RESEARCH-ARTICLE

PortFC: Designing High-performance Deadlock-free BCube Networks

PEIRUI CAO, Nanjing University, Nanjing, Jiangsu, China

RUI NING, Nanjing University, Nanjing, Jiangsu, China

HONGWEI YANG

ZHAOCHEN ZHANG, Nanjing University, Nanjing, Jiangsu, China

CHANG LIU, Nanjing University, Nanjing, Jiangsu, China

RUI LI, Nanjing University, Nanjing, Jiangsu, China

View all

Published: 08 June 2025

ICS '25: 2025 International Conference on Supercomputing
June 8 - 11, 2025
Salt Lake City, USA

Conference Sponsors:
SIGARCH

ICS '25: Proceedings of the 39th ACM International Conference on Supercomputing (June 2025)
https://doi.org/10.1145/3721145.3725749
ISBN: 9798400715372

PortFC: Designing High-performance Deadlock-free BCube Networks

Peirui Cao
Nanjing University
Nanjing, China
caopeirui@nju.edu.cn

Rui Ning
Nanjing University
Nanjing, China
rning@smail.nju.edu.cn

Hongwei Yang
China Mobile
Beijing, China
yanghongwei@chinamobile.com

Zhaochen Zhang
Nanjing University
Nanjing, China
zhaochenzhang@smail.nju.edu.cn

Chang Liu
Nanjing University
Nanjing, China
liuchang_1307@168.com

Rui Li
Nanjing University
Nanjing, China
kevin_rui_li@outlook.com

Yongqi Yang
Nanjing University
Nanjing, China
yyq15280@gmail.com

Yunzhuo Liu
Nanjing University
Nanjing, China
445126256@qq.com

Chengyuan Huang
Nanjing University
Nanjing, China
huangchengyuan@nju.edu.cn

Tao Sun
China Mobile
Beijing, China
suntao@chinamobile.com

Xiaodong Duan
China Mobile
Beijing, China
duanxiaodong@chinamobile.com

Guihai Chen
Nanjing University
Nanjing, China
gchen@nju.edu.cn

Chen Tian
Nanjing University
Nanjing, China
tianchen@nju.edu.cn

Abstract

BCube is a modular data center network. Compared with other topologies, BCube has natural advantages, such as lower deployment costs and stronger failure recovery capabilities. However, RDMA technology used in BCube 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 BCube. In this paper, we propose a per-port flow control named PortFC for BCube. 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.

CCS Concepts

• **Networks** → *Link-layer protocols.*

Keywords

Data Center Networks, Flow Control, Modular Data Center

ACM Reference Format:
Peirui Cao, Rui Ning, Hongwei Yang, Zhaochen Zhang, Chang Liu, Rui Li, Yongqi Yang, Yunzhuo Liu, Chengyuan Huang, Tao Sun, Xiaodong Duan, Guihai Chen, and Chen Tian. 2025. PortFC: Designing High-performance Deadlock-free BCube Networks. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3721145.3725749

1 Introduction

BCube [19, 21, 27, 32, 44, 45] as a representative modular data center network has gained significant attention recently due to their high flexibility, low construction costs, and strong resilience to handle network node and link failures. Constructing high-performance and stable networks on such architectures has become a focal point for many researchers and service providers.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725749

Corresponding authors: Zhaochen Zhang(zhaochenzhang@smail.nju.edu.cn) and Tao Sun(suntao@chinamobile.com).

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 [20]. 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 [33]. 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, servers also participate in relay forwarding, increasing the complexity of the network paths. Additionally, switch queues are influenced by the next two-hop switches and servers, 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 (§3.2) to match the BCube topology (§3.1) and egress-detecting per-port flow control method (§3.3) to solve the HoLB problems, PortFC guarantees low latency and high throughput. Finally, PortFC proposes a deadlock-free strategy (§3.4) that utilizes queue scheduling combined with the server-aware feature that servers can also act as forwarding nodes to eliminate potential deadlock loops in BCube. We implement PortFC (§4) on Tofino2 switch [4] and DPDK [15], and our evaluation (§5) 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.

Ethics Statement: This work does not raise any ethical issues.

2 Background and Motivation

2.1 Bring BCube into Focus

The network topology is one of the key factors affecting data center network performance, playing a significant role in fault recovery capability, ease of scale expansion, communication bandwidth, and network latency within data center networks. A good network topology architecture should possess high scalability, efficient utilization of switches and servers, and high fault tolerance. Research related to BCube [19, 21, 27, 32, 44, 45] has gained significant attention in recent years.

