Orderlock: A New Type of Deadlock and its Implications on High Performance Network Protocol Design

Weihao Jiang[*]
Shanghai Jiao Tong University

Wenli Xiao[*]
Shanghai Jiao Tong University

Yuqing Yang
Shanghai Jiao Tong University

Peirui Cao
Nanjing University

Shizhen Zhao[†]
Shanghai Jiao Tong University

Abstract

In the pursuit of designing high-performance network (HPN) protocols, three critical features for effective transmission have been extensively studied: In-order Delivery, Lossless Transmission, and Out-of-order Capability. However, no practical implementation has successfully achieved all three simultaneously. We identify and prove that the simultaneous realization of these features constitutes a necessary and sufficient condition for a new type of deadlock, which we term **Orderlock**. We demonstrate that operating in an Orderlock-risky network is impractical and conduct a comprehensive exploration and comparison of Orderlock-free protocols, through a case study tuning AI workload performance. From an Orderlock-prevention perspective, our findings provide insights into the requirements for future HPN protocol and hardware designs.

CCS Concepts

• **Networks** → **Network protocol design**; **Data center networks**.

Keywords

Data center networking, Network architecture, Network support for AI and machine learning applications, Network theory and algorithms

ACM Reference Format:
Weihao Jiang, Wenli Xiao, Yuqing Yang, Peirui Cao, and Shizhen Zhao. 2025. Orderlock: A New Type of Deadlock and its Implications on High Performance Network Protocol Design . In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3718958.3750497

1 Introduction

With the proliferation of large language models (LLMs), building AI-ready high-performance networks (HPNs) has become a necessity to address the network bottlenecks in LLM training. On one hand, Ethernet speeds in HPNs have steadily increased from 100 Gbps to 400 Gbps and even 800 Gbps. On the other hand, new network protocols are urgently needed to fully leverage the potential of high-bandwidth networks. Recognizing this, major network vendors, including Microsoft, Meta, and Intel, have formed the Ultra Ethernet Consortium (UEC [14]), aiming to develop a high-performance, full-communication stack architecture to meet the escalating network demands of AI at scale.

New workload paradigms such as AI tasks impose higher requirements on efficient load balancing. Unlike traditional data center network (DCN) workloads, which exhibit an 80% elephant flow and 20% mice flow distribution [3, 4, 50], AI workloads consist of latency-sensitive elephant flows with higher burstiness, more uniform flow sizes, and lower entropy (i.e. fewer and less diverse flows) [20, 56]. DCN-era flow-based load balancing paradigms are ill-suited for these workloads, failing to evenly distribute traffic across routes and becoming a bottleneck to optimal performance.

When designing HPN protocols, three key features for packet delivery are often considered: **$\mathbb{I}$n-order Delivery**, **$\mathbb{L}$ossless Transmission**, and **$\mathbb{O}$ut-of-order Capability**.

$\mathbb{I}$n-order Delivery reorders packets on switches or NICs before they are delivered to the end host. It provides a bitstream abstraction for the upper layer and lifts complex transmission control logic overhead from the host, thereby conserving CPU resources and enhancing performance. Countless applications and transport protocols from TCP to modern RDMA [52] are built upon this feature.

$\mathbb{L}$ossless Transmission prevents packet drops inside the network, avoiding time-consuming retransmissions. This mechanism operates by pausing the sender when the receiver is unable to handle more packets. Implementations such as InfiniBand Credit-Based Link-Layer Flow-Control [46] and IEEE 802.1Qbb Priority-based Flow Control [34] have propelled modern DCNs into the 40GbE+ era along with RDMA [23].

$\mathbb{O}$ut-of-order Capability enables aggressive sub-flow level load balancing by allowing out-of-order transmission over multiple paths, which is especially beneficial for low entropy loads. It leads to more evenly distributed workloads, higher bandwidth utilization, and a shorter latency tail. The significance, along with preliminary support, has already been highlighted by existing studies [14, 17, 25, 49].

A natural idea would be to combine all three features into a single protocol. However, we discovered that simultaneously achieving $\mathbb{I}$, $\mathbb{L}$, and $\mathbb{O}$ can lead to a new type of deadlock, which we term **Orderlock**. In §2, we analyze the necessary and sufficient conditions for Orderlock, distinguishing it from other types of deadlocks

[*]These authors contributed equally to this work.
[†]Shizhen Zhao is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCOMM '25, Coimbra, Portugal
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1524-2/25/09
https://doi.org/10.1145/3718958.3750497

and blockages identified in prior work [13, 23, 28, 75], despite their similar adverse impact on network performance.

Our experiments and analysis in §3 demonstrate that Orderlock occurs more frequently than PFC deadlocks, and it cannot be mitigated by simply adding more resources. These properties have made running a network with the risk of triggering Orderlock impractical from both performance and resource perspectives.

Ensuring protocols are Orderlock-free by breaking the necessary conditions is a more practical approach. In §5, through a case study on optimizing AI training workloads, we comprehensively explore, evaluate, and analyze the trade-offs among Orderlock-free protocols. We find that existing commodity hardware designs cannot support these protocols in achieving optimal performance and scalability. Thus, we finally discuss the essential requirements for future HPN designs.

In summary, our contributions are as follows:

(1) We identify and analyze **Orderlock**, a new type of deadlock, and explore its necessary and sufficient conditions.
(2) We demonstrate that operating an Orderlock-risky network is impractical from performance and resource perspectives.
(3) We explore and evaluate Orderlock-free High-Performance Network protocols through a case study on AI workload tuning, proposing solutions for different scales.
(4) We highlight the importance of *orderless delivery support* in AI-ready HPN hardware and protocol design, showing potential for up to 11.2× communication and 4.54× training performance improvement over off-the-shelf solutions.

This work does not raise any ethical issues.

2 Orderlock and Why They Form

We have seen that network protocols emerging nowadays are designed with more or less one of the three features $\mathbb{I}$, $\mathbb{L}$, or $\mathbb{O}$. Since they are such nice features, is it possible to design a protocol that achieves all three simultaneously? While designing our strawman protocol, we found that satisfying all three features at the same time leads to a new kind of deadlock, which we term **Orderlock**.

In this section, we first introduce the formation of Orderlock, the characteristics of Orderlock, then analyze and prove a necessary and sufficient condition of Orderlock.

2.1 What is Orderlock?

Consider a flow f being load-balanced onto two paths. An example is shown in Fig. 1. The flow is organized into packets $\{p^1, p^2, \dots\}$ and the receiver expects that the flow is in original order when it is delivered to upper layer protocols and applications ($\mathbb{I}$n-order delivery).

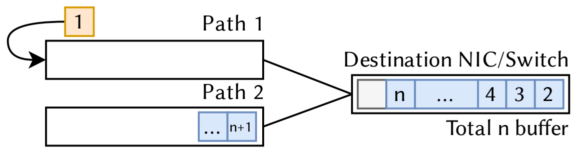
Due to load balancing, failure rerouting, etc., happening in the network, packets from a single flow can be distributed onto different paths and the transmission is $\mathbb{O}$ut-of-order. For the destination NIC/switch, $\mathbb{I}$n-order delivery requires these packets to be sorted, and those early arriving packets be stored in the reordering buffer.

In the example scenario, the reordering buffer has n empty slots for packets. Packet p^1 is assigned to Path 1 while other packets $p^2, p^3, \dots, p^{n+1}$ are assigned to the other Path 2.

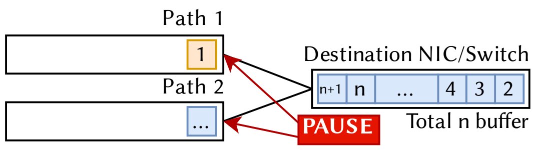
As shown in Fig. 1, for the configuration at time t, packet p^1 on Path 1 has not arrived at the last hop, while packet p^{n+1} on Path 2 is already pending to send at the last hop. The reordering buffer on the Destination already stores $n-1$ packets $\{p^2, \dots, p^n\}$ that arrived earlier. At time $t+1$, the packet p^{n+1} is transmitted to the Destination NIC/switch, at which point the packet p^1 just arrives at the last hop on Path 1, and it's late due to $\mathbb{O}$ut-of-order.

At this time, the reordering buffer is full. To ensure $\mathbb{L}$ossless transmission, the Destination NIC/Switch must send a pausing signal to hold its previous switches up, and prevent excessive packets overflowing the reordering buffer.

At this point, the system enters a standstill: the sender on the last hop of Path 1 is waiting for the Destination to release some buffer so that p^1 can be transmitted, while the Destination NIC/Switch is waiting for p^1 arrival to dispatch packets and release the reordering buffer. We denote this standstill as **Orderlock**.



Time t: Packet p^{n+1} of Path 2 is sending to the destination switch while p^1 has just arrived.



Time $t+1$: Pause is issued to prevent buffer overflow, blocking both p^1 and the packet on Path 2.

Figure 1: ILO causes Orderlock

2.2 Orderlock is a New Type of Deadlock

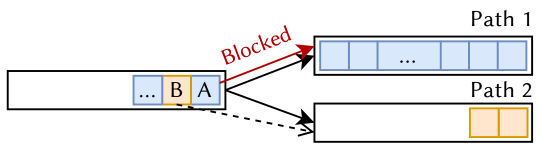
2.2.1 Orderlock has high occurrence frequency. Compared to PFC deadlocks and OS deadlocks which are caused by multiple participants competing for critical resources, i.e. *hold-and-wait*, Orderlock can be triggered by a single flow on its own, as soon as an eligible out-of-order situation occurs. This results in a higher triggering rate independent of overall network load and peer flows.

Experiments and analysis in §3.1 also demonstrated that for the existing $\mathbb{O}$ut-of-Order capable load balancing algorithm, for a single flow, Orderlock can be triggered within the first 800KB when load is balanced over 4 paths. In the case of multiple flows on a regular topology, almost all the flows are affected by Orderlock.

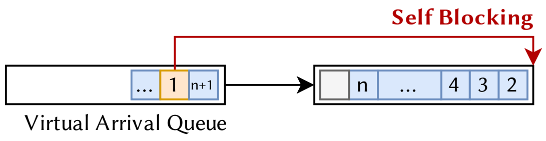
Thus, the common *mitigation* strategy for PFC/OS deadlocks, let it happen and then recover from it, is impractical here. Due to Orderlock's high occurrence frequency, once triggered, the affected flow's throughput, latency, and jitter will be seriously degraded.

2.2.2 Orderlock is different from PFC deadlock. Although PFC (and similar pausing mechanisms introduced by $\mathbb{L}$) is involved in Orderlock formation, Orderlock is fundamentally different from the well-known PFC deadlock. Previous researchers have pointed out that PFC deadlock arises from Cyclic Buffer Dependency (CBD) in the network [28], whereas Orderlock's occurrence is independent of CBD. The example given in Figure 1 would form a minimal counter-example: the blockage at the destination NIC / Switch is self-inflicted, and the buffer dependency in this example does not form a cycle.

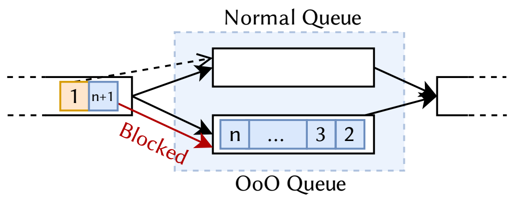
This means that many common approaches to prevent PFC deadlocks (e.g. Priority Queue switching [31] / Virtual Channel [38] breaking the cyclic dependency, and CBD-free routing algorithms [1, 23, 73, 74]) cannot circumvent Orderlock. For topologies where it is difficult to implement CBD-free routing algorithms, such as Dragonfly, Expander graph, etc., Orderlock can further make wiring, routing, and protocol design more complicated. PFC deadlock detection-recovery techniques, e.g. canceling PFC when timeout [31], cannot distinguish Orderlock from regular PFC deadlock, which may further worsen the performance.



