2$. For an 8-port switch, this is 16 queues, which is well within the capability of modern RDMA NICs and programmable switches.

We have detailed the hardware implementation using DPDK and programmable switch pipelines, showing that the "Egress-detecting" logic only requires a few stages in the ingress/egress pipeline without taxing the switch's memory footprint.

V. Evaluation

In this section, we conduct NS3 simulations to evaluate the performance of PortFC and answer the following questions:

1) How effective is the PortFC? We compare PortFC's performance to state-of-the-art lossless PFC* and the lossy IRN, besides traditional PFC and Go-Back-N algorithms.

2) What are the reasons for the excellent performance of PortFC? We investigate the reasons through the evaluation of various metrics in different scenarios.

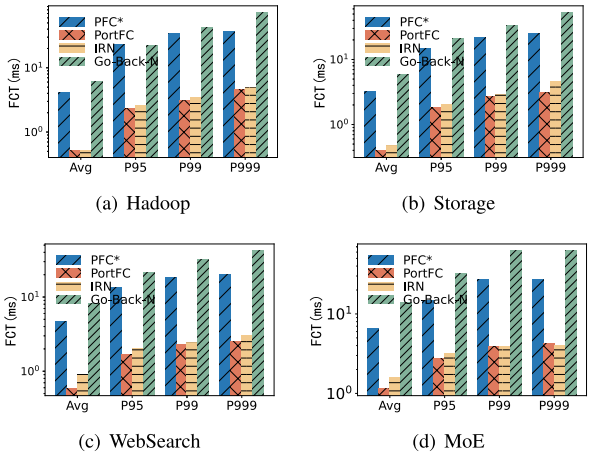
3) How is the scalability of PortFC? We validate the scalability of PortFC through experiments with different scales and various traffic patterns.

A. Experimental Setup

Network Topology: We select $NDC(4,1)$ and $NDC(8,1)$ as the simulation experiment topologies to evaluate the performance of PortFC. $NDC(4,1)$ consists of 16 nodes and 8 switches. Its structure is shown in Figure 6 specifically. Additionally, we also conduct experiments on the larger $NDC(8,1)$ topology, which consists of 64 nodes and 16 switches, to further evaluate the scalability and performance of PortFC. The link bandwidth in the aforementioned topologies is set to 100Gbps, the link delay to 1us, and the switch buffer size to 5MB.

Congestion Control: We use DCQCN [43] as the congestion control scheme for all of our experiments. We set K_{min} = 100KB, K_{max} = 400KB and P_{max} = 0.2 based on the previous works [20], [38]. For the rest of the parameters, we follow the recommendations in the Mellanox firmware [44].

Traffic Patterns: We evaluate the performance of PortFC on three common traffic patterns (WebSearch, Hadoop, and Storage [20], [45]). The three common traffic patterns have distinct flow size distributions. We use the three traffic patterns to generate a traffic workload with Poisson arrival distribution. Additionally, we generate incast traffic by using 8 nodes in $NDC(4,1)$ and 32 nodes in $NDC(8,1)$ to send to one node respectively, with each flow fixed at 1MB in size. The incast traffic is combined with the traffic generated from the three common traffic patterns to increase network congestion and more realistically simulate data center network traffic.



(a) Hadoop (b) Storage

(c) WebSearch (d) MoE

Fig. 8. Flow completion time (FCT) under $NDC(4,1)$.

Baselines: In **PortFC**, congestion onset and relief thresholds are set for the two types of queues (§III-B.1) at the switch's egress ports. We compare **PortFC** with **PFC, PFC*, IRN and Go-Back-N** algorithms. For example, **PortFC** uses thresholds proportional to the BDP, with $X_{\text{OFF}} = 6 \cdot \text{BDP}$ and $X_{\text{ON}} = 4 \cdot \text{BDP}$, while **PFC/PFC*** adopts the same X_{OFF} and X_{ON} values. The setting is chosen partly to align with prior work and partly to account for multi-hop effects; here, the delay D in BDP refers to the single-hop latency. NDC architectures involve complex relay paths. The 6 BDP setting provides a necessary safety margin (headroom) to account for the propagation delay of Pause frames across multi-hop relay nodes and the processing latency of hardware.

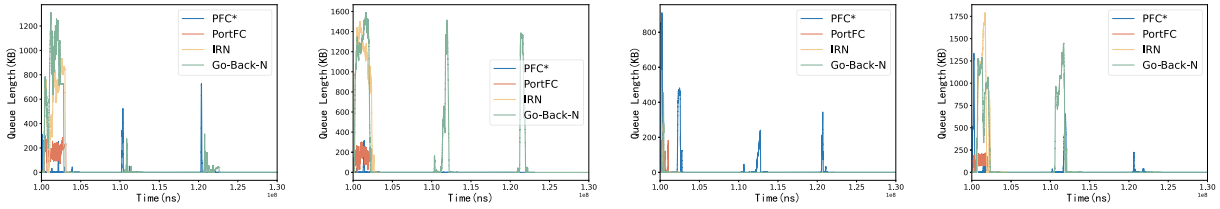
IRN aligns its parameters with prior work [10], setting $Rto_{\text{high}} = 320\mu s$, $Rto_{\text{low}} = 100\mu s$, and $RtoThreshold = 3$. The Rto of **Go-Back-N** is set to 10ms. Note that if we use traditional PFC in BCube, its deadlock will cause the FCT to become very large, and the throughput even degrades to zero within the timeout period. Therefore, the results are not shown in some experimental figures for better visual clarity.