BCube [19] allows servers to participate in forwarding, and can be modularly and recursively expanded. As illustrated in Figure 2, BCube adopts a hierarchical and modular design where servers not only act as end hosts but also participate in packet forwarding, effectively serving as part of the switching fabric. Firstly, BCube provides well flexibility for data center migration, deployment, and scalability. Secondly, BCube 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, BCube demonstrates greater stability in handling switch and server failures due to its multiple equivalent paths and switch layers, ensuring good transmission performance even during failures.

Nowadays, the bandwidth of data center network [7, 8, 49] links is rapidly advancing; 100Gbps links have been widely deployed, 200Gbps links are gradually being adopted, and the standardization of 400Gbps Ethernet links is in progress [43, 47]. However, the growth rate of network device buffers is far from keeping pace with the increase in link bandwidth [3]. As shown in Figure 1, since 2010,

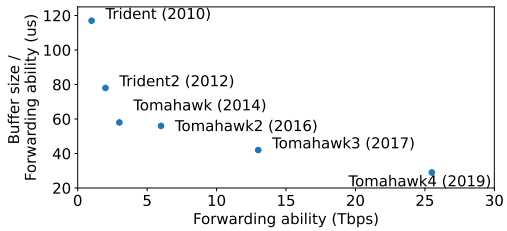


Figure 1: The trends in switch chip architecture.

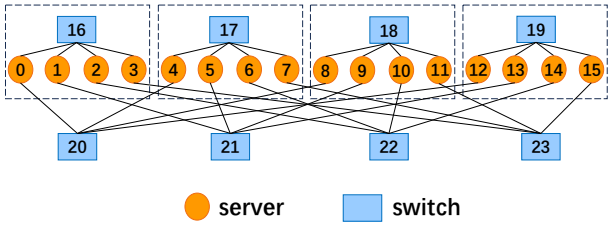


Figure 2: BCube topology example.

the ratio of switch buffer size to its forwarding ability has decreased by 3.6 times [6, 16, 18].

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) 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 [42]. Shallow-buffer switches are also highly susceptible to buffer exhaustion in a short time during large-scale incast traffic, leading to packet loss.

2.2 Deploy RDMA over Ethernet in BCube

2.2.1 BCube Based on RoCEv2 Networks. Large vendors [5, 20, 27, 29] 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 [12], there have been advanced RoCE designs, e.g., Resilient RoCE [39], IRN [33] 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 BCube networks with lossless RoCEv2 (RDMA over Converged Ethernet v2) [2] and conduct extensive experiments to confirm that our design outperforms lossy RDMA designs in BCube networks.

However, current lossless RDMA algorithms are mainly designed for switch-centric tree topologies and are not suitable for BCube. Current research on BCube primarily focuses on traffic scheduling

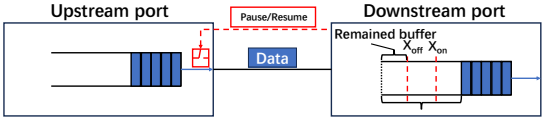
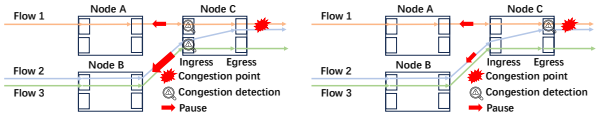


Figure 3: The PFC mechanism.



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

Figure 4: HoLB examples in BCube.

optimization, routing algorithm design [11, 22], fault diagnosis [31, 46] 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.

2.2.2 Head-of-Line Blocking Issues in BCube. PFC [1] 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. 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 BCube is low, the Pause frame can still be transmitted to another switch through relay server nodes.

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 BCube topology features (§3.2) with the penetrate server to synchronize congestion information to multi-hop switches and conduct per-port flow control (§3.3), requiring only a limited number of queues to achieve high performance.

2.2.3 Existing Works Fail to Resolve Deadlock in BCube. Under the Pause/Resume mechanism of PFC, due to the presence of loops in

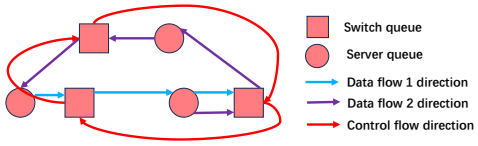


Figure 5: A deadlock example in BCube.

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 BCube 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 [20, 48, 50]. The other is to manage buffers in the queue more properly to avoid deadlock [25]. 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 BCube. Figure 5 illustrates a potential deadlock scenario in a two-dimensional BCube. The figure assumes the presence of two four-hop flows in the network, with the flow control algorithm deployed at the switches, and the servers 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.

