PRO: Deterministic Load Balancing for AI Datacenter Network

Yiran Zhang[1], Qingkai Meng[2], Yadong Liu[3], Xiangyu Han[2], Xiao Ma[1], Shangguang Wang[1]
[1]Beijing University of Posts and Telecommunications
[2]Nanjing University [3]Tsinghua University

***Abstract*—High-demand Large Language Model (LLM) services require high-bandwidth, scalable AI datacenter networks, where load balancing is critical to end-to-end application-perceived communication performance. Existing datacenter network research primarily relies on randomized load balancing mechanisms to handle bursty, dynamic workloads. However, we argue that deterministic load balancing can be revitalized with significant potential, due to the synchronized traffic patterns and uniform packet size distributions inherent in LLM training workloads. In this paper, we propose PRO, a near-optimal load balancing mechanism implemented on RDMA NICs. By orchestrating a deterministic round-robin order on RNICs with consideration of both intra-QP and inter-QP traffic, PRO realizes balanced queue distributions at both switch uplinks and downlinks. We implement a prototype of PRO using Intel FPGA NIC. Extensive evaluations show that PRO reduces 30%∼48% completion time (JCT) slowdown in the testbed, and up to 83.6% JCT slowdown compared to state-of-the-art DRILL, ConWeave, and Adaptive Routing.**

***Index Terms*—Load Balancing, RDMA, LLM**

I. Introduction

Nowadays, Large Language Models (LLMs), ranking among the largest and most computationally intensive Deep Neural Networks (DNNs), are widely developed in data centers. The latest LLMs, such as OpenAI GPT5 [24], Facebook LLaMA [10], and DeepSeek V3 [15], are estimated to have trillions of parameters and require several months for training. Reducing the training time of LLM is critical as it can directly improve the iteration speed of the model, subsequently optimizing the model quality and improving user experience.

Network load balancing is vital to pursuing the ultimate LLM training performance. Under relatively stable high-load conditions dominated by collective communication patterns (e.g., AllReduce, AllGather), more even load distribution across multiple equivalent links in data center network topologies can fully exploit abundant link bandwidth and significantly accelerate communication efficiency.

Network load balancing has long attracted substantial research attention in traditional datacenters. Existing solutions predominantly adopt randomized load balancing strategies, which introduce randomness into each load-balancing decision [11], [9] while also accounting for dynamic network conditions [2], [27]. For example, DRILL [11] employs "power of two choices" randomized algorithm to select the outgoing link for each packet based on switch queue occupancies. However, we argue that the emergence of synchronized traffic patterns and uniform packet-size distributions in LLM training workloads provides new opportunities to revisit and rethink load-balancing design. In traditional datacenters, randomized load balancing is optimized for dynamic, bursty traffic and heavy-tailed packet-size distributions. In contrast, LLM workloads exhibit far more regular and predictable characteristics, which enable optimal load balancing via deterministic strategies. Our experimental observations demonstrate that, under synchronized traffic patterns and uniform packet-size distribution, random packet spraying tends to induce traffic bursts and queue buildup, thereby degrading tail latency (§ II-B).

In this paper, we develop a near-optimal load balancing mechanism on RDMA NICs called PRO. The basic idea of PRO is to orchestrate a deterministic packet spraying order at RNICs to achieve low and uniform queue distribution at network switches. Concretely, PRO maps each Queue Pair (QP) with an equivalent path set according to the destination Top-of-Rack (ToR) switch. PRO carefully staggers the round-robin order at both the intra-QP level and inter-QP level to avoid micro load imbalance at switch uplinks and downlinks. By considering the hardware architecture of RNICs with multiple QPs, PRO is amenable to hardware implementation. We prove that PRO can guarantee a bounded queue length at the network core[1]. What's more, PRO well handles out-of-order packets and lost packets due to network failures. To avoid unnecessary retransmissions, PRO leverages the observation that there is indeed limited out-of-order degree under the orchestrated round-robin, and introduces an out-of-order window to efficiently distinguish out-of-order arrivals and packet losses. Thus, lost packets are retransmitted timely while out-of-order packets can be directly DMAed.