In **PFC***, drawing on the queue cutover strategy from Tagger [35], we enhance PFC to avoid partial deadlocks, segregating traffic into distinct queues at each hop. In fact, this approach may require too many lossless priorities to eliminate all deadlocks [33]. In **IRN**, we align the settings of the three parameters with those in IRN [10], which $RtoThreshold$, Rto_{low} and Rto_{high} are set to 3, 100us, 320us, respectively. The Rto of Go-Back-N is set to 10ms.

B. Various Data Center Traffic Evaluation

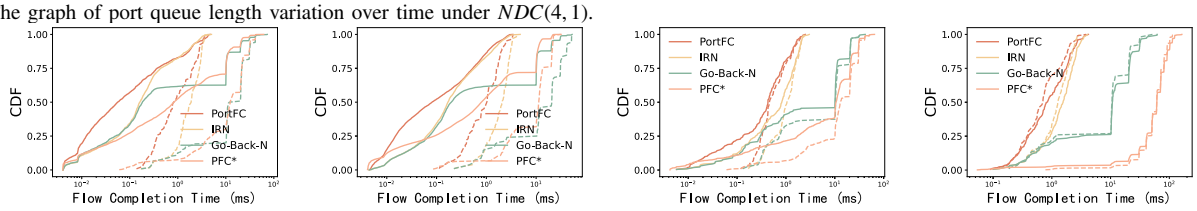
PortFC has low latency and good FCT. Figure 8 shows the FCT on three traffic patterns. We draw two key conclusions. First, PortFC achieves the lowest average and tail FCT across all three patterns. Specifically, PortFC reduces the average and tail FCT by 11.7%-69.2% and 16.9%-41.3% respectively, compared with the second-best method, i.e. IRN. Those figures are much larger compared with PFC*, i.e. 49.9%-87.3% and 53.3%-87.7%. Second, the average FCT reduction achieved by PortFC is more highlighted with a larger proportion of long flows. Storage and WebSearch contain more long flows, consequently, the average FCT reduction achieved by PortFC over IRN is increased from the 11.7% on Hadoop to 40.9% on Storage and 69.2% on WebSearch, respectively.

We also separate the impact of incast traffic, as shown in the dashed lines of Figure 10. Even though the performance of



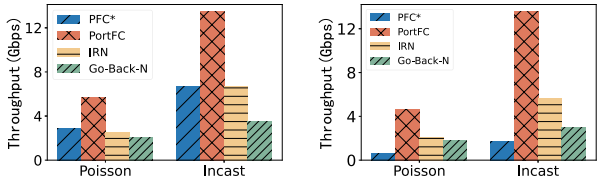
(a) Hadoop (b) Storage (c) WebSearch (d) MoE

Fig. 9. The graph of port queue length variation over time under $NDC(4,1)$.

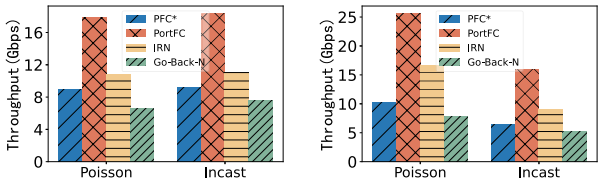


(a) Hadoop (b) Storage (c) WebSearch (d) MoE

Fig. 10. The CDF of FCT under $NDC(4,1)$. The dashed lines represent the incast traffic pattern.

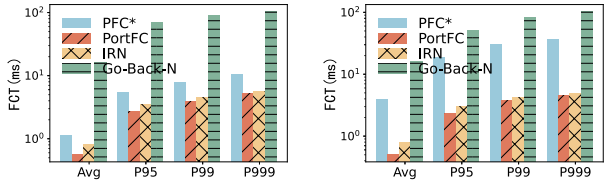


(a) Hadoop (b) Storage

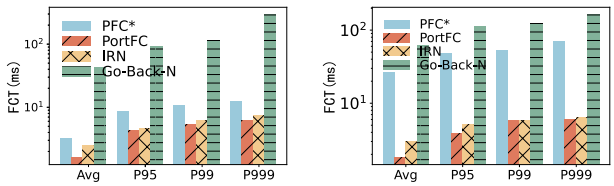


(c) WebSearch (d) MoE

Fig. 11. Average throughput under $NDC(4,1)$.



(a) Hadoop (b) Storage



(c) WebSearch (d) MoE

Fig. 12. FCT under $NDC(8,1)$.

IRN is second only to that of the best PortFC, its completion time for small flows is not ideal because the timeout threshold setting cannot simultaneously satisfy both low latency and high throughput. Although PFC* leverages multiple queues to eliminate deadlocks, its performance remains poor due to the impact of HoLB in BCube.

PortFC maintains a low queue length. In Figure 9, PortFC maintains the shortest queue length at the switch ports compared with other methods, which not only reduces the queuing time of packets in the network but also prevents packet loss due to buffer overflow. It is evident that without per-port flow control algorithm, the queue length at the switch can reach MB levels, leading to packet loss in the network.