2.3 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/server egress-port queue allocations to achieve a high-bandwidth, low-latency, deadlock-free BCube 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 BCube 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 BCube network, servers 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.

3 Design

3.1 BCube Structure

It is necessary to briefly introduce the construction method of the general BCube Data Center network. $BCube(n,k), (k \geq 1)$ is a recursively defined structure, consisting of n $BCube(n,k-1)$ units and n^k switches, each with n ports. Each server in $BCube(n,k)$ 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 servers in $BCube(n,k)$ 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, $BCube(4,0)$ within the dashed box consists of one 4-port switch and four servers, and four $BCube(4,0)$ units are connected through four 4-port layer-one switches to form $BCube(4,1)$. It is noteworthy that servers located in the same position in each $BCube(4,0)$ are connected through the same layer-one switch.

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 BCube per-port flow control algorithm designed in this paper is divided into two types: switch port queue allocation and server port queue allocation.

3.2 Per-port Queue Allocations

3.2.1 Two Types of Queues for Switch Ports. In the BCube, the adjacent nodes of a switch are all servers, 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 server because the next-hop server is not the destination server. The second is traffic reaching the destination server at the next hop.

As shown in the bottom right corner of Figure 6, there are several flows: $F11$, $F12$, $F13$, $F14$ from server 11, $F15$ from server 3 and $F16$ from server 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 server 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.

The queue allocation strategy of the BCube per-port flow control algorithm at the switch port is based on which egress port the

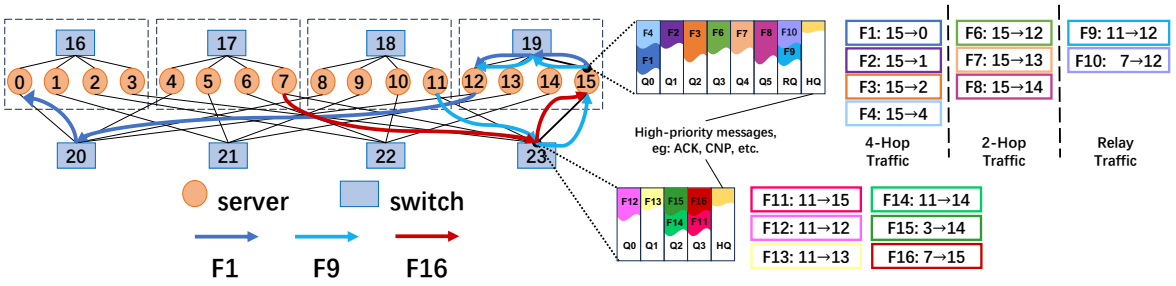


Figure 6: The BCube structure and example of queue allocation on the port of switches and servers.

packet will go through at the next-2-hop switch (since the next hop connected by the BCube switch is a server). 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 BCube per-port flow control uses minimal hardware resources on the switch.

For an $BCube(n,k)$ 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 server 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 (§3.3), each port of a switch in the $BCube(n,k)$ topology only requires $n+1$ queues.

3.2.2 Three Types of Queues for Server Ports. In the BCube topology, there are three types of traffic at the server ports. The first type is the four-hop traffic originating from the server itself. The second type is the two-hop traffic originating from the server itself. The third type is traffic originating from other servers, which needs to be forwarded by the current server. For these three types of traffic, the per-port flow control algorithm provides different queue allocation schemes. In an $BCube(n,k)$ topology, each switch has n ports. The first type of four-hop traffic originating from the server 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 second type of two-hop traffic originating from the server 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 server, 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, server 12, switch 20 and finally reaches server 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. Relay traffic contains $F9$ and $F10$, flowing only via server 15.

For the queue allocation at the port of server 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 server 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 (§3.3), resulting in finer granularity and better control effects. Even so, in an $BCube(n,k)$ topology, the number of port queues on the server is still linearly proportional to the number of switch ports. Specifically, in the $BCube(n,k)$ topology, the traffic originating from the server 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 server, and one high-priority queue. In summary, in an $BCube(n,k)$ topology, a server 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[16, 18, 41], so the queue allocation scheme designed in this paper can still support large BCube networks and has good scalability.

3.3 Per-port Flow Control

In designing the per-port flow control algorithm for BCube, congestion detection occurs at the switch's egress port. Based on §2.2.2 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 BCube network through three parts: congestion detection and mitigation at switch ports, server handling of flow control messages, and switch handling of flow control messages.