(a) Head-of-Line Blocking



(b) Orderlock



(c) Orderlock with VOQs

Figure 2: Orderlock is a self-blocking deadlock

2.2.3 Orderlock is distinct from HoL Blocking. Head-of-Line blocking (HLB) [13, 23, 75] occurs when two packets heading to different destinations are dispatched from the same sending queue. As shown in Fig. 2, packet A is heading to the upper queue and packet B is heading to the lower queue. Even though the destination of packet B has buffer availability, packet B is still blocked by packet A because A is ahead of B.

Orderlock is similar to HLB when forming, but they are fundamentally different and HLB's solution is not applicable to Orderlock. We visualize the inflight sequence from different paths as a Virtual Arrival Queue (VAQ) in Figure 2. In this case, packet B and p^1 are in a similar blocking state: they cannot leave their queue because older packets A and p^{n+1} in front of them are blocked.

However, unlike HLB where Path 1 will finally resume, taking away packet A and resolving the blocking situation, the paused queue in an Orderlock state cannot resolve itself. Orderlock is self-blocking: The condition for reordering buffer to resume and accept p^1 is that p^1 has entered the reordering buffer.

Virtual Output Queue (VOQ) is a mechanism used to achieve HLB-free switching, it's also used by some out-of-order mitigation methods such as Conweave [66] and SQR [57]. However, they cannot exempt from Orderlock. Take Conweave-like approaches as an example. As shown in figure 2, we use two virtual queues storing normal and Out-of-Order packets (OoO) separately, to avoid the critical packet p^1 being blocked behind p^{n+1}. However, these queues are still bounded by limited buffer size, and packets in the out-of-order queue cannot leave without p^1, even if the normal packet queue is empty and the receiver queue has space. Packets will eventually fill up the OoO queue and triggers Orderlock.

2.3 Necessary and Sufficient Conditions of Orderlock

Having found a new type of deadlock, the Orderlock, we explore the necessary and sufficient conditions of Orderlock.

2.3.1 Network Protocol. A *network protocol* defines how packets are forwarded in the network and delivered to the end applications. Some protocols may introduce retransmission for recovery, which resends packets that were not delivered in this time. For simplicity of our model, we treat each round of these sending as a full transmission. Packets resent in a subsequent attempt will be of next transmission. In this case, packets in each transmission are unique. We use a configuration c to represent the hidden network states and parameters in the network that may affect the packet sequences, such as congestion level, queueing delay, etc.

Definition 1 (Network Protocol)**.** *Let $\mathcal{U}$ be the space of all packets, i.e. the unit of transmission over the network. A packet sequence $P \in \mathcal{P} := \mathcal{U}^*$ is a unique list, i.e. ordered set of packets. Denote C as space of configuration.*

A network protocol Π under a configuration c is defined as a 2-tuple $\Pi^c := \langle f^c_{TX}, f^c_{RX} \rangle$, where $f^c_{TX} : \mathcal{P} \rightharpoonup \mathcal{P}$ is the transmit function and $f^c_{RX} : \mathcal{P} \rightharpoonup \mathcal{P}$ is the receiver function.

A transmission of an **input sequence** P works as follows: The transmit function f^c_{TX} will first transform it to an **inflight sequence** $P' = f^c_{TX}(P)$, which describes the order of the sequence after transmission inside the network and will arrive at the receiver. Then, the receiver function f^c_{RX}, processes P' to produce the **delivery sequence** $P^* = f^c_{RX}(P')$, the final order when it is delivered to upper layer protocols and/or applications.

However, for a network protocol not all sequences and transmission are valid and *acceptable*, and the protocol may enter a failing state. Therefore, the functions f^c_{TX}, f^c_{RX} are modeled as partial functions that have value only when they *accept* their sequences as an input, otherwise they are in *deadlock* state. For example, P^* will not exist when $P' \notin \operatorname{dom}\left(f^c_{RX}\right)$.

Since deadlocks may happen in f^c_{TX}, and discussing f^c_{RX}'s behavior without an inflight sequence P' is meaningless. We introduce a simplifying assumption that abnormal transmission scenarios where $f^c_{TX} \not\rightarrow$, suchas routing loop and PFC deadlocks, are excluded. In the following context, the transmit function will be considered as a total function: $f^c_{TX} : \mathcal{P} \to \mathcal{P}$.

For example, consider a sequence $P := [1, 2, 3, 4, 5]$ transmitted via three protocols, and the inflight sequence becomes $P' := [1, 2, 4, 5]$, where packet 3 is lost. The first protocol does not allow out-of-order packets at all, the second uses Go-back N retransmission, and the third takes everything as is (e.g. RTP [61]). As shown in table 1, each protocol exhibits different acceptance criteria, leading to different (or non-existent) delivery sequences.

Table 1: Receivers have different acceptance criteria

Protocol	Accept	Delivery Sequence
No Drop Allowed	$\not\twoheadrightarrow$	*not exist*
Go-back N	$\rightarrow$	`[1,2]`
RTP	$\rightarrow$	`[1,2,4,5]`

2.3.2 Max Out-of-order Arrival. Reordering requires buffering packets that arrive early. We define the *Max Out-of-Order Arrival* (MOA) of an inflight sequence, which represents the required buffer size for reordering the whole sequence:

Definition 2 (MOA). *Given an input sequence P and its inflight sequence $P' = f^c_{TX}(P)$. Let $\text{idx}_P(p)$ be the index of a given packet p in the sequence P. The Out-of-Order Arrival (OA) level of a packet p is defined as the total number of packets that are located after p in P but before p in P':*

$$OA(p) := ||\{s \in P, \text{idx}_P(s) > \text{idx}_P(p) \text{ and } \text{idx}_{P'}(s) < \text{idx}_{P'}(p)\}||$$

Then Max Out-of-Order Arrival is defined as

$$MOA := \max\{OA(p), \forall p \in P'\}$$

For packet p, $OA(p)$ represents the number of packets that will arrive earlier than p. These packets must be stored inside the reordering buffer for correct packet reordering. Thus MOA equals to the minimum reordering buffer requirement: It represents the maximum packets being stored in this buffer while reordering the whole sequence.

With MOA definition, we can quantize $\mathbb{O}$ut-of-order Capability: For reordering limit n of the protocol (e.g. the reordering buffer size), an $\mathbb{O}$ut-of-order capable protocol allows an inflight sequence P' with $MOA \geq n$.

2.3.3 ILO Theorem and Orderlock. Here we formally define, $\mathbb{I}$n-order Delivery, $\mathbb{L}$ossless Transmission, and $\mathbb{O}$ut-of-order Capability as follows:

Definition 3 ($\mathbb{I}$, $\mathbb{L}$, and $\mathbb{O}$).
In-order Delivery ($\mathbb{I}$)

The protocol needs to ensure that it delivers the content in original order to the upper layer: For delivery sequence $P^* = f^c_{RX}\left(f^c_{TX}(P)\right)$: $\forall p, q \in P^*, \text{idx}_{P^*}(p) < \text{idx}_{P^*}(q) \Rightarrow \text{idx}_P(p) < \text{idx}_P(q)$.

Lossless Transmission ($\mathbb{L}$)

The protocol and its supporting devices should not drop packets in in any part of the network: (1) transmit: for input sequence P and inflight sequence $P' = f^c_{TX}(P)$, $||P|| = ||P'||$. (2) receiver: for inflight sequence $P' \in \text{dom}\left(f^c_{RX}\right)$ and arrival sequence $P^* = f^c_{RX}(P')$, $||P'|| = ||P^*||$.

Out-of-order Capability ($\mathbb{O}$)

The protocol allows having out-of-order arrival sequences exceeding protocol's reordering limit n: $\exists P$ s.t. $P' = f^c_{TX}(P), MOA \geq n$.

By this definition, we propose the ILO theorem delimiting the necessary condition of Orderlock (proof in Appendix A).

Theorem 1 (ILO). *Simultaneous achievement of In-order Delivery, Lossless Transmission, and Out-of-order Capability is a necessary condition for deadlock.*

We formally define Orderlock, and show that it's a kind of deadlock triggered under ILO condition:

Definition 4 (Orderlock). *For a protocol achieving In-order Delivery and Lossless Transmission, at the receiver with limited reordering buffer n, when handling an inflight sequence P' of P with $MOA \geq n$, with an initially empty reordering buffer R, it will enter a situation where R becomes filled up by out-of-order packets. The final situation is called Orderlock.*

No packets can leave or enter R under Orderlock situation unless breaking $\mathbb{I}$ or $\mathbb{L}$: By MOA definition, $\exists S[1 \dots m]$ being a subsequence of P', s.t. $S[m] = p$, $OA(p) \geq n$. By OA definition, $m \geq n+1$, and we have that $S[1 \dots m-1]$ containing at least n packets s that $\text{idx}_P(s) > \text{idx}_P(p)$ and $\text{idx}_{P'}(s) < \text{idx}_{P'}(p)$, which must be stored in R. Then R is full before p can enqueue. As required by $\mathbb{I}$, reordering buffer shall not send $\forall s$ s.t. $\text{idx}_P(s) > \text{idx}_P(p)$ unless p is sent, and none of R shall be dropped to comply with $\mathbb{L}$. Meanwhile, $p = S[m]$ cannot be inserted into R before some packet is dequeued from R, thereby entering a deadlocking state and $f^c_{RX}(P') \not\twoheadrightarrow$.

By this definition, we give the sufficient condition of Orderlock (proof in Appendix A):

Theorem 2 (Sufficient Condition of Orderlock). *A protocol with reordering limit n achieving $\mathbb{I}$n-order delivery and $\mathbb{L}$ossless transmission, handling an inflight sequence P' with $MOA \geq n$ will trigger Orderlock.*

3 Properties of Orderlock-Risky Network

This section analyzes key characteristics of Orderlock-risky networks through empirical evaluation.

3.1 High Occurrence Frequency of Orderlock

We evaluate 3 $\mathbb{O}$ load balancing mechanisms: Random Packet Spraying (RPS) [16], QDAPS [29], and Adaptive Routing. For each packet, there could be multiple shortest paths to its destination. RPS randomly selects a path for each packet at each switch; QDAPS chooses a port with slightly higher queue length to minimize out-of-order delivery; Adaptive Routing algorithm directs each packet to the port with the least queue length. In Figure 3, we compare their out-of-order delivery rates, i.e. the percentage of flows suffering out-of-order delivery.

3.1.1 Single Flow Orderlock. To check the incidence of Orderlock, we simulate a three-layer fat-tree topology as shown in Figure 4, using SQR-based in-network reordering [57] at destination switch with reordering limit at 64 packets, denoted as D in the figure.

Since QDAPS and Adaptive Routing are queue-length dependent, apart from the observed flow (named Flow 1), we add an interfering

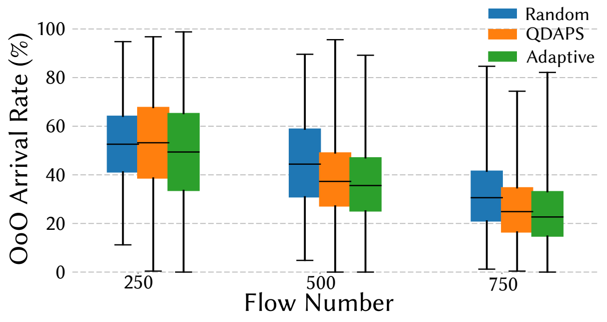


Figure 3: Packet spray causes out-of-order arrival

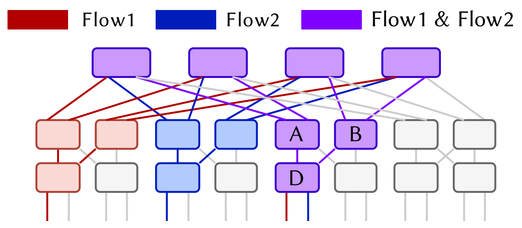


Figure 4: Topology and flow configuration

flow Flow 2 to create extra randomness. This background flow is separated from the observed flow when it enters the Destination switch *D* and will not participate in Flow1's reordering process. Note that Orderlock is a self-blocking deadlock.

If there were other deadlocks in the network, such as PFC deadlocks, it will interfere with our measurements. We use a CBD-free shortest path routing scheme to measure how often Orderlock occurs in a PFC deadlock-free environment. With PFC enabled, we evaluate the three fine-grained load balancing strategies with packet-level simulator NetBench [37].