PortFC has high throughput performance. Figure 11 shows that, in Storage, PFC* frequently encounters HoLB issues due to the more long flows, and the average throughput of PortFC outperforms that of PFC* by about 8 times. PortFC outperforms PFC* by about 2 times in other scenarios. Compared to IRN and Go-Back-N, PortFC outperforms them by 1.7-2.3 times and 2.4-3.9 times, respectively.

C. Large-Scale Simulation for Scalability Evaluation

We also conducted larger-scale experiments to verify the scalability of PortFC. The $NDC(8,1)$ is composed of 64 servers and 16 switches.

In the $NDC(8,1)$ scenario, incast traffic involves 32 servers simultaneously sending data to one server, which is a higher degree of incast compared to the $NDC(4,1)$ topology, where 8 servers send data to one server. Figure 12 shows that PortFC significantly reduces FCT compared to the PFC*, IRN and Go-Back-N algorithms. Compared to Go-Back-N and IRN for lossy RDMA, PortFC reduces the average FCT by at most 96.5% and 41.7% respectively, and reduces the 99.9th percentile tail latency by at most 97.9% and 16.6% respectively. Compared to PFC* for lossless RDMA, PortFC reduces the average FCT and 99.9th percentile tail latency by at most 19.8% and 85.5% respectively.

In Figure 12, observing the tail latency of the Go-Back-N algorithm under the three traffic patterns, it can be found that in the WebSearch scenario, the 99.9th percentile FCT is significantly higher than the 99th percentile and 95th percentile FCT compared to the other two scenarios. This is because the Poisson traffic generated by the WebSearch traffic contains more long flows, exacerbating network congestion, leading to more packet loss, and extending the 99.9th percentile FCT.

Furthermore, PortFC demonstrates more stable performance across different traffic scenarios. As shown in Figure 12(c), the tail latency difference between the Go-Back-N algorithm and IRN and PortFC is much larger in the WebSearch traffic model

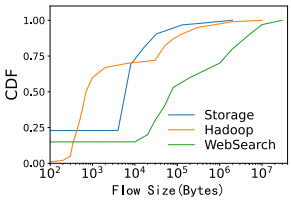
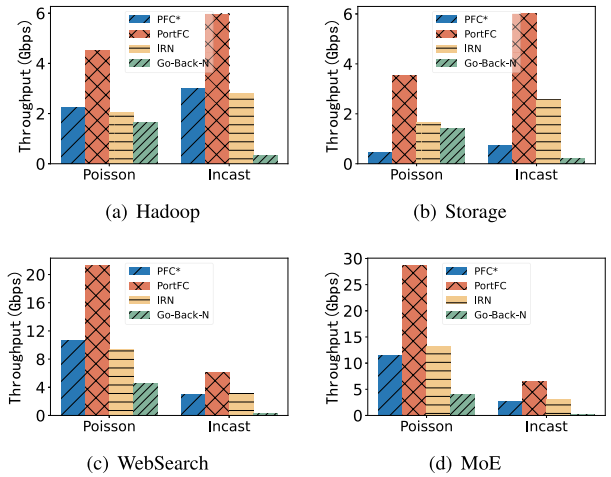


Fig. 13. The CDF graph of traffic patterns.



(a) Hadoop (b) Storage

(c) WebSearch (d) MoE

Fig. 14. Average throughput under $NDC(8,1)$.

compared to the Hadoop and Storage traffic patterns. This is because WebSearch traffic contains more MB-sized long flows than the other two patterns (as referenced in Figure 13), indicating that longer Poisson traffic results in more severe network congestion and greater damage to tail latency. As a result, applying the PortFC in the Bcube network for flow control can keep the FCT consistent across the three traffic model scenarios.

Due to the WebSearch traffic containing more long flows compared to Hadoop and Storage, it can be observed that the Poisson traffic generated by WebSearch achieves higher throughput when competing with incast traffic. This can be corroborated by comparing the average throughput of Poisson traffic in Figure 14(c), Figure 14(a), and Figure 14(b).

Figure 14 also illustrates that, across the three traffic scenarios, PortFC significantly improves the average throughput of traffic compared to PFC*, IRN and Go-Back-N designs. Taking the WebSearch scenario in Figure 14(c) as an example, compared to Go-Back-N and IRN, PortFC increases the average throughput under Poisson traffic by 4.7 times and 2.3 times, respectively, and increases it under incast traffic by 21.6 times and 1.9 times, respectively. Compared to PFC*, PortFC increases the average throughput by 2.0 times and 2.2 times under Poisson traffic and incast traffic, respectively.

D. Parameter Server Traffic Evaluation

The process of parameter synchronization is divided into two stages. In the first stage, all computing servers push their locally computed parameters to the parameter server, generating many-to-one incast traffic. In the second stage, after the parameter server receives all the parameters from the computing servers and performs local parameter updates, it sends the updated parameters to all the computing servers. This stage will generate one-to-many broadcast traffic from the parameter server to all the computing servers.

The network pressure is low during the second stage. This experiment analyzes the time consumption of the two stages of parameter updates in the parameter server scenarios. Figure 15(a) presents the communication time for each stage, with different flow sizes in $NDC(4,1)$. The dark bars in the figure represent the time when the parameter server receives the last incast flow in the first stage, while the light bars represent the completion time of the broadcast traffic triggered in the second stage of the parameter update. The total height of the bars represents the overall time taken for a complete parameter update. The logarithmic scale makes it hard to compare broadcast cost in different algorithms, so in Figure 15(b), the time consumption for the broadcast stage (light bars in Figure 15(a)) is described separately, showing that there is no performance difference among the three algorithms in the broadcast communication mode.