3.3.1 Congestion Detection and Mitigation. In §3.2.1, the allocation of two types of queues on the switch is described. Thus, in the per-port flow control algorithm of BCube, two sets of congestion detection and mitigation thresholds are defined. 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. For the second type of queue, which stores traffic where the next hop is the destination server (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 $BCube(n,k)$.

```
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
```

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 first type of queue in $BCube(n,k)$. 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 first type of queue, the lengths of the first type 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 first type of queue receives a packet, the logic for detecting whether congestion has occurred when a packet is received by the second type of queue, 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.

Algorithm 2 Packet transmission operation at the switch egress port in $BCube(n,k)$.

```
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 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 first type of queue. 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 first type of queue, lines 16 to 24 of Algorithm 2 provide the logic for handling congestion mitigation for the second type of queue 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.

3.3.2 Handling Messages at Servers. 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 server. First, the server 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 server receives a flow control message generated by the first type of queue on the switch, it sets the corresponding state for its own first $n-1$ queues based on the type of control message for the queue numbered *congestedQIdx*. Because in the BCube topology, traffic is classified into four-hop traffic and two-hop traffic. When a server receives a control message from the first type of queue on a downstream switch, it means that the four-hop traffic sent by the server may continue to congest the downstream port numbered *congestedPortIdx*, so the local queue corresponding to the congested downstream port should be paused.

If the server receives a flow control message from the second type of queue on the downstream switch, it indicates that the queue on the downstream switch destined for the next-hop server is congested. This traffic may be two-hop traffic originating from the server or four-hop traffic from a further upstream point. Thus, the corresponding queue numbered *congestedPortIdx* in the second type of queue on the server (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 server 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 server does not forward this pause frame, the upstream switch may keep sending packets to the server, resulting in the accumulation of packets on the server, causing more queuing delay and thus affecting the throughput rate.

Observing the server'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 first type of queue on the switch (traffic that needs to be forwarded by the next-hop server) will only pause the corresponding queue on the upstream server of that switch. In contrast, congestion caused by traffic in the second type of queue on the switch (queues storing traffic for which the next-hop server is the destination server) will pause both the corresponding queue on the upstream server and the corresponding queue on the upstream switch. Since new congestion is not detected at the server, in the flow control algorithm designed in this paper, congestion will at most propagate three hops in the network.

3.3.3 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.

3.4 Deadlock-free Strategy

The example of deadlock in BCube, as Figure 5 shows, is basically discussed in §2.2.3. This paper analyzes the root cause of the BCube deadlock scenario: Not only switch-forwarded traffic, but also server-forwarded traffic in BCube, can form CBD loops, causing deadlocks. Unlike traditional PFC, where all traffic at both switches and servers 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 [26], 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 server and traffic for which the next-hop server 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 server, the traffic types are slightly more complex and divided into three categories: four-hop traffic originating from the server, two-hop traffic originating from the server, and traffic from other servers that needs to be forwarded by the current server. These three types

Algorithm 3 Handling flow control messages received by the switch in $BCube(n,k)$.

```
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
```

Algorithm 4 Handling flow control messages received by the switch in $BCube(n,k)$.