We implement a prototype of PRO using Intel FPGA NIC (∼8K lines of Verilog excluding basic RDMA semantics). PRO only consumes 21K ALMs, 34.7K registers, and 53KB RAMs at NIC. We evaluate PRO using a 32-GPUs testbed and 128/1024-node simulations against state-of-the-art alternatives. Experiments show that PRO reduces 30%∼48% completion time (JCT) slowdown compared to ECMP in the testbed, and 50.9%∼83.6% JCT slowdown for LLM model training compared to state-of-the-art approaches including DRILL [11], ConWeave [25], Adaptive Routing [5], and ECMP.

[1]The network core for LB refers to the remaining links other than server-ToR links (e.g., ToR uplinks and Aggregation downlinks).

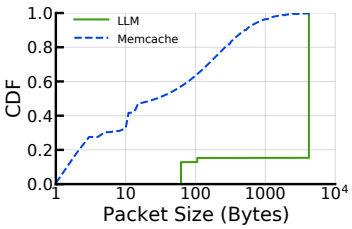
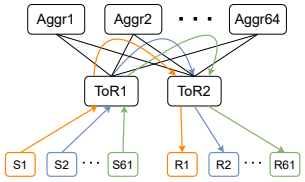
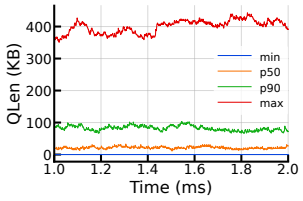


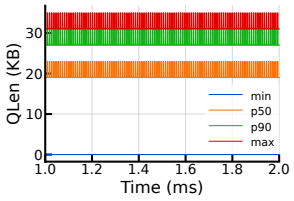
Figure 1: CDF of packet size.



(a) Topology



(b) Queue length with random per-packet spraying



(c) Queue length with an ideal deterministic round-robin

Figure 2: [Simulation] ToR1 switch queue lengths.

II. Background and Motivation

A. Network Load Balancing in Traditional Datacenters

Traditional datacenters mainly host Online Data-Intensive (OLDI) applications, which are prone to generate bursty and heavy-tailed traffic [3]. The goal of network load balancing is distributing these traffic evenly across multiple equivalent paths. There are mainly two categories of existing works:

Deterministic load balancing. Deterministic load balancing routes traffic based on fixed rules (e.g., hash and round-robin). For example, ECMP assigns flows based on hash values of header fields. However, ECMP suffers from large flow collisions when the flow size distributions are heavy-tailed [12]. As a result, dynamic and bursty traffic of OLDI application workloads can lead to inefficient utilization of bandwidth. DRB [6] proposes to use per-packet round-robin for TCP applications, but also hard to achieve optimal performance under skewed and varying traffic. On the whole, the efficiency of deterministic load balancing is constrained by the characteristics of traffic loads.

Randomized load balancing. To overcome the limitations of deterministic load balancing, many randomized load balancing are proposed in traditional datacenters. Randomized load balancing adds randomness for each load balancing decision [11], [9] and may also consider dynamic network conditions [2], [27]. For example, DRILL [11] employs "power of two choices" randomized algorithm to select the outgoing link for each packet based on switch-local queue occupancies. Broadcom switches support Adaptive Routing (AR) [5], which sprays packets at each switch according to the link utilization. Recently, Ultra Ethernet Consortium [8] also proposes to adopt per-packet spraying [9] at NICs to reduces flows collisions and thus improve link utilization. By introducing randomness, this line of works can better adapt to OLDI application workloads and enhance overall network efficiency.

B. Rethinking Load Balancing in AI datacenters

Unlike traditional datacenters, AI datacenters typically serve LLM workloads and thus have unique characteristics.

Highly synchronized traffic patterns. To achieve low-latency and high-throughput communication in distributed LLM training, RDMA technology has gained widespread adoption, as it offloads transport protocol processing to hardware. Building on RDMA-based communication, LLMs further heavily use collective communication library (e.g., NCCL [20]) to facilitate efficient data exchange across distributed GPUs. This results in synchronized traffic patterns across multiple GPUs. For instance, in data-parallel distributed training, after each batch of data is processed, all GPUs must synchronize gradients and calculate their average value via the AllReduce communication primitive. This synchronization occurs at fixed stages of each iteration, leading to periodic traffic bursts [22]. Moreover, all nodes involved in the synchronization must complete the communication simultaneously. Thus, the timing of transmitting intermediate gradients/activations aligns with the progress of the preceding and subsequent stages of the training pipeline.