PortFC has good communication time performance. The performance differences among the three algorithms are mainly caused by the communication time in the first stage. According to Figure 15(a), PortFC gets the highest performance with each flow size. Specifically, PortFC reduces the time consumption of the first stage by at most 12.6% and 79.2% compared to IRN for lossy RDMA and PFC* for lossless RDMA respectively. IRN suffers from more tail latency caused by packet loss and increases the time consumption of the first stage. The PortFC, by maintaining the switch queue lengths within a stable range during congestion, ensures lossless links, and finally reaches the best performance.

Furthermore, Figure 16(a) presents the FCT performance of the three algorithms when the flow size is 8MB. PortFC algorithm outperforms the other algorithms in terms of both average FCT and tail latency. This further illustrates PortFC can effectively handle potential network congestion during distributed training parameter updates.

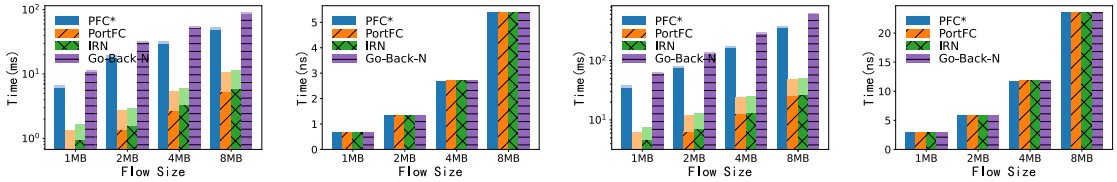
The evaluation trend of larger-scale $NDC(8,1)$ is consistent with the aforementioned one.

E. Evaluation Under Rail-Only Topology of NDC

The Rail-only topology is a special type of NDC architecture that treats GPUs as nodes and incorporates two different types of bandwidth switches (heterogeneous bandwidth). It is commonly used for serving Large Language Model (LLM) tasks. Therefore, besides poisson and incast traffic, we evaluate the traffic patterns of the most popular Mixture of Experts (MoE) architectures in current LLM implementations.

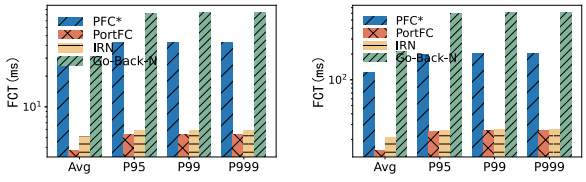
The traffic pattern of MoE models are significantly different from traditional dense models. In MoE, the communication workflow can be broken down into two stages. In the first stage, each token is routed to a small subset of experts based on a gating function. Since expert selection is typically sparse (e.g., top-1 or top-2 routing), only a fraction of the experts are activated for each token. This stage introduces many-to-few traffic patterns, resulting in highly skewed communication, where a small number of experts may receive a disproportionately large volume of data. In the second stage, the outputs from experts are collected and merged back to the original token order, forming a few-to-many traffic pattern. The ability of PortFC to perform well under normal all-to-all traffic pattern also transfers to few-to-many and many-to-few traffic pattern in MoE.

Experimental settings. Our evaluation uses a NDC topology, which falls under the Rail-Only category. Within this



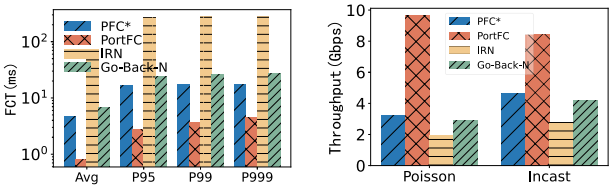
(a) Parameter synchronization in $NDC(4,1)$. (b) Broadcast of the 2nd stage in $NDC(4,1)$. (c) Parameter synchronization in $NDC(8,1)$. (d) Broadcast of the 2nd stage in in $NDC(8,1)$.

Fig. 15. Communication time evaluation of parameter server traffic.



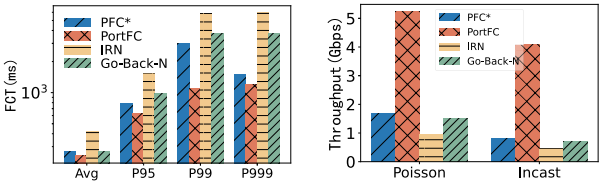
(a) $NDC(4,1)$. (b) $NDC(8,1)$.

Fig. 16. Flow completion time for synchronizing 8MB parameters evaluation of parameter server traffic.



(a) FCT (b) Throughput

Fig. 17. MoE traffic evaluation in $NDC(4,1)$ (Rail-Only topology).



(a) FCT (b) Throughput

Fig. 18. MoE traffic evaluation in $NDC(8,1)$ (Rail-Only topology).

topology, due to policy and regulatory constraints [46], the bandwidth inside the high bandwidth domain (HBD) is 160 GBps [47], while the bandwidth on the rail links is 100 Gbps. The results of rail-only topology are shown in Figure 17 and Figure 18.