```
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
```

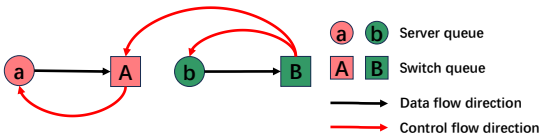


Figure 7: Avoiding deadlocks in BCube.

of traffic are placed into three different queues, denoted as a, b, and c, respectively.

$$switch_i Q_B \geq \hat{X}_{off} \Longrightarrow \begin{matrix} Pause(server_{i-1} Q_b) \\ Pause(switch_{i-2} Q_A) \end{matrix} \quad (1)$$

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

The congestion propagation path after detecting congestion at the switch egress port is briefly illustrated in Figure 7. As described in §3.3.2, when congestion is detected in an A type switch queue, the signal will be propagated to an a type queue in previous-hop server. 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 server 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 server 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 server. 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 server 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 server, 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.

4 Implementation

We implemented the PortFC prototype on Tofino2, a state-of-the-art programmable switch [4] 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.

4.1 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 BCube 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 BCube'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.

4.2 Server 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 server-side prototype of PortFC.

According to the DPDK[15] 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 server side. For the signal packet responder, we pause or resume the appropriate queue according to the contents of the signal packet.

5 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.

5.1 Experimental Setup

Network Topology: We select $BCube(4,1)$ and $BCube(8,1)$ as the simulation experiment topologies to evaluate the performance of PortFC. $BCube(4,1)$ consists of 16 servers and 8 switches. Its structure is shown in Figure 6 specifically. Additionally, we also conduct experiments on the larger $BCube(8,1)$ topology, which consists of

64 servers 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 [51] 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 [29, 41]. For the rest of the parameters, we follow the recommendations in the Mellanox firmware [34].

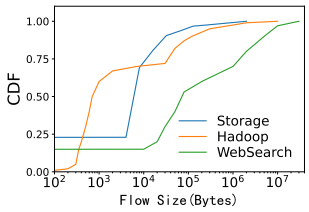


Figure 8: The CDF graph of traffic patterns.

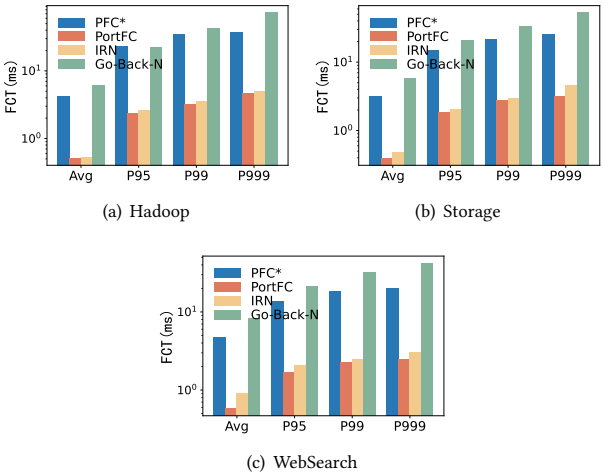


Figure 9: Flow completion time (FCT) under $BCube(4, 1)$.

Traffic Patterns: We evaluate the performance of PortFC on three common traffic patterns (WebSearch, Hadoop, and Storage [29, 37]). The three common traffic patterns have very distinct flow size distributions, as shown in Figure 8. It demonstrates that their traffic distributions are very different, which enhances the generalizability of the experiment. We use the three traffic patterns to generate a traffic workload with Poisson arrival distribution. Additionally, we generate incast traffic by using 8 servers in $BCube(4, 1)$ and 32 servers in $BCube(8, 1)$ to send to one server 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.

Baselines: In **PortFC**, congestion onset and relief thresholds are set for the two types of queues (§3.2.1) 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. **IRN** aligns its parameters with prior work [33], 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 [25], 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 [50]. In **IRN**, we align the settings of the three parameters with those in IRN[33], which $RtoThreshold$, Rto_{low} and Rto_{high} are set to 3, 100us, 320us, respectively. The Rto of Go-Back-N is set to 10ms.

5.2 Various Data Center Traffic Evaluation

PortFC has low latency and good FCT. Figure 9 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.

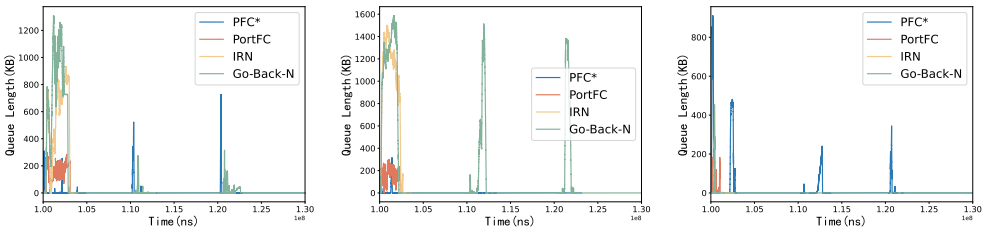
We also separate the impact of incast traffic, as shown in the dashed lines of Figure 11. Even though the performance of IRN is second only to that of the best PortFC, its completion time for small flows is not ideal because the timeout threshold setting cannot simultaneously satisfy both low latency 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 10, 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 12 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.