Uniform packet size distribution. Traditional OLDI applications have a skewed packet size distribution, while LLM traffic endows a more uniform packet size distribution. Figure 1 shows that the packet size distribution of a Memcached workload and a LLM training workload collected from a production datacenter network. For the Memcached workload, over 90% of packets have sizes smaller than 1000 Bytes and varies greatly. On the contrary, the packet sizes in the LLM training workload are limited to a few values. This is because, in distributed training, gradients are usually optimized for network transmission by being sized to fit efficiently within the MTU to avoid fragmentation and improve performance. As a result, a uniform packet size can facilitate packet-level load balancing.

We argue that synchronized traffic patterns and uniform packet-size distributions in AI datacenters presents new opportunities to rethink load balancing. In traditional datacenters, randomized packet-level load balancing is optimized for dynamic, bursty traffic and heavy-tailed packet size distributions. However, AI datacenters exhibit far more regular workload characteristics, enabling the realization of optimal load balancing through deterministic strategies.

To understand the potential of deterministic load balancing under the LLM workloads, we conduct NS3 simulations under a typical scenario as shown in Figure 2(a). We generate synchronized traffic (with same packet sizes) at S1 to S61. S1 to S61 sends traffic to another 61 servers under switch ToR2 with no subscription and in-network congestion. ToR1 randomly sprays each packet to equivalent paths. Figure 2(b) shows the ToR1 switch queue lengths distributions of the ToR1 switch at the 50th, 90th, and 100th percentiles. With random per-packet spraying, the maximum queue length fluctuates greatly (up to 400KB). This indicates that under synchronized traffic patterns, random per-packet spraying is prone to induce

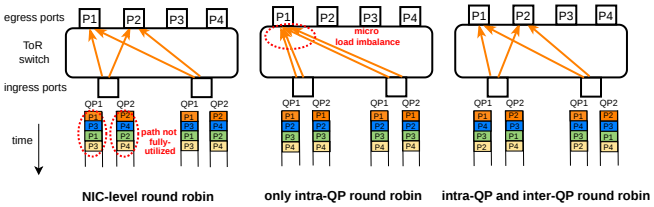


Figure 3: Different round robin mechanisms.

traffic bursts and queue buildup, thereby increasing tail latency. For comparison, we implement an ideal deterministic round-robin load balancing scheme at ToR1. In contrast, this scheme can achieve the maximum queue length to less than 40 KB, demonstrating that deterministic load balancing has the potential to evenly distribute switch queue loads and maintain low tail latency.

III. PRO Design

We propose PRO, a deterministic load balancing for LLM workloads on RDMA NICs (RNICs). The design goals of PRO are as follows.

• Fine-grained load balancing: To achieve near-optimal performance under typical LLM traffic patterns, PRO should be able to maintain low queue lengths at network switches (both switch uplinks and downlinks).

• Robust to packet losses: PRO should be able to handle out-of-order packets and lost packets due to network failures.

• Low-cost at RNICs: PRO should consider the hardware architecture of RNICs to ensure it is amenable to efficient implementation, i.e., maintaining compatibility with the native QP scheduler of RNICs while incurring low memory overhead.

A. Orchestrated Round Robin

We first present a orchestrated round robin mechanism to achieve uniform load balancing performance under a ideal per-packet scheduling model in RNICs. Then, we introduce how the proposed mechanism adapts to a more realistic batch scheduling model in RNICs.

RNIC basis. RNICs host multiple Queue Pairs (QPs), with each QP representing a connection. Specifically, RNICs employ a QP scheduler to fetch packets from different QPs equitably [14] at each round. The QP scheduler handles packets of one QP at a time. RNIC leverages a QPC per QP to track QP/connection related contexts. Besides, as the traffic patterns of collective communications in LLM workloads are synchronized and concurrent, the number of QPs in each RNIC is relatively stable over time.

Conceptual per-destination equivalent path set. Since multiple QPs may have *different destinations*, PRO introduces a conceptual equivalent path set per destination Top-of-Rack (ToR) switch. For each destination ToR, the equivalent paths are defined as the combination of the uplink ID of ToR switches and downlink IDs of aggregation switches.

For example, the egress ports ID of the uplinks of ToRs and aggregation switches are denoted as $I_t(0, ..., n-1)$ and $I_a(0, ..., m-1)$, respectively. Then an available path is identified as (I_t, I_a), which is denoted by $PathID$. Note that given a topology with n ToR switches and m aggregation switches, $PathID$ can be calculated in advance[2]. In this way, PRO does not maintain per-path states and only determines $PathID$ for each packet.