For three selected $\mathbb{O}$ protocols, RPS, QDAPS, and Adaptive Routing, figure 5 shows the reordering process of switch *D* during Orderlock events, and the sending status of the downlink from switch *A* to *D*. The reordering process is represented by the Next Sequence Number, the minimum packet sequence number of the remaining arrival sequence that has not been delivered to upper layer protocol or applications, i.e. the *key* packet. Port Sending Rate shows downlink status, where the sending rate at 100% marks line speed transmission and the sending rate at 0% represents pausing where no packet is transmitted.

Take RPS in Fig. 5a as an example: at around 964 μs, the reordering process has forwarded the packet with sequence number 770 and begins to wait for the packet with sequence number 771. However, this packet is held by *A*, which is waiting for a PFC resume frame before transmitting. At this time, Orderlock occurs, and we mark a red star in Figure 5a indicating the start of Orderlock. As shown in the figure, all three protocols eventually lead to the occurrence of Orderlock. Among them, RPS is triggered the least frequently, while the other two algorithms designed for mitigating delay or out-of-order, are both more likely to trigger Orderlock.

For QDAPS, in every cycle, packets are put into the queue which has slightly higher queue length than previously chosen queue, in order to reduce out-of-order in this cycle. Once the queue length exceeds the threshold, the packet will be directed to the port with the lowest queue length and start a new cycle. The cycle shifting would enlarge the MOA: 1) Similar to flowlet with insufficient time gap, where the arrival of two cycles will overlap, leading to fewer, but longer out-of-order sequences and resulting in larger MOA. 2) QDAPS always selects a longer queue, which is detrimental to flow control: longer queues are more likely to congest and trigger a pause, which further lengthens MOA (see §3.2).

For Adaptive Routing, the forwarder tends to evenly distribute the workload to avoid congestion, which naturally leads to out-of-order. With RPS, packets may enter longer queues, which in turn reduces the natural out-of-order during queueing (which is the motivation of QDAPS), whereas in Adaptive Routing queue lengths are more balanced, which is more susceptible to pauses and volatility in the network. On the other hand, Adaptive Routing also reduces the occurrence of pauses. This is the reason why Adaptive Routing has a lower average out-of-order arrival rate in Figure 3, but still triggers Orderlock earlier than RPS.

3.1.2 *Orderlock in the Whole Network.*

To compare the occurrence frequency of Orderlock under different scenarios, we conduct a comparison test under the same topology in Table 4, with same reordering limit at 64 packets. We randomly generate 250, 500, and 750 concurrent flows for a duration of 0.1s and run each scenario 1000 times for RPS, QDAPS, and Adaptive Routing strategies. A larger number of flows means that network concurrency will increase, and the randomness will also increase. For each scenario, we count the number of runs that trigger Orderlock and depicted in Table 2. All strategies lead to Orderlock occurrences at a notable frequency, and eventually lead to the risk of Orderlock occurrence, with various frequencies differing from the exact flow numbers. Even taking the smallest estimation from these data, none previously mentioned mitigation methods will deliver a decent performance.

Table 2: Orderlock occurrence frequency

1000 times total

Strategy	Number of Flow		
	250	500	750
RPS	71	581	904
QDAPS	253	673	915
Adaptive	201	603	861
No reorder	0	0	0

To further reinforce the fact that the results we measured were Orderlocks and not some other kind of PFC-caused deadlock, we also ran the control test without the reordering process at *D* (Ignore the out-of-order arrival). As shown in Table 2, without reordering at *D* switches, not a single deadlock will occur in all scenarios. All flows are completed successfully without retransmission. Thus, we are granted to conclude that all deadlocks described above are Orderlocks.

Takeaway. In today's typical data center network architecture, Orderlock occurs very frequently if all ILO are achieved. It is far different from PFC deadlocks. Mitigation i.e. *fail and recover* strategy would not be practical.

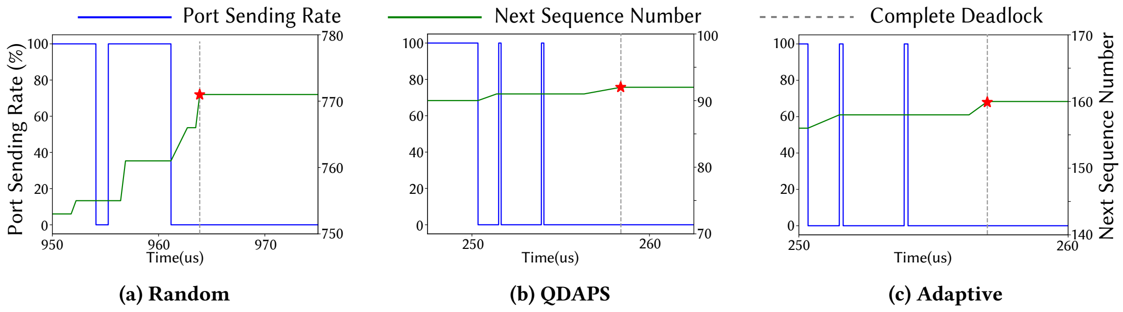


Figure 5: Orderlock occurrence time, with detailed port sending rate from A to D

3.2 Understanding the Buffer Requirement

From the definition of $\mathbb{O}$, whether a protocol qualifies as $\mathbb{O}$ depends on the reordering limit n, i.e., the size of the reordering buffer. By comparing the necessary and sufficient conditions for Orderlock, we observe that in a risky network, the frequency of Orderlock can be reduced if the number of flows with $MOA(P) > n$ is minimized. This raises two questions and we will discuss them in this subsection: 1) Is increasing the reordering buffer size n an effective strategy for avoiding Orderlock? 2) How large should the reordering buffer be to handle the traffic?

Is a BDP-sized buffer enough? Experienced network engineers often use the bandwidth-delay product (BDP) as a reference for tuning the reorder buffer, as it represents the maximum amount of data *on-the-fly* i.e. sent but not yet received. However, this correlation does not hold in $\mathbb{LO}$ networks. As illustrated in Figure 6:
1) In lossy networks, queues drain at the line rate, and excess packets are dropped, ensuring a bounded delay while reducing the effective bandwidth and the actual on-the-fly packet amount. Packets either have a guaranteed delay or dropped. In contrast, $\mathbb{L}$ networks pause queues when full, leading to longer and overall utilization dependent delays.
2) For $\neg\mathbb{O}$ protocols like ECMP, a flow only occupies buffers along a single path. In this case, the buffer forms a chain connected by links along the route. Congestion increases delay but simultaneously reduces bandwidth, keeping the total on-the-fly data comparable to BDP + chain total buffer. However, $\mathbb{O}$ load balancing break flows across multiple buffer chains, and inconsistent pausing across these paths makes delay unpredictable.

In $\mathbb{O}$ and $\mathbb{L}$ scenarios, the worst-case latency is not guaranteed, rendering BDP-based buffer size estimation ineffective. The combination of $\mathbb{O}$ and $\mathbb{L}$ exacerbates the issue. The reordering buffer must hold all early-arrived packets from non-congested paths, while in-order packets may be blocked due to congestion elsewhere in the network. Consider the scenario in Figure 6, where the lower path is HoL blocked, but the upper path is not. In such case, even a reordering buffer sized at BDP + the total buffer in network is insufficient to guarantee Orderlock-free.

Buffer Requirement. To further investigate the factors affecting the required buffer size, we conduct simulations using a 3-layer fat-tree topology, parameterized by their switch node fan-out. The nominal BDP (i.e., bandwidth and delay measured in an ideal congestion-free scenario) between any two servers in the network remains unaffected by the fan-out. We introduce a reorder buffer with no size limit to evaluate the buffer size requirements for NIC-based reordering.

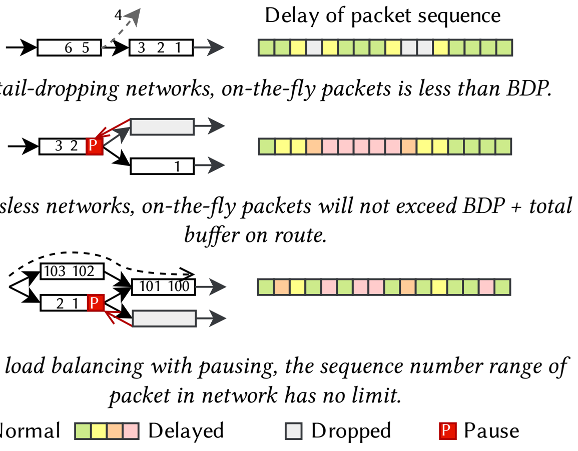

Delay of packet sequence
4
6 5
3 2 1
For tail-dropping networks, on-the-fly packets is less than BDP.
3 2 P
1
For lossless networks, on-the-fly packets will not exceed BDP + total buffer on route.
103 102
101 100
2 1 P
When load balancing with pausing, the sequence number range of packet in network has no limit.
Normal
Delayed
Dropped
P Pause

Figure 6: Temporal distribution of packet arrival delays in different networks

Using a shared headroom instead of a per-flow dedicated buffer can reduce buffer usage. However, because of the need for ordering, it is still required to reserve free space for all possible early-arrived packets to prevent Orderlock.

Our experiments demonstrate that with $\mathbb{LO}$, specifically RPS in a PFC network, the buffer occupancy and demand for receiver-side reordering increase not only with BDP and total link buffer size but also with flow length. Even at a small scale (e.g. a fat-tree with fanout=16 and 15×10MB concurrent symmetric flows), the buffer demand for the worst-performing flow reaches 5.45MB, which is 12 times the BDP ($\sim$ 456KB). The total reordering demand for the entire workload amounts to 47MB, requiring 24 pieces of Connect-X 5 NIC chip's buffer to accommodate it [45, 71].

We analyze the reordering buffer demand for each packet arrival within a single flow. As illustrated in Figure 7, the buffer demand exhibits a long-tail. The tail grows with the flow length, requiring a buffer large enough to accommodate the entire tail; otherwise, Orderlock is triggered.

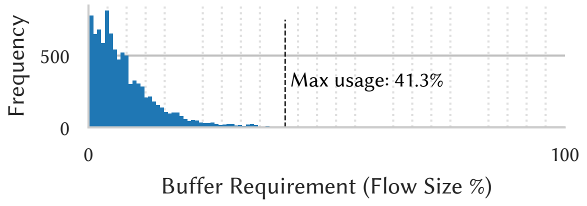


Figure 7: Buffer requirement during entire flow

Takeaway. For typical out-of-order (𝕆) load balancing protocols, achieving ILO requires a buffer size both costly and impractical. Simply increasing the buffer size to extend the reordering limit is insufficient to prevent Orderlock.

3.3 Orderlock in Real Networks

3.3.1 Testbed Experiments. In this section, we use a testbed to corroborate the findings from the previous analysis and simulation. We use a Tofino switch to construct a topology similar to Figure 4: We implemented P4 program and a custom isolation routing table to realize VRF and isolate ports to create a three-layer forwarding structure of fat tree, and for egress ports on equivalent paths, we use random strategy to spray packets. On the host side, we use NVIDIA ConnectX-5 NIC to send and receive packets. All links work at 100Gbps with 1024 bytes MTU. We configured lossless network based on PFC on the switch and NICs, and constructed two flows sending simultaneously each time for experiment. Since in-NIC reordering requires hardware support which CX-5 NIC does not have, additional sequence numbers are carried by the packets and flows are emitted under RoCEv2 UD mode to measure the Orderlock trigger frequency along with maximum MOA.

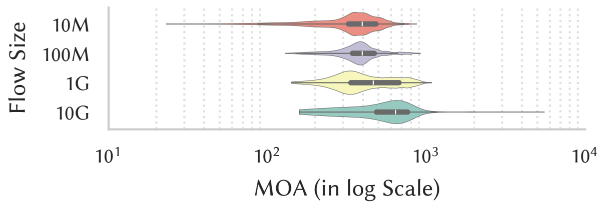
Orderlock Occurence Frequency. We set the flow length from 1 MB to 10 GB, used random packet spray and different reordering buffer size from 64 to 256, measured the Orderlock trigger frequency of 2000 flows, and obtained the results shown in Table 3. The experiment shows that the Orderlock trigger frequency is high and orderlock itself is unavoidable in this situation.