Flow-level BDP underestimates capacity in HBD. Since the BDP-FC used by IRN does not account for hop-level variations, the number of in-flight packets is constrained by the slowest path segment. As a result, even when packets are still within the HB domain—where available bandwidth may be higher—IRN throttles the flow too early, leaving significant link bandwidth underutilized. This leads to extended flow completion times, especially for short flows frequently seen in MoE expert communication. In contrast, PortFC, by performing hop-local feedback and controlling queue buildup directly at each switch, achieves better utilization of heterogeneous link bandwidth and avoids premature backpressure.

PortFC adapts better to heterogeneous bandwidth. This difference becomes more prominent in our MoE communication workload, which is composed of a large number of bursty, small-sized flows across hierarchical domains. Figure 17 shows that PortFC achieves maximum throughput and

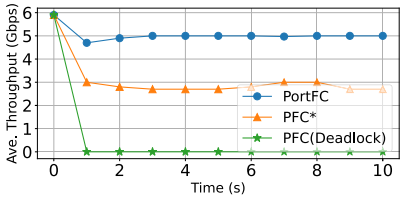


Fig. 19. Average throughput of the testbed experiment.

noticeably lower average and tail latency compared to IRN, with up to 79% reduction in the P99 flow completion time. As shown in Figure 18, the advantage of PortFC remains consistent when the scale is expanded to NDC(8, 1). This demonstrates that PortFC can achieve better scalability, FCT and throughput.

F. Evaluation in a Small-Scale Testbed

Topology: We build a small $NDC(2,1)$ testbed using 2 Tofino2 switches (each switch is virtualized as 2 isolated switches) with 100Gbps port speeds and 4 servers.

Workloads: We evaluate the performance of PortFC, PFC (Deadlock) and PFC* under workloads with background flows (starting at 0s) and loop-producing flows (starting at 1s). We primarily focus on the performance and the effect on background flows victimized by loop-producing flows. Hence, the experimental results only display the average throughput performance of the background flows and do not include the statistical results of the loop-producing flows.

Results: In Figure 19, the traditional PFC mechanism is heavily affected by deadlocks. When loop-producing flows generate CBD and cause a deadlock without any mitigation mechanism, the network throughput of PFC drops to zero. PFC* can relieve part of the deadlocks, but it cannot effectively deal with HoLB issues. Consequently, the average throughput experiences degradation over a long period of time. In contrast, PortFC guarantees deadlock-free operation and effectively eliminates HoLB, increasing the average throughput by about 1.9 times compared with the second-best method.

We measure the hardware resource usage of PortFC on Tofino switches, and results are normalized with the values of switch.p4. As illustrated in Table II, our design consumes only a small amount of resources, leaving ample capacity for other data plane network functions. It is worth noting that switch.p4 is primarily optimized for stateless packet forwarding; consequently, it uses very few RAM blocks and Meter ALUs. This causes the normalized usage percentage to appear disproportionately high, whereas, in reality, our design occupies only a small fraction of the chip's total physical resources.

TABLE I
NOTATIONS USED IN THIS PAPER

n	Each switch with n ports.
k	Each node has $k+1$ ports.
X_{off}	Congestion threshold.
X_{on}	Congestion mitigation threshold.
$Q_{A/B}$	At the switch, traffic is divided into two types: traffic that needs to be forwarded by (A) the next-hop node /(B) traffic for which the next-hop node is the final destination.
$Q_{a/b/c}$	At the node, (a) four-hop traffic originating from the node / (b) two-hop traffic originating from the node / (c) traffic from other nodes that needs to be forwarded by the current node.
D_{ij}	Unallocated traffic demands.
B_H	The capacity of high-bandwidth domain.
B_N	The capacity of network domain.
w_{ij}^H.	The proportion of traffic from source i to destination j that takes the first hop through the high-bandwidth domain.
w_{ij}^N.	The proportion of traffic from source i to destination j that takes the first hop through the network domain.

TABLE II
HARDWARE RESOURCE CONSUMPTION NORMALIZED WITH SWITCH.P4

Resource Category	Match Crossbar	Hash Bits	Meter ALU	Stats ALU	Map RAM
Normalized Usage	0.56%	0.93%	15.3%	0.1%	9.7%
Resource Category	SRAM	VLIW	TCAM		
Normalized Usage	2.2%	0.9%	0.6%		

VI. DISCUSSION

A. Protocol Compatibility and Pause Frame Format

How Pause queue index is encoded. PortFC's Pause message explicitly carries the queue number (and the congested port number), i.e., a pause message containing the queue number and the congested port number is sent upstream. For example, Pause queue number 15 is represented by setting the queue-number field to 15, rather than mapping it into an 8-bit PFC priority mask.

Compatibility. In our design, the switch must be programmable to generate/consume the signal and translate it into queue pause/resume (implemented on Tofino2 via AFC), and the endpoint must have programmable logic to parse and act on the signal. However, in deployments that already use programmable switches (e.g. Tofino) and programmable endpoints (SmartNIC/DPU or host DPDK), PortFC can be deployed as a practical drop-in enhancement by adding this switch-endpoint data-plane logic, without requiring commodity NICs to natively understand new flow-control semantics.

B. Challenges of NDC With High k-Level

The design of this paper primarily targets NDC with $k = 1$. This section mainly discusses the challenges brought by increasing k, and whether $k = 1$ has good scale scalability.

Increased node port requirements. In the NDC architecture, each node requires $k + 1$ ports. As the number of layers k increases, the number of ports needed per node also grows correspondingly. This imposes higher hardware requirements on nodes, potentially necessitating upgrades to existing nodes or the purchase of nodes supporting more ports, which increases hardware costs and deployment complexity.