Algorithm 1 Orchestrated Round Robin.

Require: M paths, global counter C with initial value zero, active QP list QPs with the same destination ToR, the number of active QPs P.

1: **function** NIC_Send_Pkts(QPs)
2: **for** QP_i in QPs **do**
3: CALC_PATH(QP_i, P)
4: Send a packet in QP_i to path $QP_i.path$
5: **function** CALC_PATH(QP_i, $span$)
6: **if** $span$ is even **then**
7: $span \leftarrow span + 1$
8: **if** $QP_i.is\_inited$ **then**
9: $QP_i.path \leftarrow (QP_i.path + span) \mod M$
10: **else** //the first packet of QP_i
11: $QP_i.path \leftarrow C \mod M$
12: $C \leftarrow QP_i.path + 1$

Orchestrated round-robin within QP scheduler. Given the per-destination equivalent path set, PRO can map each QP with an equivalent path set according to the destination ToR. Then PRO manipulates the sending path of each packet within the same equivalent path set in a orchestrated round robin manner:

Intra-QP round robin to balance downlink load: For packets within each QP, PRO assigns $PathID$ within the equivalent paths set in a round-robin order. As packets belonging to the same QP always choose the same source ToR switch, thus round robining packets in the same QP means distributing packets across all egress ports of the source ToR switch. As a consequence, packets can be uniformly distributed among the multiple downlinks between the aggregation switches and the destination ToRs.

Inter-QP round robin to balance uplink load: However, only intra-QP-level round robin can not guarantee the balanced queues at all switches. Since LLM traffic exhibits a synchronized pattern, multiple RNICs can round robin sending packets simultaneously. What's more, collective communications like AlltoAll incorporate many/one-to-many patterns, thus one RNIC may eventually send traffic to multiple RNICs. On the sender side, if multiple QPs (within the same RNIC) concurrently choose the ToR uplink egress ports in the equivalent paths set in the same order, the synchronization contention among QPs can cause micro load imbalance at uplinks.

As shown in Figure 3, assume a AlltoAll communication pattern, there are 2 QPs in 2 RNICs sending to the same destination ToR (i.e., with the same equivalent path set). These QPs have data to send simultaneously. There are four equivalent uplink egress ports (P1, P2 P3, and P4), i.e., four equivalent paths. As illustrated in the left most part of Figure 3, with a simple QP-unaware NIC-level round robin, each QP can not fully utilize the downlink bandwidth of multiple paths. However, with only intra-QP round robin, multiple RNICs may

[2]PRO can support 3-stage topology and extend to more stages. In this paper, we adopt the two-stage network by default, similar to [22].

result in simultaneously sending data to the same ToR uplink egress ports P1 (at the first scheduling round), thereby causing transient queue spikes.

To mitigate such micro imbalance at uplinks while fully utilizing the uplink bandwidth, PRO manipulates the round-robin order within each equivalent path set of multiple QPs. For each QP in the same RNIC, the round-robin order within the conceptual equivalent path set is different so that uplinks be staggered. As illustrated in the right most part of Figure 3, the round-robin order of QP1 and QP2 is staggered under Algorithm 1, which alleviates the uplink load imbalance. The egress queues in P1, P2 are more balanced at the first round.

Algorithm 1 shows the orchestrated round-robin mechanism in PRO. Assume there are M equivalent network paths[3]. PRO maintains a global counter C per-destination equivalent path set to coordinate path selection among multiple QPs. For multiple QPs, the native QP scheduler in RNIC schedules QPs one by one, and fetches one packet in a QP at a time. Then PRO calculates paths for each packet and update corresponding QPC (lines 1-4). Specifically, within each QP, the round-robin order in the equivalent path set can be characterized by *span*, which is the order increase step within the path set. For instance, if *span* is 1, it means $PathID$ will be increased one by one to achieve round-robin. By assigning different values of *span* to different QPs, PRO can stagger the round-robin order among QPs. Considering that the number of network paths M is an integer power of 2 in the LLM cluster, PRO makes *span* of each QP an odd number (lines 6-9), then any QP can fully use M different paths to send M packets[4]. For newly-arrived QPs appending at the end of the QP list, PRO staggers the paths of the new QP with existing QPs, preventing the QP of the same RNIC from choosing the same path (lines 10-12).