Table 3: Orderlock occurence frequency in testbed

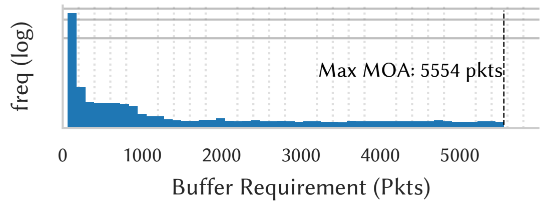
2000 times total

Buffer size (n)	Flow Size				
	1MB	10MB	100MB	1GB	10GB
64	142	1994	2000	2000	2000
128	0	1950	2000	2000	2000
256	0	1740	1868	1856	1856
(Flow ECMP)	0	0	0	0	0

Buffer Requirement. We measured the requirement for reordering buffers throughout the process. Under the conditions above, for 2000 flows per size, we measured the MOA for each flow, and their distribution are displayed in Figure 8a. The figure shows that as the flow size increases, not only does the average MOA increase, but the maximum MOA among all flows also increases accordingly.



(a) MOA distribution of different flow sizes



(b) Buffer requirement of single flow

Figure 8: Flow buffer requirement

We also measured the OA, i.e. the buffer requirement during the entire flow, of 10GB flows. As shown in Figure 8b, the buffer requirement still yields a long-tail distribution, and the tail becomes longer as the overall length of the flow increases.

3.3.2 Orderlock in Production Networks. Our testbed experiment on two flows already shows why Orderlock-risky networks is impractical, while in a fully utilized production network having more concurrency, more randomness, and more pauses, Orderlock should be triggered more often and exhibit a longer MOA tail.

This explains why mature protocols are all Orderlock-free, and such issues are rare in past networking practices. In traditional TCP/RDMA data centers, adopting an 𝕆 load balancing algorithm results in zero throughput and crashes applications.

In the past, there were designs inadvertently formed ILO, triggering unidentified Orderlocks and causing failures. Early TCP Selective Repeat implementation in Linux kernel stores all early-arrived packets, with no dropping or out-of-order delivery mechanisms when memory is full. This ultimately allowed attackers to carry out DoS attacks by constructing malicious packet sequences with large MOA to fill up server's memory [44, 62].

Conclusion. The high occurence rate of Orderlock and impractical buffer requirement outlined above render Orderlock-risky networks impractical from both performance and resource perspectives.

4 Existing Protocol Design Outlook

Given the demonstrated impracticality of Orderlock-risky networks, implementing Orderlock-free protocols would be the only viable solution. As established by Th. 1, breaking any of the three necessary conditions prevents Orderlock formation. There are no silver bullets in network protocol design. Before designing our own solution, we first analyze existing research through the lens of three design concepts, with categorization framework shown in Figure 9.

We break each feature into two directions: For protocols achieving specific features (e.g. 𝕀), we identify their core mechanisms. For the opposite (e.g. Non 𝕀), we discuss their compensation strategies

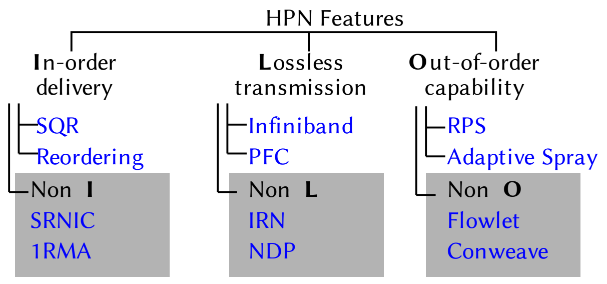


Figure 9: Three features and the corresponding complements.

for the absent features and optimization techniques in their specific scenario.

4.1 𝕀n-Order Delivery

In-order delivery enables receivers to reduce buffer usage for reordering and respond more quickly to network faults, such as packet loss and other anomalies. For instance, TCP Reno's Fast Recovery can rely on duplicate ACKs due to 𝕀, to quickly adjust the rate and congestion window [18].

In single-path networks, in-order delivery is inherently guaranteed. For networks at risk of out-of-order arrivals, reordering these packets before it is delivered to the upper layer is the natural 𝕀n-order delivery solution. Existing approaches can be categorized into *in-network reordering*, performed on the switch, or *reordering in NICs*, executed just before packets are forwarded up to applications.

Current in-network switch reordering approaches [7, 21, 35, 39, 57, 66] typically rely on supplementary queues to handle out-of-order packets. For instance, SQR [57] and Linkguardian [35] use a single queue to store both in-order and out-of-order packets, deciding whether to forward or recirculate packets based on their sequential order. In contrast, FlowSail [39] and ConWeave [66] employ a dedicated reservation queue for out-of-order packets, leveraging a *flag packet* mechanism to mark transition between the normal queue and the reservation queue.

On the other hand, solutions like Juggler [7, 21, 51] perform reordering within the NIC buffer. However, NIC-based reordering typically demands large buffers (cf. §3.2), which are expensive to implement. To prevent buffer overflow, Juggler uses a timer to flush long-waiting packets, passing the packet loss error to upper protocols. Similarly, SRD [63] delivers packets to the host even if they are out-of-order after hitting the threshold. In cases of buffer overflow, these best-effort approaches violate 𝕀n-order delivery and can no longer be considered having 𝕀n-order properties.

Non 𝕀. Both in-network and on NIC reordering require additional operations and buffer, thus not widely supported by commercial devices. In such cases, delivering out-of-order content to the upper layer becomes a viable alternative, sometimes referred to as *connectionless*. Works like 1RMA [49, 65] encapsulate operations and payloads directly in packets. However, messaging beyond the MTU size ultimately necessitates sequencing and ordering, which can impose significant CPU overhead if handled by upper-layer applications. Solutions like SRNIC [71] add remote memory address locators for each packet, enabling direct writing into receiver memory via DMA without requiring context from other packets. This approach relieves reordering tasks from the CPU.

4.2 𝕃ossless Transmission

Lossless transmission is important to simplify the design of protocols, increase throughput, and reduce latency. Due to traditional tail-dropping queue design, packet loss in a traditional network is very common. In order to avoid packet loss, mechanisms such as PFC [33, 34] and hop-by-hop credit-based flow control in Infiniband [46] are proposed. The former will send a notification to its peer when the queue is about to overflow, while the latter sends packets only when there is free space to store them in the receiver queue.

Non 𝕃. While 𝕃 addresses the problem of packet loss, it also introduces some new problems such as Head-of-Line blocking [13, 23, 75] and PFC deadlocks [28]. Therefore, many researchers have also studied how to improve the performance in lossy networks. Work like IRN and NDP [24, 48] introduces *Selective Repeat* (SR) to replace vanilla RoCEv2's Go-back-N algorithm for adapting lossy RDMA. Similar to SACK in TCP [19], by storing early arrived packets and notifying the sender, the amount of retransmitted data in case of out-of-order arrivals is reduced. NDP further uses switches to pass information of dropped packets to better support 𝕆 and avoid unneeded retransmissions.

SR requires extra memory to store out-of-order packets, which raises challenge for NIC design [36]. Works like 1RMA [65] and other *connection-less* protocols use software-based flow and congestion control, which will burden the CPU like TCP does. SRNIC [71] implements SR by storing these packets directly into memory via DMA and bypassing the CPU. However, DMA poses challenges to the hardware architecture and driver design. For example, in AI tasks, technologies like NVIDIA GPUDirect RDMA avoid duplicate copies of host memory through PCIe peer-to-peer communication between the GPU and the NIC, which are not fully exposed to the NIC's DMA engine [53].

4.3 𝕆ut-of-Order Capability

The requirement for Out-of-Order Capability is inseparable from the development of load balancing techniques. Load balancing allows us to spread the load over multiple paths to fully utilize the bandwidth. However, traditional protocols are not designed with multi-path transmission, and unable to handle frequent out-of-order transmission efficiently, which restricts load balancing options.

Apart from pure packet spraying scheme like RPS [16, 32, 67], QDAPS [29], and Adaptive Routing [30] discussed earlier in §3.1, there are works aiming at reducing the out-of-order during load balancing. DRILL [22] and similar solutions dynamically select links for packet forwarding based on real-time load conditions, favoring links with lighter load while preventing large out-of-order by limiting path selection. There are also partial packet-level routing works [27, 41] distinguish between large and small flows to implement different routing strategies, preventing disturbance on critical flows while reducing conflict.

Non-clos networks may require higher out-of-order capability to fully utilize its network bandwidth. These network architectures, such as expander networks [73, 74] and hybrid optical/electrical networks [9, 10, 55, 68] can establish low-latency networks at lower cost by leveraging traffic-aware topology and routing optimization. However, routing in these networks are more complicated than that in Clos networks. Due to the insufficiency of shortest path bandwidth, traffic may be split between shortest and non-shortest paths [42, 43], which naturally increases MOA.

Non $\mathbb{O}$. Researchers have also proposed $\neg\mathbb{O}$ friendly load balancing algorithms to improve performance. Some protocols can tolerate small out-of-order arrival with limited MOA, which brings more room for maneuver.

LetFlow and other time-based flowlet-level load balancing approaches [2, 58, 70] perform further load balancing by means of flowlets, i.e. flow separated by large enough time gaps. Since flowlets are originally based on TCP flow control bursts, while RDMA networks barely have a proper gap due to differences in flow control algorithms, works such as HF^{2}T [11] manually trigger flowlet switching by introducing a deliberate delay on the host side.

Gaps must be sufficient between regular flowlets, otherwise two sequential flowlets may overlap when arrival. If the protocol can handle limited MOA, the gaps can be shrinken and provides better load balancing. Works like Presto [17, 25] uses fixed-size flowlets without gap, and perform reordering with CPU/NIC on the overlapped part. Works like Conweave [66] work basically with flow-level load balancing, but allow path changing in the middle of a flow and handle the out-of-order delivery through in network reordering to circumvent flow congestion.

Works like Alibaba-HPN [8, 56] further break down the flow directly from the source, dividing a single logical flow into multiple actual flows to better participate in load balancing, without changing original flow-based NIC behavior. This approach requires modifications to the client side and is commonly observed in optimization of collection communication libraries.

5 Case study: Implications on AI-HPN performance tuning

As discussed in §1, AI training workloads differ significantly from traditional DCN workloads, and current protocols designed for IDC fall short of delivering optimal performance. Which combination of features will achieve the best performance without triggering Orderlock? In this section, we explore the trade-offs in designing Orderlock-free protocols, aiming to improve throughput in non-oversubscribed fat-tree topology running distributed AI training workloads.

5.1 Orderlock-free Protocols

Since ILO is a necessary condition for Orderlock, breaking any one of these properties can prevent its occurrence. As illustrated in Figure 10, we explore 4 Orderlock-free approaches: 1) For $\mathbb{IL}$, we evaluate $\neg\mathbb{O}$ load-balancing schemes, such as flow ECMP, flowlet switching, ConWeave, and credit-based flow control, to ensure out-of-order is under control; 2) For $\mathbb{LO}$, we employ orderless RDMA, leaving the sorting work to the upper-layer application; 3) For $\mathbb{IO}$, we utilize lossy RDMA networks with selective repeat mechanisms to reduce retransmission overhead. 4) For $\mathbb{O}$-only, we examine NDP, which employs Cut Payload (CP) for precise retransmission of dropped packets. Other methods are less compelling: TCP, a well-established $\mathbb{I}$-only solution, does not have a good reputation in AI-HPN. $\mathbb{L}$-only is a degraded version of $\mathbb{LO}$, as without the requirement for $\mathbb{I}$, there is no advantage to using a $\neg\mathbb{O}$ load-balancing scheme (cf. §5.1.1).