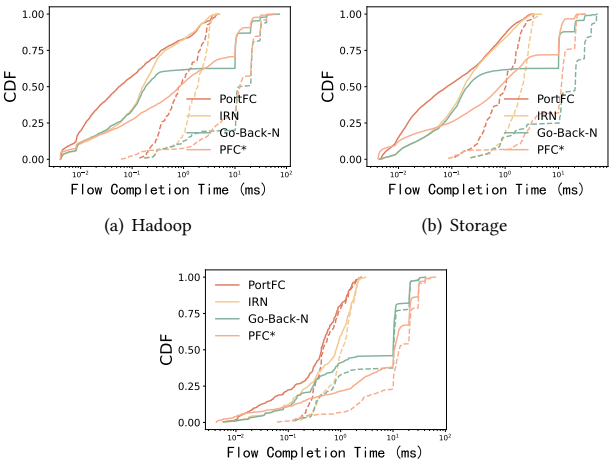
5.3 Large-Scale Simulation for Scalability Evaluation

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



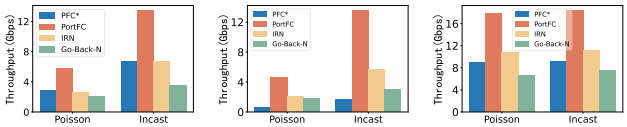
(a) Hadoop (b) Storage (c) WebSearch

Figure 10: The graph of port queue length variation over time under $BCube(4,1)$.



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

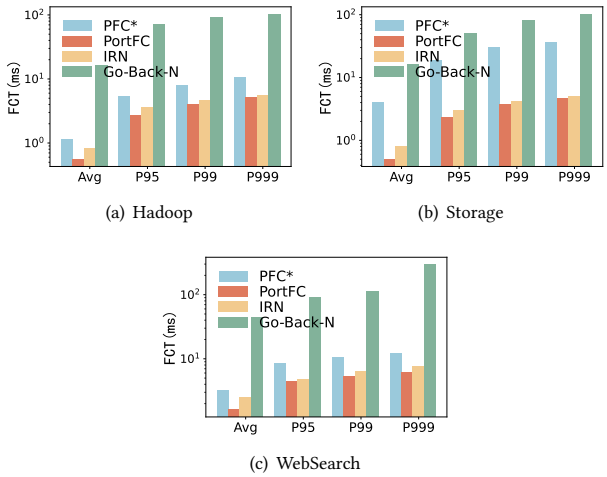
Figure 11: The CDF of FCT under $BCube(4,1)$. The dashed lines represent the incast traffic pattern.



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

Figure 12: Average throughput under $BCube(4,1)$.

In the $Bcube(8,1)$ scenario, incast traffic involves 32 servers simultaneously sending data to one server, which is a higher degree of incast compared to the $Bcube(4,1)$ topology, where 8 servers send data to one server. Figure 13 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.



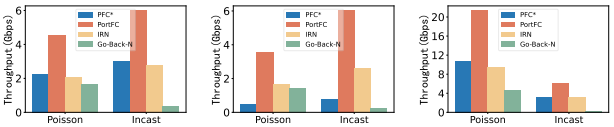
(a) Hadoop (b) Storage (c) WebSearch

Figure 13: FCT under $Bcube(8,1)$.

In Figure 13, 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 13(c), the tail latency difference between the Go-Back-N algorithm and IRN and PortFC is much larger in the WebSearch traffic model 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 8), 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



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

Figure 14: Average throughput under $Bcube(8, 1)$**.**

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.

6 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 [9, 26], which have static configurations for the mapping from flows to queues in switches so they are in lack of flexibility. The other methods [10, 23, 28], 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 [14, 20, 38, 48, 50, 52]. They design special routing rules and avoid CBD occurrence, but performance and throughout are both impacted negatively. Other works adopts deadlock recovery [36, 40], but cannot solve the root cause of this issue. The last one, GFC [35], 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 [13, 17, 24, 30] are all based on buffer management. PLB [24], a PFC-aware load balancer, tries to reroute traffics when sending of switch is paused by PFC to mitigate HoLB. Greedy PFC (G-PFC) [13] 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 [17] 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, servers acting as intermediate forwarding nodes can also become HoLB points in BCube.

7 Conclusion

RDMA technology used in BCube 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 BCube network efficiency and reliability.

Acknowledgments

We sincerely thank anonymous reviewers for their helpful comments. This research is supported by the National Key R&D Program of China (2022YFB2702800), the National Natural Science Foundation of China under Grant Numbers 62325205 and 62172204, the Key Program of Natural Science Foundation of Jiangsu under grant No. BK20243053, the Nanjing University-China Mobile Communications Group Co., Ltd. Joint Institute.

References

[1] 2011. IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks – Amendment: Priority-based Flow Control.
[2] 2014. Infniband Architecture Specifcation Volume 1 Release 1.2.1 Annex A17: RoCEv2. https://cw.infinibandta.org/document/dl/7781
[3] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. 2024. Reverie: Low Pass Filter-Based Switch Buffer Sharing for Datacenters with RDMA and TCP Traffic. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA*.
[4] Anurag Agrawal and Changhoon Kim. 2020. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 1–32.
[5] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. 2023. Empowering azure storage with {RDMA}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 49–67.
[6] BROADCOM. 2019. BCM56990 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series.
[7] Peirui Cao, Shizhen Zhao, Min Yee The, Yunzhuo Liu, and Xinbing Wang. 2021. TROD: Evolving from electrical data center to optical data center. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 1–11.