Increased network latency. Although the NDC architecture itself has a small network diameter, excessive layers k may lead to longer data transmission paths. Data needs to be forwarded through more switches and nodes, causing increased network latency. This affects the performance of real-time applications such as online video and real-time transactions.

Higher routing computation complexity. In NDC networks, routing decisions require complex calculations based on the status of nodes and switches. An increase in layers k leads to more nodes and links in the network, making the network topology more complex and significantly increasing the complexity of routing computations. This not only consumes more computing resources but also may prolong routing convergence time, affecting the network's recovery speed during failures or topology changes.

Increased network management complexity. As the number of layers k grows, the quantity of devices and connection relationships in the network also increases significantly, making network management and maintenance more difficult. Administrators need to handle more tasks such as device configuration, fault troubleshooting, and performance monitoring, increasing the complexity and workload of network management. This imposes higher requirements on the technical capabilities and management skills of network management teams.

Increased costs. A higher number of layers k means requiring more hardware devices such as nodes, switches, and connection cables. Meanwhile, the increased network management complexity demands more human resources for network management and maintenance. Additionally, the operation of more devices consumes more power, increasing the energy consumption costs of data centers.

C. Expanding NDC With OCS

Commercial OCS offers advantages of ultra-low power consumption and reconfigurable logical topology, but it does not have routing capabilities. A more reasonable approach currently is to use it as a reconfigurable patch panel for scaling instead of the switching function [48], so some vendors refer to it as an Optical Cross Connect (OXC) [49]. In the future, topology design and traffic engineering in NDC with OCS will be a promising research topic.

D. Rail-Optimized Network and Queue of GPU

Rail-optimized networks, such as Alibaba's HPN, in fact also connect GPUs to a high-bandwidth intra-host switch inside each host. Thus, hosts themselves possess forwarding capabilities. Compared with the rail-only design in §III-A, the difference is simply that, outside the host, an additional layer of aggregation switches is introduced above the ToR layer. PortFC remains compatible with this topology.

There is no need to worry about the number of GPU queues. This is determined by the capabilities of NVLink/NVSwitch and the RDMA NICs used in the scale-out network connected to the GPUs. Supporting on the order of tens of queues is sufficient [13]. For a typical $NDC(n, k)$ with $k = 1$, each node port only requires $(k + 1) \times (n - 1) + 2$ queues. In an 8-port switch environment ($n = 8$), this amounts to only 16 queues per port, which is well within the 32-queue limit of standard commodity NICs under optimal performance.

E. Deployment Scope and Industrial Relevance

While PortFC demonstrates superior performance in NDCs, its deployment is most effective in topologies featuring a combination of active end-nodes and electrical switches.

Hybrid NDCs: PortFC is highly compatible with rail-optimized architectures like Alibaba HPN [8], where end-nodes bridge scale-up and scale-out domains. By managing

traffic at both the host NIC and the electrical packet switch's egress ports, it solves the intrinsic HoLB and deadlock issues of these heterogeneous networks.

Comparison with Optical/Torus Fabrics: In architectures like Google's TPUv4 [50], which utilize Optical Circuit Switches (OCS) for topology reconfiguration, the switches themselves lack the packet buffers and RMT-based processing stages required for PortFC's egress-detecting flow control. While our queue-based isolation logic could theoretically be implemented within the TPU's internal routing logic, the OCS layer remains transparent to link-layer flow control.

Limitations: PortFC is not designed for proprietary fabrics like Tesla's Dojo [51], which rely on specialized topology and custom transport protocol protocols.

VII. Related Work

Both BCube and high-performance RDMA networks are hot topics. Although no solution efficiently addresses the deadlock and HoLB problems in BCube topologies, previous works offer interesting and meaningful insights.

Flow Control: Some methods are designed based on PFC [40], [52], which have static configurations for the mapping from flows to queues in switches so they are in lack of flexibility. The other methods [53], [54], [55], which dynamically allocates queues for different flows, are much more flexible, but suffers from the relatively low performance.

Facing Deadlock: To avoid deadlock, previous works attempt to avoid the appearance of CBD [9], [33], [34], [56], [57], [58]. They design special routing rules and avoid CBD occurrence, but performance and throughout are both impacted negatively. Other works adopts deadlock recovery [59], [60], but cannot solve the root cause of this issue. The last one, GFC [36], avoids the hold and wait condition by balancing the sending rate and draining rate. However, GFC requires timers and rate limiters, which make its complexity significantly higher. None of these methods considers the characteristics of BCube, and therefore cannot eliminate deadlocks and HoLB simultaneously in BCube.

Facing HoLB: For mitigating the HoLB, the mainstream schemes [30], [61], [62], [63] are all based on buffer management. PLB [61], a PFC-aware load balancer, tries to reroute traffics when sending of switch is paused by PFC to mitigate HoLB. Greedy PFC (G-PFC) [62] adopts the greedy strategy to reduce the appearance of HoLB, which means that when congestion occurs, G-PFC always tries to pause queues with the lowest priorities. BFC [30] uses a few metadata to manage buffers efficiently so that HoLB can be mitigated relatively. However, none of these methods take into account that, in addition to switches, nodes acting as intermediate forwarding nodes can also become HoLB points in BCube.