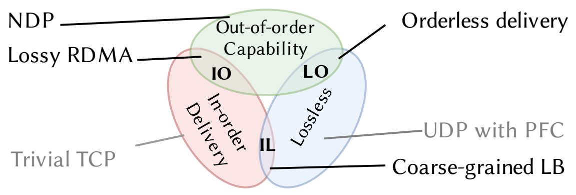


Figure 10: Orderlock-free approaches

5.1.1 $\mathbb{IL}$*: Limited Out-of-Order Arrival.* We discuss of multiple $\neg\mathbb{O}$ protocols, from those strictly no out-of-order ECMP, flowlet, adaptive route switching, to buffer bounded packet spray with credit-based flow control, which uses credit to strictly limit MOA less than reordering buffer size.

No-Reordering Approaches. For ECMP, the switch extracts flow identifiers `(src_ip, src_port, dst_ip, dst_port)` from packets and uses them for multipath next-hop selection. For flowlets, we adopt the LetFlow approach [70], which employs an aging flow table with an additional hash factor to identify flowlets, directing them onto different paths. We also evaluate the adaptive switching method proposed in ConWeave [66], which allows switches to reroute flows upon congestion with controlled in-switch reordering.

Flow conflict in ECMP is a well-known issue: since flow paths are fixed, multiple flows may be directed onto the same link, causing congestion. This significantly impacts FCT: if two concurrent flows share the same link, the FCT at least doubles. For AI workloads, which are highly sensitive to tail latency, this severely degrades performance.

We estimate the incidence of ECMP flow conflicts through simulations of random flows in a Clos network. The results, shown in Figure 11, reveal that in large-scale HPNs, over 30% of flows experience more than 2 flow conflicts, with conflict probability increasing along with network scale.

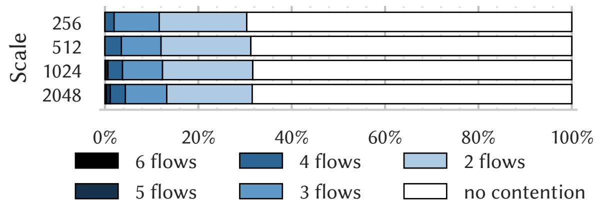


Figure 11: Flow conflict of ECMP

LetFlow and ConWeave's intra-flow switching approaches do not yield better results for AI workloads. For flowlets, a temporal gap is required to slice flows and balance them across paths. However, such gaps are absent in bursty AI scenarios, causing the behavior fallbacks to ECMP. ConWeave reroutes flows during congestion and when free paths are available, extreme congestion can be alleviated. While this approach proven effective in IDC environments, as separating two elephant flows can significantly reduce average FCT, it offers limited benefits in AI scenarios. With higher load, flows are more likely being redirected to already congested links.

Bounded Reordering: Credit-Based Flow Control. Orderlock occurs when the reordering buffer is insufficient to handle packet sequences. By limiting the number of on-the-fly packets with a *credit*, thereby limiting MOA, we can prevent Orderlock while retaining the benefits of $\mathbb{O}$-only load balancing schemes. For Credit-Based Flow Control (CBFC), we use RPS for load balancing scheme, with PFC preventing loss, and a sending credit of buffer size for following evaluation.

The primary performance bottleneck lies in the insufficient reordering buffer. As illustrated in Figure 12, a buffer of at least one BDP size is required to keep flow running. Otherwise, the throughput is throttled (cf. §5.2.1).

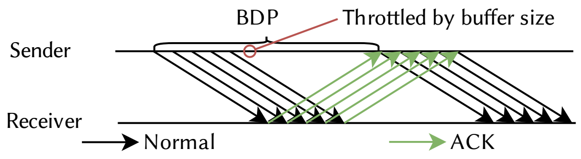


Figure 12: Sending rate is throttled by buffer size

5.1.2 $\mathbb{LO}$: Orderless Delivery. As analyzed in §3.2, performing reordering within the limited buffer of a NIC or switch is impractical, as it will inevitably trigger Orderlock. We left the reordering task to the host side and use in-memory reordering as an example of $\mathbb{LO}$ implementation: similar to bucket sorting, each packet carries the memory address on the receiver and then can be directly written into memory. Every arriving packet is ACKed individually, and the sender uses a bitmap to track ACKed packets along with a timeout timer to retransmit unACKed packets (which indicates failure-related packet loss as general loss is prevented through $\mathbb{L}$).

The primary challenge for $\mathbb{LO}$ lies not in the network implementation but in the system design. Supporting direct data path and eliminating redundant memory copies to host memory and/or devices could be the most significant obstacle when implementing in real practices.

5.1.3 $\mathbb{IO}$: Lossy RDMA. Allowing packet drops during transmission also prevents Orderlock, with extra cost of retransmissions. Using Go-back-N, as implemented in vanilla RoCEv2, is not a good idea, as it requires retransmitting all packets whenever out-of-order delivery occurs. Instead, *Selective Repeat* mechanisms as in IRN and SRNIC, represent the state-of-the-art for lossy RDMA networks. We propose an $\mathbb{IO}$ protocol combining IRN-based RDMA SR with random packet spraying as the load-balancing mechanism.

However, current SR design struggle to handle widespread out-of-order delivery in lossy networks. In such environments, the receiver cannot distinguish between dropped packets and those that are delayed. Consider a scenario where a set of n packets arrives out-of-order, with the head packet arriving last. In this case, the receiver sends $n-1$ SACK packets, and out-of-order delivery is resolved only when the head packet arrives. On the sender side, upon receiving the first SACK packet, it enters recovery mode and rewinds to the head packet. In the worst-case scenario (illustrated in Fig. 13 with $n=6$), this results in $n-2$ packets being retransmitted before the sender catches up, creating significant overhead.

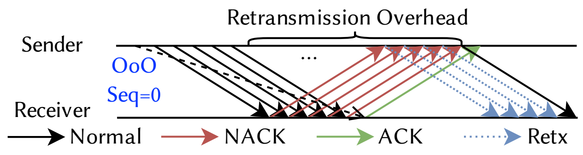


Figure 13: Retransmission when out-of-order happens

SR mechanisms are also affected by buffer size: if the receiver lacks sufficient buffer to hold all out-of-order packets, these packets must be discarded. The $\mathbb{IO}$ protocol imposes a sender-side requirement on the buffer size, typically set to the BDP. However, achieving this within commercial NICs is already challenging, which motivates solutions like SRNIC [71].

5.1.4 $\mathbb{O}$: NDP. The primary limitation of $\mathbb{IO}$ and other retransmission based recovery mechanisms is the inability to distinguish between late and lost packets. NDP [24] addresses this by using on-switch trimming to help receivers identify packet status: at the ingress queue, overflowed packets are not entirely discarded. Instead, their payloads are stripped while retaining their headers, enabling the receiver to determine whether a packet is lost or delayed.

Due to its proactive pull mechanism, NDP faces buffer size throttling problem similar to CBFC. Additionally, trimmed packet headers still consume buffer space, and in real-world switches, full packet loss can occur when buffers are too full and even headers are too large. Such losses can only be detected through retransmission timers, and when they occur, the performance impact is severe. Furthermore, packet trimming that occurs at the end of a flow, where retransmission delays cannot be masked by regular transmission, can lead to higher and non-deterministic FCT.

5.2 Evaluation

In the following evaluations, we use a fat-tree topology with a fanout of 8. As described earlier, to prevent PFC deadlocks, we employ CBD-free shortest-path multipath routing. We run AllReduce operations using Ring, Half-Doubling, and Butterfly [59] algorithms, and AlltoAll operations using Spreadout, Pairwise, and Bruck [5] algorithms across all nodes. To minimize the impact of locality and thoroughly test network performance, we randomly assign ranks to nodes for communication and measure the average performance of shuffled collective communication operations. Unless specified by the protocol, all experiments use DCQCN [75] for flow control. We evaluate performance using the packet-level simulator NS-3 [69], focusing on FCT and the elapsed time for each collective communication operation. To estimate the impact on AI training tasks, as shown in Table 4, we collected timing data from actual clusters

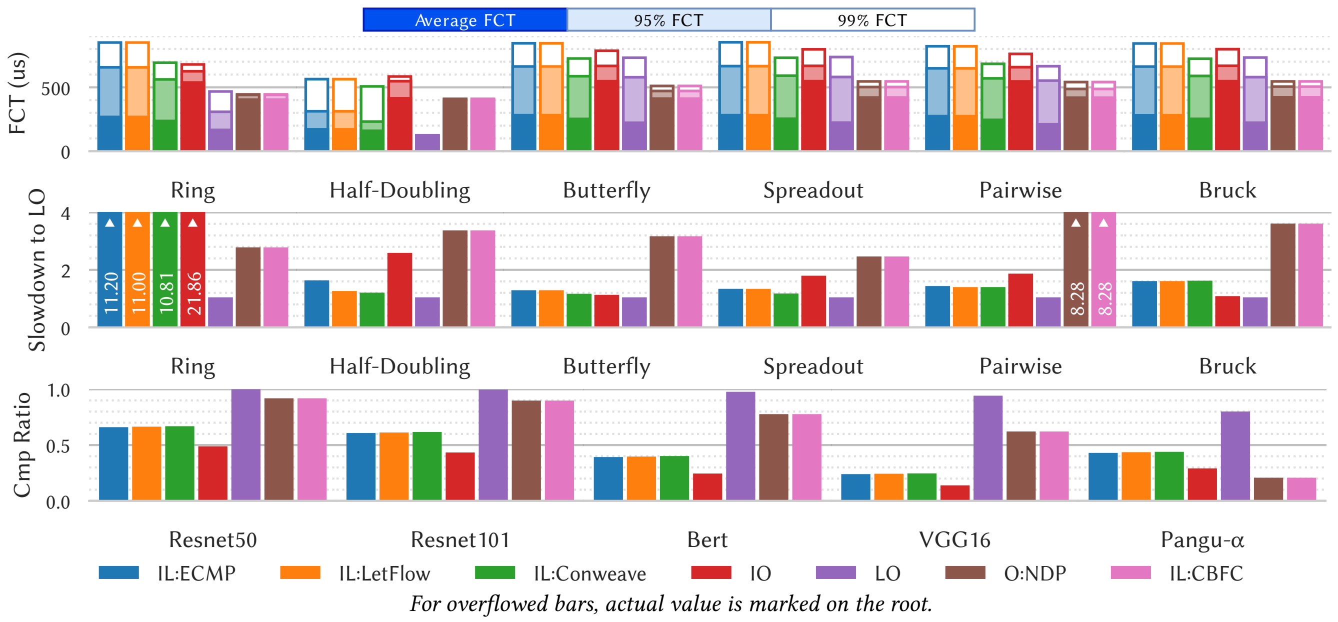


Figure 14: Performance comparison

equipped with 64× Ascend 910b NPUs interconnected via a 100GE leaf-spine network, training models such as ResNet [26], VGG [64], BERT [15], and PanGu-α [72]. We use this data to estimate the *competitive ratio*, i.e., $t_{opt} : t_{method}$. To reach ideal network behavior, we performed routing planning on the cluster to ensure each flow occupies a dedicated path for all possible communication patterns. Results are presented in Figure 14.

Table 4: Time share of different models

Model	Batch	Compute	AllReduce	All2All
Resnet50	16	96%	4%	/
Resnet101	16	95%	5%	/
Bert	16	88%	12%	/
VGG16	4	75%	25%	/
PanGu-α	/	67%	7%	26%

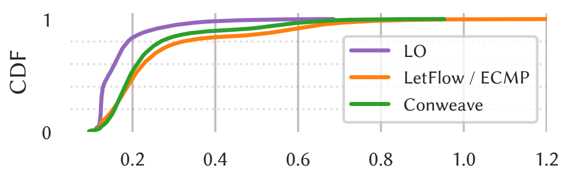


Figure 15: Distribution of FCT (ms)

As shown in Figure 14, $\mathbb{LO}$ achieves the best performance among all evaluated approaches. This is primarily due to its ability to fully leverage fine-grained load balancing through $\mathbb{O}$capability while avoiding retransmissions with a $\mathbb{L}$ network. As previously mentioned, the main challenge for $\mathbb{LO}$ lies not in the network implementation but in potential performance bottlenecks and implementation difficulties related to memory and hardware, which are not evaluated and revealed in our network-wise simulation.