[8] Peirui Cao, Shizhen Zhao, Dai Zhang, Zhuotao Liu, Mingwei Xu, Min Yee Teh, Yunzhuo Liu, Xinbing Wang, and Chenghu Zhou. 2023. Threshold-based routing-topology co-design for optical data center. *IEEE/ACM Transactions on Networking* 31, 6 (2023), 2870–2885.
[9] Wenxue Cheng, Kun Qian, Wanchun Jiang, Tong Zhang, and Fengyuan Ren. 2020. Re-architecting Congestion Management in Lossless Ethernet. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 19–36. https://www.usenix.org/conference/nsdi20/presentation/cheng
[10] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 239–252. https://doi.org/10.1145/3098822.3098840
[11] Wei-Kang Chung, Yun Li, Chih-Heng Ke, Sun-Yuan Hsieh, Albert Y Zomaya, and Rajkumar Buyya. 2021. Dynamic parallel flow algorithms with centralized scheduling for load balancing in cloud data center networks. *IEEE Transactions on Cloud Computing* 11, 1 (2021), 1050–1064.
[12] Ultra Ethernet Consortium. 2023. *Overview of and Motivation for the Forthcoming Ultra Ethernet Consortium Specification.*
[13] Zhenguo Cui and Steven Y. Rim. 2020. G-PFC: A Packet-Priority Aware PFC Scheme for the Datacenter. In *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 385–388. https://doi.org/10.23919/APNOMS50412.2020.9236778
[14] Jens Domke, Torsten Hoefler, and Wolfgang E. Nagel. 2011. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *2011 IEEE International Parallel &*

Distributed Processing Symposium. 616–627. https://doi.org/10.1109/IPDPS.2011.65

[15] Linux Foundation. 2015. Data Plane Development Kit (DPDK). http://www.dpdk.org

[16] Prateesh Goyal, Preey Shah, Naveen Kr Sharma, Mohammad Alizadeh, and Thomas E Anderson. 2019. Backpressure flow control. In *Proceedings of the 2019 Workshop on Buffer Sizing*. 1–3.

[17] Prateesh Goyal, Preey Shah, Naveen Kr. Sharma, Mohammad Alizadeh, and Thomas E. Anderson. 2020. Backpressure Flow Control. In *Proceedings of the 2019 Workshop on Buffer Sizing* (Palo Alto, CA, USA) *(BS '19)*. Association for Computing Machinery, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.1145/3375235.3375239

[18] Prateesh Goyal, Preey Shah, Naveen Kr Sharma, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E Anderson. 2022. Backpressure flow control. In *NSDI*. 779–805.

[19] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 63–74.

[20] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.

[21] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. 2008. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. 75–86.

[22] Deke Guo. 2016. Aggregating uncertain incast transfers in BCube-like data centers. *IEEE Transactions on Parallel and Distributed Systems* 28, 4 (2016), 934–946.

[23] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[24] Jinbin Hu, Chaoliang Zeng, Zilong Wang, Hong Xu, Jiawei Huang, and Kai Chen. 2023. Load Balancing in PFC-Enabled Datacenter Networks. In *Proceedings of the 6th Asia-Pacific Workshop on Networking* (Fuzhou, China) *(APNet '22)*. Association for Computing Machinery, New York, NY, USA, 21–28. https://doi.org/10.1145/3542637.3542641

[25] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2017. Tagger: Practical PFC deadlock prevention in data center networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 451–463.

[26] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2017. Tagger: Practical PFC Deadlock Prevention in Data Center Networks. https://doi.org/10.1145/3143361.3143382. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies* (Incheon, Republic of Korea) *(CoNEXT '17)*. Association for Computing Machinery, New York, NY, USA, 451–463. https://doi.org/10.1145/3143361.3143382

[27] B Karagounis. 2020. Introducing the Microsoft Azure Modular Datacenter.

[28] Wenxue Li, Chaoliang Zeng, Jinbin Hu, and Kai Chen. 2023. Towards Fine-Grained and Practical Flow Control for Datacenter Networks. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. 1–11. https://doi.org/10.1109/ICNP59255.2023.10355582

[29] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*. 44–58.

[30] Kexin Liu, Chen Tian, Qingyue Wang, Hao Zheng, Peiwen Yu, Wenhao Sun, Yonghui Xu, Ke Meng, Lei Han, Jie Fu, et al. 2021. Floodgate: Taming incast in datacenter networks. In *Proceedings of the 17th International Conference on emerging Networking Experiments and Technologies*. 30–44.