VIII. Conclusion

RDMA technology used in NDC still faces high retransmission overhead, Head-of-Line Blocking and deadlock problems. Existing solutions for traditional data centers cannot simultaneously address these issues. By implementing a Pause/Resume control signal, per-port queue allocation, a next-hop-based flow control mechanism, and advanced queue scheduling, PortFC eliminates HoLB and deadlocks while ensuring zero retransmissions. Our evaluation shows that PortFC significantly outperforms state-of-the-art lossy and lossless RDMA methods, highlighting its potential to enhance NDC network efficiency and reliability.

References

[1] W. Wang, M. Ghobadi, K. Shakeri, Y. Zhang, and N. Hasani, "Rail-only: A low-cost high-performance network for training LLMs with trillion parameters," 2023, *arXiv:2307.12169*.

[2] M. Lv, J. Fan, W. Fan, and X. Jia, "A high-performantal and server-centric based data center network," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 2, pp. 592–605, Mar. 2023.

[3] B. Karagounis, "Introducing the Microsoft Azure modular datacenter," Microsoft, Redmond, WA, USA, Tech. Rep., 2020.

[4] W. Wang, M. Ghobadi, K. Shakeri, Y. Zhang, and N. Hasani, "Optimized network architectures for large language model training with billions of parameters," Massachusetts Inst. Technol. (MIT), Cambridge, MA, USA, Tech. Rep., 2023. [Online]. Available: https://arxiv.org/abs/2307.12169

[5] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2008, pp. 75–86.

[6] C. Guo et al., "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 63–74.

[7] G. Wang et al., "HS-DCell: A highly scalable DCell-based server-centric topology for data center networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 5, pp. 3808–3823, Oct. 2024.

[8] K. Qian et al., "Alibaba HPN: A data center network for large language model training," in *Proc. ACM SIGCOMM 2024 Conf.*, 2024, pp. 691–706.

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

[10] R. Mittal et al., "Revisiting network support for RDMA," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 313–326.

[11] A. Agrawal and C. Kim, "Intel Tofino2—A 12.9Tbps P4-programmable Ethernet switch," in *Proc. IEEE Hot Chips Symp. (HCS)*, Aug. 2020, pp. 1–32.

[12] L. Foundation, "Data plane development kit (DPDK)," Linux Found., Wilmington, DE, USA, Tech. Rep., 2015.

[13] NVIDIA. (2024). *Nvlink and Nvlink Switch*. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/

[14] A. Li et al., "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 94–110, Jan. 2020.

[15] H. Liao et al., "UB-mesh: A hierarchically localized nD-fullmesh datacenter network architecture," 2025, *arXiv:2503.20377*.

[16] : Ethernet Amendment—400G/800G Physical Layer Specifications, Standard IEEE 802.3df-2024, IEEE Standards Association, 2024. [Online]. Available: https://standards.ieee.org/standard/8023df-2024.html

[17] Cisco Systems. (2025). *Ethernet Speed Transitions for AI Networks: 400 Gbps and 800 Gbps Adoption by 2025*. [Online]. Available: https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2025/pdf/BRKOPT-2699.pdf

[18] 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. 21st USENIX Symp. Networked Syst. Design Implement. (NSDI 24)*, 2024, pp. 651–668.

[19] Y. Su et al., "Hermes: An efficient building block for RDMA incast in datacenters," in *Proc. 9th Int. Conf. Comput. Commun. (ICCC)*, Dec. 2023, pp. 2306–2311.

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

[21] W. Bai et al., "Empowering Azure storage with RDMA," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 49–67.

[22] U. E. Consortium. (2023). *Overview of and Motivation for the Forthcoming Ultra Ethernet Consortium Specification*. [Online]. Available: https://ultraethernet.org/wp-content/uploads/sites/20/2023/10/23.07.12-UEC-1.0-Overview-FINAL-WITH-LOGO.pdf

[23] A. Shpiner et al., "RoCE rocks without PFC: Detailed evaluation," in *Proc. Workshop Kernel-Bypass Netw.*, Aug. 2017, pp. 25–30.

[24] "Infniband architecture specifcation volume 1 release 1.2.1 annex a17: Rocev2," InfiniBand Trade Assoc. (IBTA), Beaverton, OR, USA, Tech. Rep. 1.2.1, 2014.

[25] W.-K. Chung, Y. Li, C.-H. Ke, S.-Y. Hsieh, A. Y. Zomaya, and R. Buyya, "Dynamic parallel flow algorithms with centralized scheduling for load balancing in cloud data center networks," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 1050–1064, Jan. 2023.

[26] D. Guo, "Aggregating uncertain incast transfers in BCube-like data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 934–946, Apr. 2017.