For strict non-reordering $\mathbb{IL}$, performance with Flow-level ECMP is significantly degraded due to flow conflicts. The FCT distribution for Pairwise All-to-All shown in Figure 15, further corroborates this issue. LetFlow and ConWeave's adaptive switching approaches do not yield significantly better results for AI workloads, as their FCT performance shows no significant improvement over ECMP.

For $\mathbb{IO}$, SR performs poorly in scenarios with widespread out-of-order delivery, aligning with our analysis. The inability to distinguish between late and dropped packets results in excessive retransmission overhead. In constrast, NDP-like approaches can offer a more effective solution by packet trimming.

5.2.1 *CBFC / NDP Requires Large Buffer.*

CBFC and NDP exhibit similar performance, both delivering relatively consistent FCTs. For NDP, the receiver-controlled pull pacing mechanism functions similarly to credits in CBFC, self-limiting the number of on-the-fly packets and reducing actual packet loss or congestion. The small retransmission delays in NDP are masked by normal packet transmission. In CBFC, although pausing is employed to ensure lossless transmission, it is rarely observed due to fine-grained load balancing, which minimizes congestion and the need for pausing.

Under infinite buffer conditions, both CBFC and NDP should achieve performance equivalent to $\mathbb{LO}$. However, in our evaluation scenario, where with 64KiB buffer per Queue Pair to support all collective communication peers, buffer throttling is evident and results in an overall slowdown. The average FCT performance appears worse than ECMP.

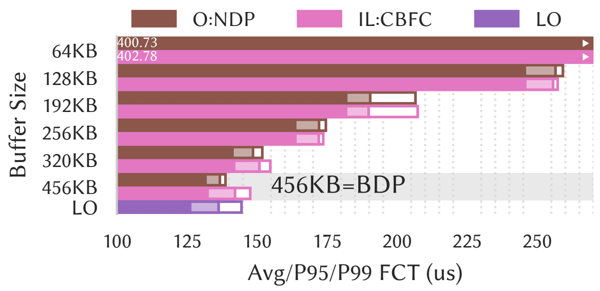


Figure 16: NDP/Credit-based FC have good performance with sufficient buffer

We conducted the same Ring AllReduce experiment while incrementally increasing the buffer size, as shown in Figure 16. When the buffer reaches 1× BDP, performance approaches optimal levels, achieving 95% of $\mathbb{LO}$'s performance.

However, achieving such performance requires significantly more reordering memory, which reduces the number of concurrent flows supported by a single NIC. In our scenario, with a BDP of ∼ 460KB, a CX-5 with 2MB of on-chip memory can only support 4 simultaneous QPs. This limitation heavily restricts the collective communication algorithms and patterns that can be executed across nodes.

We also evaluated these approaches under different topologies, oversubscription scenarios, and link failure conditions (see Appendix B). The results align with our primary findings, leading to the same conclusions.

5.3 Summary

For designing Orderlock-free protocols tailored to AI workloads, as summarized in Figure 17, flow ECMP remains the only off-the-shelf solution, though its performance is suboptimal and lacks scalability. Regular $\mathbb{IL}$ strategies, such as flowlet and adaptive path switching, fail to significantly compensate performance drop in AI scenarios. Similarly, $\mathbb{IO}$ approaches like SR are incompatible with highly out-of-order packet arrivals. NDP and CBFC can be deployed on merchant infrastructure but achieve acceptable performance only at the cost of scalability and traffic pattern flexibility. Compared to CBFC, NDP is less sensitive to congestion from other flows but suffers from instability of lossy networks and requires additional switch modifications. $\mathbb{LO}$ delivers the best performance compared to all, but necessitates an orderless delivery NIC architecture with engineering difficulties.

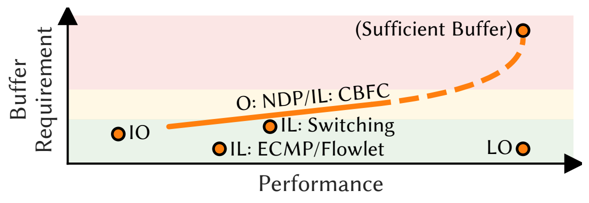


Figure 17: Comparison on Orderlock-free protocols

For future designs, implementing *orderless delivery* to enable $\mathbb{LO}$ or other $\mathbb{O}$ would significantly enhance overall performance. For the bucket sort approach, SRNIC provides an example implementation: each packet can be directly written to memory via DMA. For operations like `RDMA Send` where the sender does not know the remote address, packet headers may need to include additional information, such as memory size and offsets, to allow the NIC to manage buffers and perform DMA. In AI scenarios, NIC further directly writing to GPU memory, rather than writting to host memory and then copy, could be highly beneficial. However, this requires further tight integration with training tasks, framework, and drivers, as GPU memory is hierarchical and more complicated, while locality is critical for optimizing training speed.

For existing commercial NICs with limited on-chip memory, achieving better performance using CBFC or NDP alongside communication algorithms with fewer peers (e.g., Ring) offers a low-cost solution. Although internal closed-source data is unavailable, manufacturers have reported support for *Selective Repeat* [54] and packet trimming [12] in commercial devices. This can expand the potential application range and scale of CBFC and NDP-like approaches.

Other approaches, such as flowlet, sub-flow adaptive switching, and $\mathbb{IO}$, may perform well with traditional DCN environments and traffic, but they are relatively less suitable for AI workloads and requires further tuning.

6 Discussion

Is Reordering Buffer Really Needed for Reordering? Some reorder methods do not explicitly use a reordering buffer, which may lead to confusion about whether Orderlock are still applicable. In such methods like SQR and Conweave, the core component is a packet selector that determines which incoming packets to forward and where they are forwarded to. Out-of-order packets may be located in different queues or circulated through multiple pipelines, where no explicit buffer space can be identified.

However, from the packet's perspective, the buffering mechanism is clear. Regardless of the implementation, early-arriving packets cannot be released before packets that precede them in the original order. The place where these packets stay is the *de facto* reordering buffer and has a corresponding reordering limit.

Slow Receiver Problem. The slow receiver problem refers to non-network bottlenecks on the receiving side, e.g. limited PCIe bandwidth or memory bank conflict, prevent packets received by the NIC from being processed at link speed [6, 23, 60]. While this can fill the buffer and cause packet loss in $\neg\mathbb{L}$ networks, the resulting blockage is distinct from Orderlock. Packets in the buffer not blocked by reordering will eventually be released, and the full buffer situation is not permanent. In contrast, when Orderlock is triggered, packets cannot leave the buffer even without such bottlenecks (see Th. 2 and Appendix A).

However, the cascading propagation effect of pausing in $\mathbb{L}$ networks can exacerbate performance degradation when combined with the slow receiver problem (cf. §3.2). Similar to gray failures, e.g. silent deterioration of link speed or packet forwarding latency, this issue does not directly cause Orderlock but increases the MOA and disrupts assumptions underlying certain $\neg\mathbb{O}$ load-balancing schemes, leading to unexpected Orderlock and errors. For example, flowlets may experience severe delays in paused queues, resulting in out-of-order delivery and unexpected retransmissions after triggering an unidentified Orderlock, which harms performance. The $\mathbb{LO}$ approach discussed earlier relies on lossless transmission and can also be affected. While proper congestion control and load balancing can minimize pause propagation caused by slow receivers, malfunctioning NICs where congestion control fails to self-limit may undermine the performance benefits of $\mathbb{L}$.

To Pause or to Drop? The use of PFC has long been a topic of debate. PFC was initially introduced to mitigate buffer overflow-induced packet loss and improve performance. However, in recent years, its limitations have become apparent, and works such as IRN and NDP have proposed promising alternatives for PFC-free networks. In §5, the performance of CBFC and NDP is shown to be comparable, raising further questions about the necessity of PFC or lossless networks for next-generation HPNs. In an ideal scenario, the delays introduced by PFC (via pausing) and packet trimming (via dropping) when handling bursts are similar: the pausing delay caused by PFC is equivalent to the retransmission delay caused by dropping (illustrated in Appendix C). Our experiments revealed no significant performance difference between the two approaches. However, subtle distinctions may warrant further exploration in future implementations and more detailed scenarios.

Congestion Control for $\mathbb{LO}$ Protocols. In Figure 14, we observe a long tail in FCT for $\mathbb{LO}$, which appears to contradict our expectation that an $\mathbb{O}$ load-balancing algorithm should evenly distribute congestion and delay across flows, reducing variance in arrival times. Upon inspecting packet-sending behavior during transmission, we find that the tail is caused by the congestion control (CC) algorithm's response to Congestion Notification Packets. Specifically, for Butterfly AllReduce running on $\mathbb{LO}$, approximately 26% of flows are tagged with ECN as specified in RoCEv2. The sender's CC algorithm (in our case, DCQCN [75]) prematurely throttles the sending rate, degrading performance.

Virtually all current CC algorithms based on explicit congestion sensing, including DCQCN and HPCC [40], suffer from this issue: ECN only signals congestion on a single path, yet the sending rate is throttled across all paths. Alternative approaches, such as RTT-based Timely [47], may offer partial solutions. However, Timely adjusts the sending rate based on timestamps collected from multiple paths, making it challenging to accurately estimate overall network congestion. Designing effective CC algorithms for multipath environments is critical for optimizing $\mathbb{O}$ protocol performance and needs to be elaborated thoroughly in the future.

Implementation of $\mathbb{LO}$ Protocols. Our simulations show that LO protocols deliver the best performance, but this comes at the cost of significant engineering complexity. The primary challenge lies in implementing flow tracking entirely on the NIC. To handle general OoO packet tracking, the NIC must maintain heavy context. Unlike SR which only tracks a limited window, a general $\mathbb{O}$ implementation requires a full-sized bitmap (cf. §3.2), whose memory and processing overhead is unacceptable on current NIC architecture. Theoretically, replacing bitmaps with simpler mechanisms, e.g. counters, could solve this, but this hinges on an extremely low packet loss rate to be effective, which is difficult to achieve with current devices.

In contrast, application-participated reordering is far easier to implement. This approach, however, shifts complexity to the host CPU, burdening the userspace with tasks such as memory partitioning, flow state management, and reordering logic, thereby introducing significant computational overhead.

7 Conclusion

In this paper, we introduce a new type of deadlock, termed Orderlock. We show that the simultaneous achievement of In-order Delivery, Lossless Transmission, and Out-of-order Capability can trigger Orderlock and proved the necessary and sufficient conditions for its occurrence. We show that operating under an Orderlock-risky network is impractical and compare Orderlock-free protocol implementations in the scenario of AI workloads. We find that orderless delivery is important supporting future AI-ready HPNs.

Acknowledgments

We sincerely thank our shepherd James Hongyi Zeng and the anonymous reviewers for their constructive feedback. This work was supported by the National Key R&D Program of China (No. 2024YFB2906700) and the NSF China (No. 62272292).