[31] Mengjie Lv, Jianxi Fan, Weibei Fan, and Xiaohua Jia. 2022. Fault diagnosis based on subsystem structures of data center network BCube. *IEEE Transactions on Reliability* 71, 2 (2022), 963–972.

[32] Mengjie Lv, Jianxi Fan, Weibei Fan, and Xiaohua Jia. 2022. A high-performantal and server-centric based data center network. *IEEE Transactions on Network Science and Engineering* 10, 2 (2022), 592–605.

[33] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting network support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 313–326.

[34] NVIDIA. 2023. Mellanox firmware. https://network.nvidia.com/support/firmware/mlxup-mft.

[35] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. 2019. Gentle flow control: avoiding deadlock in lossless networks. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 75–89. https://doi.org/10.1145/3341302.3342065

[36] Aniruddh Ramrakhyani, Paul V. Gratz, and Tushar Krishna. 2018. Synchronized Progress in Interconnection Networks (SPIN): A New Theory for Deadlock Freedom. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 699–711. https://doi.org/10.1109/ISCA.2018.00064

[37] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 123–137.

[38] J.C. Sancho, A. Robles, and J. Duato. 2004. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Transactions on Parallel and Distributed Systems* 15, 8 (2004), 740–754. https://doi.org/10.1109/TPDS.2004.28

[39] Alexander Shpiner, Eitan Zahavi, Omar Dahley, Aviv Barnea, Rotem Damsker, Gennady Yekelis, Michael Zus, Eitan Kuta, and Dean Baram. 2017. RoCE rocks without PFC: Detailed evaluation. In *Proceedings of the Workshop on Kernel-Bypass Networks*. 25–30.

[40] Alex Shpiner, Eitan Zahavi, Vladimir Zdornov, Tal Anker, and Matty Kadosh. 2016. Unlocking Credit Loop Deadlocks. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (Atlanta, GA, USA) *(HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 85–91. https://doi.org/10.1145/3005745.3005768

[41] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network load balancing with in-network reordering support for rdma. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 816–831.

[42] Yuzhen Su, Jiao Zhang, Zirui Wan, Pingping Lin, Yunpeng Zhang, Tian Pan, and Tao Huang. 2023. Hermes: An Efficient Building Block for RDMA Incast in Datacenters. In *2023 9th International Conference on Computer and Communications (ICCC)*. 2306–2311. https://doi.org/10.1109/ICCC59590.2023.10507657

[43] Versa Technology. 2021. 400G Ethernet: It's Here, and It's Huge. http://www.versatek.com/400g-ethernet-its-here-and-its-huge/.

[44] Guijuan Wang, Yazhi Zhang, Jiguo Yu, Meijie Ma, Chunqiang Hu, Jianxi Fan, and Li Zhang. 2024. HS-DCell: A Highly Scalable DCell-Based Server-Centric Topology for Data Center Networks. *IEEE/ACM Transactions on Networking* (2024).

[45] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2023. Rail-only: A Low-Cost High-Performance Network for Training LLMs with Trillion Parameters. *arXiv preprint arXiv:2307.12169v4* (2023).

[46] Yihong Wang, Weibei Fan, Jianxi Fan, Jingya Zhou, and Baolei Cheng. 2024. Subsystem Reliability Analysis of Data Center Network BCube. *IEEE Transactions on Reliability* (2024).

[47] Yifan Yuan, Jinghan Huang, Yan Sun, Tianchen Wang, Jacob Nelson, Dan RK Ports, Yipeng Wang, Ren Wang, Charlie Tai, and Nam Sung Kim. 2023. RAMBDA: RDMA-driven Acceleration Framework for Memory-intensive μs-scale Datacenter Applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 499–515.

[48] Xiao Zhang, Peirui Cao, Yongxi Lyu, Qizhou Zhang, Shizhen Zhao, Xinbing Wang, and Chenghu Zhou. 2023. FC+: Near-optimal Deadlock-free Expander Data Center Networks. In *2023 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 1–9.

[49] Shizhen Zhao, Peirui Cao, and Xinbing Wang. 2021. Understanding the performance guarantee of physical topology design for optical circuit switched data centers. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–24.

[50] Shizhen Zhao, Qizhou Zhang, Peirui Cao, Xiao Zhang, Xinbing Wang, and Chenghu Zhou. 2023. Flattened clos: Designing high-performance deadlock-free expander data center networks using graph contraction. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 663–683.

[51] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.

[52] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484