[27] M. Lv, J. Fan, W. Fan, and X. Jia, "Fault diagnosis based on subsystem structures of data center network BCube," *IEEE Trans. Rel.*, vol. 71, no. 2, pp. 963–972, Jun. 2022.
[28] Y. Wang, W. Fan, J. Fan, J. Zhou, and B. Cheng, "Subsystem reliability analysis of data center network BCube," *IEEE Trans. Rel.*, vol. 73, no. 4, pp. 1946–1957, Dec. 2024.
[29] IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks–Amendment: Priority-based Flow Control, Standard 802.1Qbb-2011, Aug. 2011.
[30] 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.
[31] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 370–383.
[32] 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.
[33] S. Zhao, Q. Zhang, P. Cao, X. Zhang, X. Wang, and C. Zhou, "Flattened clos: Designing high-performance deadlock-free expander data center networks using graph contraction," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 663–683.
[34] X. Zhang et al., "FC+: Near-optimal deadlock-free expander data center networks," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. With Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, Dec. 2023, pp. 1–9.
[35] 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.
[36] 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.
[37] K. Liu et al., "Pyrrha: Congestion-root-based flow control to eliminate head-of-line blocking in datacenter," in *Proc. NSDI*, 2025, pp. 379–405.
[38] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan, "Network load balancing with in-network reordering support for rdma," in *Proc. ACM SIGCOMM Conf.*, 2023, pp. 816–831.
[39] P. Goyal et al., "Backpressure flow control," in *Proc. NSDI*, 2022, pp. 779–805.
[40] S. Hu et al., "Tagger: Practical PFC deadlock prevention in data center networks," in *Proc. 13th Int. Conf. Emerg. Netw. EXperiments Technol.*, New York, NY, USA, 2017, pp. 451–463.
[41] *Nvlink High-Speed GPU Interconnect*, NVIDIA, Santa Clara, CA, USA, 2023.
[42] *Nvswitch: Leveraging Nvlink to Maximum Effect*, NVIDIA, Santa Clara, CA, USA, 2018.
[43] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, Sep. 2015.
[44] NVIDIA. (2023). *Mellanox Firmware*. [Online]. Available: https://network.nvidia.com/support/firmware/mlxup-mft
[45] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Sep. 2015.
[46] C. Zhao et al., "Insights into DeepSeek-v3: Scaling challenges and reflections on hardware for AI architectures," in *Proc. ISCA*, 2025, pp. 1731–1745.
[47] A. Liu et al., "DeepSeek-V3 technical report," 2024, *arXiv:2412.19437*.
[48] P. Cao, S. Zhao, M. Y. The, Y. Liu, and X. Wang, "TROD: Evolving from electrical data center to optical data center," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–11.
[49] N. A. Jackman, S. H. Patel, B. P. Mikkelsen, and S. K. Korotky, "Optical cross connects for optical networking," *Bell Labs Tech. J.*, vol. 4, no. 1, pp. 262–281, Jan. 1999.
[50] N. Jouppi et al., "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–14.
[51] E. Quinnell, "Tesla transport protocol over Ethernet (TTPoE)," presented at the Hot Chips, Tesla Inc., 2024. Accessed: Dec. 23, 2025, pp. 1–23.
[52] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless Ethernet," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement.*, 2020, pp. 19–36.
[53] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 239–252.
[54] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 29–42.
[55] W. Li, C. Zeng, J. Hu, and K. Chen, "Towards fine-grained and practical flow control for datacenter networks," in *Proc. IEEE 31st Int. Conf. Netw. Protocols (ICNP)*, Oct. 2023, pp. 1–11.
[56] J. Domke, T. Hoefler, and W. E. Nagel, "Deadlock-free oblivious routing for arbitrary topologies," in *Proc. IEEE Int. Parallel & Distrib. Process. Symp.*, May 2011, pp. 616–627.
[57] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," in *Proc. ACM Conf. Special Interest Group Data Commun.*, New York, NY, USA, 2015, pp. 523–536.
[58] J. C. Sancho, A. Robles, and J. Duato, "An effective methodology to improve the performance of the up*/down* routing algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 8, pp. 740–754, Aug. 2004.
[59] A. Ramrakhyani, P. V. Gratz, and T. Krishna, "Synchronized progress in interconnection networks (SPIN): A new theory for deadlock freedom," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 699–711.
[60] A. Shpiner, E. Zahavi, V. Zdornov, T. Anker, and M. Kadosh, "Unlocking credit loop deadlocks," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 85–91.
[61] 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.*, 2023, pp. 21–28.
[62] Z. Cui and S. Y. Rim, "G-PFC: A packet-priority aware PFC scheme for the datacenter," in *Proc. 21st Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2020, pp. 385–388.
[63] K. Liu et al., "Floodgate: Taming incast in datacenter networks," in *Proc. 17th Int. Conf. Emerg. Netw. Experiments Technol.*, 2021, pp. 30–44.

Peirui Cao is currently a Research Assistant Professor with the School of Computer Science, Nanjing University (NJU). His research interests include reconfigurable data center networks/AI clusters, large-scale network simulator and network bottleneck analysis.

Rui Ning, photograph and biography not available at the time of publication.

Guangyu Zhao, photograph and biography not available at the time of publication.

Zhaochen Zhang, photograph and biography not available at the time of publication.

Chang Liu, photograph and biography not available at the time of publication.

Yunzhuo Liu, photograph and biography not available at the time of publication.

Rui Li, photograph and biography not available at the time of publication.

Chengyuan Huang (Member, IEEE), photograph and biography not available at the time of publication.

Tao Sun, photograph and biography not available at the time of publication.

Zhiqiang Li, photograph and biography not available at the time of publication.

Guihai Chen (Fellow, IEEE), photograph and biography not available at the time of publication.

Baochun Li (Fellow, IEEE), photograph and biography not available at the time of publication.

Chen Tian (Senior Member, IEEE), photograph and biography not available at the time of publication.