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08)*. Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/1402958.1402967

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 503–514. doi:10.1145/2619239.2626316

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/1851182.1851192

[4] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. Association for Computing Machinery, New York, NY, USA, 267–280. doi:10.1145/1879141.1879175

[5] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. 1997. Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. *IEEE Transactions on Parallel and Distributed Systems* 8, 11 (1997), 1143–1156. doi:10.1109/71.642949

[6] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding Host Network Stack Overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 65–77. doi:10.1145/3452296.3472888

[7] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. 2013. Per-Packet Load-Balanced, Low-Latency Routing for Clos-Based Data Center Networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. Association for Computing Machinery, New York, NY, USA, 49–60. doi:10.1145/2535372.2535375

[8] Peirui Cao, Wenxue Cheng, Shizhen Zhao, and Yongqiang Xiong. 2024. Network Load Balancing with Parallel Flowlets for AI Training Clusters. In *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing (NAIC '24)*. Association for Computing Machinery, New York, NY, USA, 18–25. doi:10.1145/3672198.3673794

[9] 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)*. 1–11. doi:10.1109/ICNP52444.2021.9651977

[10] 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 (Dec. 2023), 2870–2885. doi:10.1109/TNET.2023.3265276

[11] Chuhao Chen, Jiarui Ye, Yongbo Gao, Sen Liu, and Yang Xu. 2024. HF^2T: Host-Based Flowlet Fine-Tuning for RDMA Load Balancing. In *Proceedings of the 8th Asia-Pacific Workshop on Networking (APNet '24)*. Association for Computing Machinery, New York, NY, USA, 9–15. doi:10.1145/3663408.3663410

[12] Xiaoqi Chen, Shay Vargaftik, and Ran Ben Basat. 2024. When ML Training Cuts Through Congestion: Just-in-Time Gradient Compression via Packet Trimming. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets '24)*. Association for Computing Machinery, New York, NY, USA, 177–185. doi:10.1145/3696348.3696880

[13] 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)*. 19–36. https://www.usenix.org/conference/nsdi20/presentation/cheng

[14] Ultra Ethernet Consortium. 2023. *Overview of and Motivation for the Forthcoming Ultra Ethernet Consortium Specification*. Technical Report. https://ultraethernet.org/wp-content/uploads/sites/20/2023/10/23.07.12-UEC-1.0-Overview-FINAL-WITH-LOGO.pdf

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. doi:10.18653/v1/N19-1423

[16] Advait Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the Impact of Packet Spraying in Data Center Networks. In *2013 Proceedings IEEE INFOCOM*. IEEE, Turin, Italy, 2130–2138. doi:10.1109/INFCOM.2013.6567015

[17] Xiaodong Duan, Jieyu Li, Weiqiang Cheng, Han Li, Ruixue Wang, and Haojie Wang. 2024. Challenges and key technologies of new Ethernet for intelligent computing center. *Telecommunications Science* 40, 6 (July 2024), 146–159. doi:10.11959/j.issn.1000-0801.2024171

[18] Kevin R. Fall and W. Richard Stevens. 2012. *TCP/IP Illustrated*. Addison-Wesley Professional.

[19] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Allyn Romanow. 1996. *TCP Selective Acknowledgment Options*. Request for Comments RFC 2018. Internet Engineering Task Force. doi:10.17487/RFC2018

[20] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. doi:10.1145/3651890.3672233

[21] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: A Practical Reordering Resilient Network Stack for Datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/2901318.2901334

[22] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 225–238. doi:10.1145/3098822.3098839

[23] 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 (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 202–215. doi:10.1145/2934872.2934908

[24] 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 (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. doi:10.1145/3098822.3098825

[25] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based Load Balancing for Fast Datacenter Networks. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 465–478. doi:10.1145/2829988.2787507

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs] doi:10.48550/arXiv.1512.03385

[27] Jinbin Hu, Jiawei Huang, Wenjun Lv, Yutao Zhou, Jianxin Wang, and Tian He. 2019. CAPS: Coding-Based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center. *IEEE/ACM Transactions on Networking* 27, 6 (Dec. 2019), 2338–2353. doi:10.1109/TNET.2019.2945863

[28] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2016. Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 92–98. doi:10.1145/3005745.3005760

[29] Jiawei Huang, Wenjun Lyu, Weihe Li, Jianxin Wang, and Tian He. 2021. Mitigating Packet Reordering for Random Packet Spraying in Data Center Networks. *IEEE/ACM Transactions on Networking* 29, 3 (Feb. 2021), 1183–1196. doi:10.1109/TNET.2021.3056601

[30] Huawei. 2023. CloudEngine 12800 Manual: Dynamic Load Balancing. https://support.huawei.com/hedex/api/pages/EDOC1100136523/31180BRY/13/resources/dc/dc_cfg_low-latency_0008.html

[31] Huawei. 2023. CloudEngine 9800 Manual: PFC Deadlock Prevention. https://support.huawei.com/hedex/api/pages/EDOC1100366031/AEN0412J/02/resources/dc/dc_fd_dcb_pfc_0003_copy.html

[32] IBM. 2024. AIX 7.1: EtherChannel Load-Balancing Options. https://www.ibm.com/docs/en/aix/7.1?topic=aggregation-etherchannel-load-balancing-options

[33] IEEE Standards Board. 1997. IEEE Standards for Local and Metropolitan Area Networks: Specification for 802.3 Full Duplex Operation. https://ieeexplore.ieee.org/document/9768038/

[34] IEEE Standards Board. 2011. IEEE Standard for Local and Metropolitan Area Networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 17: Priority-based Flow Control. doi:10.1109/IEEESTD.2011.6032693

[35] Raj Joshi, Cha Hwan Song, Xin Zhe Khooi, Nishant Budhdev, Ayush Mishra, Mun Choon Chan, and Ben Leong. 2023. Masking Corruption Packet Losses in Datacenter Networks with Link-local Retransmission. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 288–304. doi:10.1145/3603269.3604853

[36] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter {RPCs} Can Be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 1–16. https://www.usenix.org/conference/nsdi19/presentation/kalia

[37] Simon Kassing, Asaf Valadarsky, and Ankit Singla. 2016. Netbench. https://github.com/ndal-eth/netbench

[38] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *2008 International Symposium on Computer Architecture*. 77–88. doi:10.1109/ISCA.2008.19

[39] 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. doi:10.1109/ICNP59255.2023.10355582

[40] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58. doi:10.1145/3341302.3342085

[41] Jingling Liu, Jiawei Huang, Wenjun Lv, and Jianxin Wang. 2022. APS: Adaptive Packet Spraying to Isolate Mix-Flows in Data Center Network. *IEEE Transactions on Cloud Computing* 10, 2 (April 2022), 1038–1051. doi:10.1109/TCC.2020.2985037

[42] Xiyuan Liu, Yang Liu, Jingyi Cheng, Ximeng Liu, and Shizhen Zhao. 2025. FauTE: Fault-tolerant Traffic Engineering in Data Center Network. In *Proceedings of the 9th Asia-Pacific Workshop on Networking (APNET '25)*. Association for Computing Machinery, New York, NY, USA, 214–219. doi:10.1145/3735358.3735364

[43] Ximeng Liu, Shizhen Zhao, Yong Cui, and Xinbing Wang. 2024. FIGRET: Fine-Grained Robustness-Enhanced Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 117–135. doi:10.1145/3651890.3672258

[44] Jonathan Looney. 2019. *CVE-2019-11478*. Technical Report CVE-2019-11478. https://nvd.nist.gov/vuln/detail/cve-2019-11478

[45] Yuanwei Lu, Guo Chen, Zhenyuan Ruan, Wencong Xiao, Bojie Li, Jiansong Zhang, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2017. Memory Efficient Loss Recovery for Hardware-based Transport in Datacenter. In *Proceedings of the First Asia-Pacific Workshop on Networking (APNet '17)*. Association for Computing Machinery, New York, NY, USA, 22–28. doi:10.1145/3106989.3106993

[46] Mellanox. 2014. InfiniBand Credit-Based Link-Layer Flow-Control. https://www.ieee802.org/1/files/public/docs2014/new-dcb-crupnicoff-ibcreditstutorial-0314.pdf

[47] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 537–550. doi:10.1145/2785956.2787510

[48] 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 (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 313–326. doi:10.1145/3230543.3230557

[49] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 221–235. doi:10.1145/3230543.3230564

[50] Mohammad Noormohammadpour and Cauligi S. Raghavendra. 2018. Datacenter Traffic Control: Understanding Techniques and Tradeoffs. *IEEE Communications Surveys & Tutorials* 20, 2 (2018), 1492–1525. doi:10.1109/COMST.2017.2782753

[51] NVIDIA. 2023. Nvidia ConnectX-7: NDR 500G Infiniband Adapter Card. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/infiniband-adapters/infiniband-connectx7-data-sheet.pdf

[52] NVIDIA. 2023. RDMA Aware Networks Programming User Manual: Transport Modes. https://docs.nvidia.com/networking/display/RDMAAwareProgrammingv17/Transport+Modes

[53] NVIDIA. 2024. GPUDirect RDMA. https://docs.nvidia.com/cuda/gpudirect-rdma/

[54] NVIDIA. 2024. Linux Kernel Upstream Release Notes v6.7: Changes and New Features History. https://docs.nvidia.com/networking/display/Kernelupstreamv67/Changes+and+New+Features+History

[55] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 66–85. doi:10.1145/3544216.3544265

[56] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 691–706. doi:10.1145/3651890.3672265

[57] Ting Qu, Raj Joshi, Mun Choon Chan, Ben Leong, Deke Guo, and Zhong Liu. 2019. SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. 1–12. doi:10.1109/ICNP.2019.8888055

[58] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals Are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. doi:10.1145/3544216.3544226

[59] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra (Eds.). Springer, Berlin, Heidelberg, 1–9. doi:10.1007/978-3-540-24685-5_1

[60] Alireza Sanaee, Farbod Shahinfar, Gianni Antichi, and Brent E. Stephens. 2022. Backdraft: A Lossless Virtual Switch That Prevents the Slow Receiver Problem. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1375–1392. https://www.usenix.org/conference/nsdi22/presentation/sanaee

[61] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. 2003. *RTP: A Transport Protocol for Real-Time Applications.* Request for Comments RFC 3550. Internet Engineering Task Force. doi:10.17487/RFC3550

[62] sh-beta. 2009. Answer to "When to Turn TCP SACK Off?". https://serverfault.com/a/11002

[63] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (Nov. 2020), 67–73. doi:10.1109/MM.2020.3016891

[64] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs] doi:10.48550/arXiv.1409.1556

[65] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 708–721. doi:10.1145/3387514.3405897

[66] 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 (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 816–831. doi:10.1145/3603269.3604849

[67] Cisco switch. 2014. IP Switching: Cisco Express Forwarding Configuration Guide, Cisco IOS Release 15S. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipswitch_cef/configuration/15-s/isw-cef-15-s-book/isw-cef-load-balancing.html

[68] Min Yee Teh, Shizhen Zhao, Peirui Cao, and Keren Bergman. 2023. Enabling Quasi-Static Reconfigurable Networks With Robust Topology Engineering. *IEEE/ACM Transactions on Networking* 31, 3 (June 2023), 1056–1070. doi:10.1109/TNET.2022.3210534

[69] The NS-3 Team. 2008. The NS-3 Network Simulator. https://www.nsnam.org

[70] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[71] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. 2023. SRNIC: A Scalable Architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1–14. https://www.usenix.org/conference/nsdi23/presentation/wang-zilong

[72] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, Chen Li, Ziyan Gong, Yifan Yao, Xinjing Huang, Jun Wang, Jianfeng Yu, Qi Guo, Yue Yu, Yan Zhang, Jin Wang, Hengtao Tao, Dasen Yan, Zexuan Yi, Fang Peng, Fangqing Jiang, Han Zhang, Lingfeng Deng, Yehong Zhang, Zhe Lin, Chao Zhang, Shaojie Zhang, Mingyue Guo, Shanzhi Gu, Gaojun Fan, Yaowei Wang, Xuefeng Jin, Qun Liu, and Yonghong Tian. 2021. PanGu-a: Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation. arXiv:2104.12369 [cs] doi:10.48550/arXiv.2104.12369

[73] 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)*. 1–9. doi:10.1109/ISPA-BDCloud-SocialCom-SustainCom59178.2023.00033

[74] 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)*. USENIX Association, Boston, MA, 663–683. https://www.usenix.org/conference/nsdi23/presentation/zhao-shizhen

[75] 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 (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. doi:10.1145/2785956.2787484

Appendix

Appendices are supporting material that has not been peer-reviewed.

A Proof of Necessary and Sufficient Conditions

Proof of necessary condition (Theorem 1):

Proof. We prove Theorem 1 by contrapositive: that relaxing any three of $\mathbb{I}$, and $\mathbb{L}$, and $\mathbb{O}$ causes Orderlock to release or not happening.

$\neg\mathbb{I} \rightarrow \neg$Orderlock: If $\mathbb{I}$n-order Delivery is not met, the packet p' having $\text{idx}_P(p') > \text{idx}_P(p)$ can be sent out even when p is not sent, and in this situation R's buffer is released and standstill no longer holds.

$\neg\mathbb{L} \rightarrow \neg$Orderlock: If $\mathbb{L}$ossless Transmission is not met, the packet $p' \in R$ can be dropped to free R's buffer, standstill no longer holds.

$\neg\mathbb{O} \rightarrow \neg$Orderlock: If $\mathbb{O}$ut-of-order Capability is not met, sequence P' having $MOA > n$ does not exist in network.

Thus Orderlock $\rightarrow \mathbb{I} \wedge \mathbb{L} \wedge \mathbb{O}$.[1] □

[1] $\wedge$ is the logical AND symbol.

Proof of sufficient condition (Theorem 2):

Proof. We show that 1) for given situation, an empty reordering R with reordering limit n will finally turn into that $||R|| = n$ with the arrival sequence P and $MOA > n$ and 2) in this situation R is standstill.

By MOA definition, $\exists S[1 \ldots m]$ being a sub-sequence of P', s.t. $S[m] = p$, $OA(p) \geq n$. By OA definition, $m \geq n + 1$, and we have that $S[1 \ldots m-1]$ containing at least n packets p that $\mathrm{idx}_P(p') > \mathrm{idx}_P(p)$ and $\mathrm{idx}_{P'}(p') < \mathrm{idx}_{P'}(p)$.

With condition $\mathbb{I}$ and $\mathbb{L}$, the R, next to send packet i, and next to arrive packet j, works as defined in the following transition conditions:

$$\begin{cases} R, i, j \to R \cup \{P'[j]\}, i, j+1 & \text{if } |R| < n \\ R, i, j \to R \backslash \{P[i]\}, i+1, j & \text{if } \min\{R\} = P[i] \\ R, i, j \not\to & \text{Otherwise} \end{cases}$$

(1) We show that the configuration will turn into Orderlock state. For given p, assume the R ends in state that does not met $\forall p' \in R, \mathrm{idx}_P(p') > \mathrm{idx}_P(p)$ and $\mathrm{idx}_{P'}(p') < \mathrm{idx}_{P'}(p)$. i.e. $\exists p^* \in R, \mathrm{idx}_P(p^*) < \mathrm{idx}_P(p)$ or $\mathrm{idx}_{P'}(p^*) > \mathrm{idx}_{P'}(p)$. For $\mathrm{idx}_{P'}(p^*) > \mathrm{idx}_{P'}(p)$ this is impossible as p^* will not enter R unless p entered. For $\mathrm{idx}_P(p^*) < \mathrm{idx}_P(p)$, either it will be released out according to the 2nd transition rule, or $\exists p'', \mathrm{idx}_P(p'') < \mathrm{idx}_P(p^*) < \mathrm{idx}_P(p)$ which is $P[i]$ and p'' also meet $OA(p'') > n$. In this case, p'' triggers Orderlock earlier.

(2) We show that the situation doesn't progress. In Orderlock state, $|R| = n$, then the first option does not meet, and $\min\{R\} \neq P[i]$ as the packet p have that $p := P[i] := P'[I]$ and $I > j$. In this state it falls into $R, i, j \not\to$ transition condition.

□

B Supplementary Evaluations

In this section, we discuss some less occurred scenarios to complement the experimental part of the article. We compare different topologies, over-subscribed networks i.e. the throughput of core layer is limited, and unbalanced network caused by random link failure. Overall, there are no major performance differences from the analysis in the body. We take the FCT of butterfly as an example and the result is shown in figure 18.

Topologies. We choose another clos-like topology Leaf-Spine with up-down routing, in order to prevent PFC deadlocks which may affect the observability. Compared to 3-layer Fat-Tree, Leaf-Spine has fewer layers and thus having smaller network diameter, reducing the delay at expense of scalability.

Noticing that the overall FCT tail is shorter for the leaf-spine topology compared to fat-tree, and the FCT improves for those fine grained load balancing protocols. While for flow-level load balancing protocols such as Letflow and ECMP, the FCT is worse. The overall latency has decreased due to less layers and smaller network diameter. However, deploying leaf-spine requires switches with higher fanout comparing to fat-tree when providing network at the same scale. Also, the equal cost path number are reduced and the probability of conflicts in flow-level load balancing is higher, affecting the FCT. Due to higher flow conflict rate, rerouting congested flow, as shown as Conweave in the figure, has better performance comparing to original flow level solutions.

Over-subscription. When network scales up, the cost to maintain full bandwidth can be very high, and allowing over-subscription provides an economical way. Some of the spine-less network architectures proposed in recent years, such as dragonfly, and expander graph, suffer from similar bottlenecks.

Currently, the CapEx and OpEx of networking takes a small part comparing to computing units, and operations like AlltoAll has high bandwidth requirement which do not perform well under oversubscribed networks. For current clusters, it is more cost-effective to use a non-over-subscription network. As individual data center power consumption reaches the infrastructure limit. However, performance with limited networks will further impact LLM training efficiency and scalability in the near future.

In our experiment, we shrink the link bandwidth connecting the aggregation layer and the core layer according to the over-subscription ratio. With a low overall bandwidth requirement (butterfly AllReduce in the figure), flow/flowlet level load balancing suffers due to throttled link. For the lossy network, large packet loss happens at the bottleneck while CC algorithm failed to react as fast, resulting in longer FCT. In the 1:2 case, we observed slightly worse performance of NDP compared to CBFC, which is due to overfull buffer and packet are dropped instead of trimmed. In this situation timeout retransmission is triggered and having long FCT.

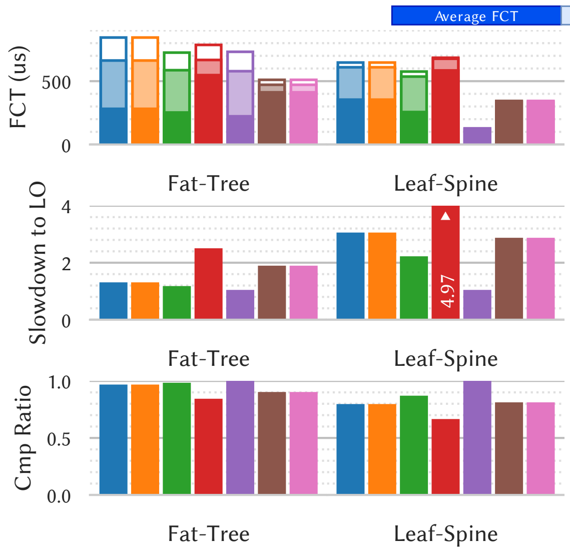
Link failure. Link failure treatment is gaining more research attention nowadays. In addition to the general network error as physical failure, interface overheating, etc., due to the widespread use of RoCE for AI tasks and the GPU/NPU is more prone to failure, PFC deadlocks and Slow receiver-triggered PFC storms can also lead to similar failures.

For small-scale clos networks, by route planning, e.g., handpicking port numbers, using ACL directing flow to port, etc., it is possible to achieve congestion-free with given collective communication pattern using existing ECMP (the time share of model training was measured in such network). One problem with such approach is that if link failure occurs, route planning can be completely disrupted, making it less robust.

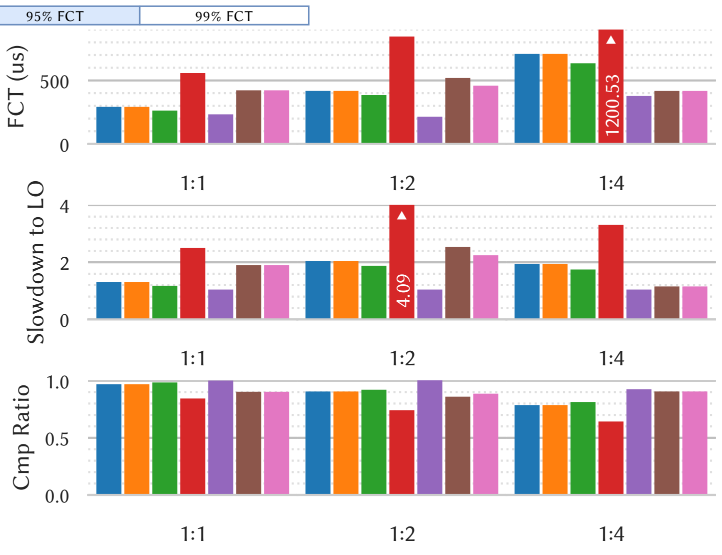
We also evaluated the link failure scenario under both topologies for 10% and 20% offline links. With mild workload, no significant impact was observed, with small and proportional FCT increase for all protocols. That's because we've used a *uniform* ECMP hashing strategy, where was already a lot of flow conflict even without link failures. Leaf-Spines are more affected by link failures than Fat-Trees, also because Leaf spines have relatively fewer paths to choose from, and the overall bandwidth may be shrunken when there is a link failure.

C Pausing/Dropping Performance

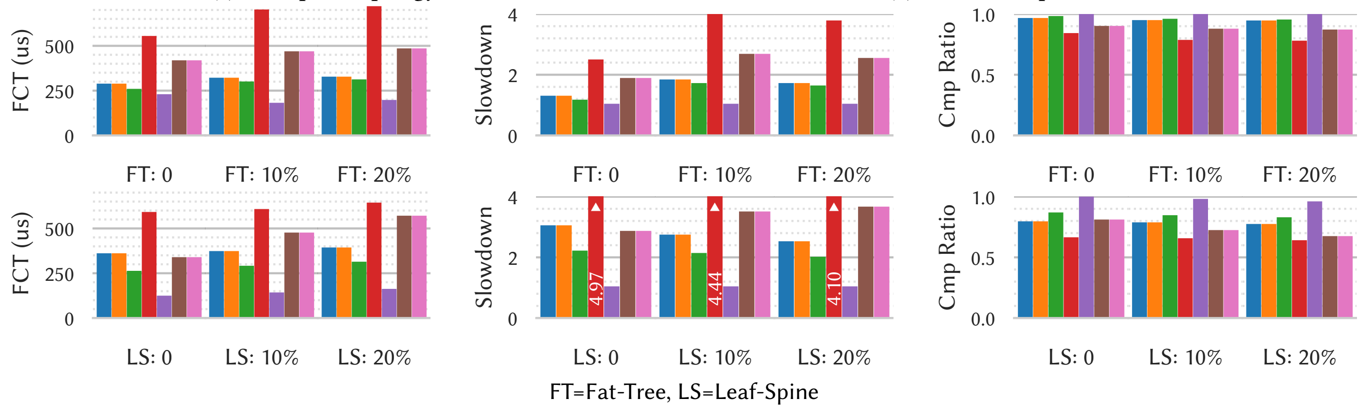
Take a simplified scenario as shown in fig , PFC has ideal XOFF and XON thresholds where no buffer is wasted and won't cause packet loss, and the flow is long enough that retransmission of NDP is pipelined with regular transmission without extra timeout. It can



(a) Leaf-spine Topology



(b) Over subscription

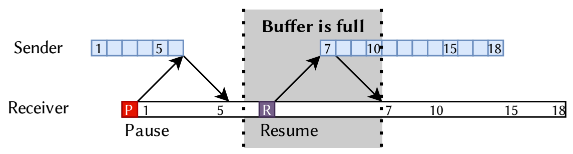


(c) Link failure

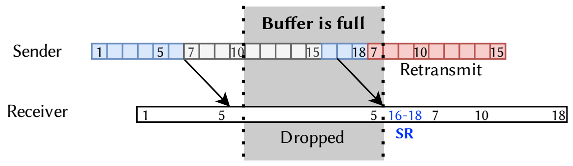
IL:ECMP
IL:LetFlow
IL:Conweave
IO
LO
O:NDP
IL:CBFC

For overflowed bars, actual value is marked on the root.

Figure 18: Supplementary Comparison



(a) PFC Pausing



(b) NDP Drop

Figure 19: Pausing and Dropping Performance

be seen from the figure that PFC delay caused by PFC is the same as the retransmission delay of NDP.

For NDP and other lossy protocols, if packets at the end of the flow are dropped, retransmission time cannot be covered by regular sending and must take at least 1 RTT, which results in extra delay and longer FCT. On the other hand, PFC needs time to propagate and the pause / resume thresholds might not be as accurate, also result in extra delay. Meanwhile, pausing can propagate to upstream switches and affecting extra flows, while full loss of packet in NDP is also not measured. In the $\mathbb{O}$ situation, both pausing and packet loss/trimming are rare and the performance loss are not affected in our previous experiments. Further analysis and experience is needed to judge between two approaches.