Adapting to batch scheduling at RNICs. Algorithm 1 assumes that the QP scheduler in RNICs schedule QPs in a per-packet manner, i.e., each QP sends only one packet during its round [14]. In practice, QP schedulers in RNICs may opt for batch scheduling, i.e., a QP sends multiple packets (e.g., 8 packets) at a time to avoid the need to maintain the WQE cache [28]. With the batch scheduling, PRO calculates paths for packets in a batch in sequence. To guarantee that M paths are evenly scattered among concurrent batches of multiple QPs, assume the batch size is n, then n packets belonging to the same batch should be sprayed among M/n paths. Specifically, the update of counter C should considers the batch size, i.e., $C \leftarrow QP_i.path + n$ (line 12) and M should be changed to M/n (line 9 and line 11).

Theorem 1. *(Bounded queue length property) Orchestrated round robin in PRO can guarantee bounded queue lengths at the network core even with full load.*

Proof. The detailed proof is listed in § IV. □

[3]the number of equivalent network paths are identical for all destinations under a symmetric topology

[4]This can be proved according to Extended Euclidean algorithm [29].

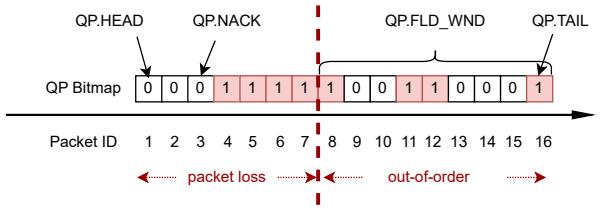


Figure 4: Dynamic OOO window to distinguish out-of-order and lost packets.

Algorithm 2 PRO OOO window adjustment and retransmission algorithm. $QP.wnd$ is initialized as BDP/MTU. $QP.NACK$ indicates the packet sequence number that has already sent NACK.

```
1:  function RECEIVE(QP)
2:      while NIC.eventq.pop() do
3:          if event is PKT_ACCEPT then
4:              pass;
5:          if event is PKT_REPEAT then
6:              QP.wnd = QP.wnd + 1;
7:          if event is QP_TIMEOUT then
8:              QP.wnd = QP.wnd/2;
9:          CheckAndSendNACK(QP);
10: function CHECKANDSENDNACK(QP)
11:     Skip QP.NACK while seq QP.NACK has received;
12:     if QP.tail − QP.wnd > QP.NACK then
13:         QP.NACK = QP.NACK + 1;
14:         Send Nack with sequence ← QP.NACK ;
15:     else
16:         return;
```

B. Fast Loss Retransmission

With the aid of orchestrated per-packet round robin, packets can be spread evenly across switch ports. Thus, packet loss due to congestion rarely happens. However, flows inevitably experience packet loss due to network failures (e.g., chip faults, routing black holes, etc.). Although PFC can be used to avoid packet loss due to congestion in RDMA-enabled datacenters, packet loss caused by failures remains unsolved.

What's more, per-packet round robin imposes more challenges in distinguishing lost packets. Traditionally packet loss in the context of RDMA can be easily identified by out-of-order (OOO) arrivals. However, per-packet round robin itself may introduce OOO arrivals and a single link failure can affect all flows going out of the switch. Thus, a flow may suffer from a mix of packet loss and truly OOO packets. On the whole, two situations coexist with naturally incompatible demands: lost packets should be retransmitted timely after failures while truly OOO packets should wait for subsequent packets to arrive without triggering unnecessary retransmissions.

Our key insight is that *the degree of out-of-order is indeed limited with orchestrated round robin while the lost packets due to failures may never arrive*. Thanks to the ability of PRO to achieve a well-balanced load and extremely low variances among queue lengths, the accumulated queue lengths are similar irrespective of the different equivalent paths. Thus, the key problem is to find a proper threshold of OOO, i.e., the maximum window containing OOO packets.

To this end, PRO adjusts an OOO window to reflect the

bound of the OOO degree. Based on the OOO window, PRO can determine whether the current out-of-order arriving packet is indeed lost or not. As shown in Figure 4, for each QP, PRO maintains a bitmap at the receiver side to record whether packets arrive or not. $QP.head$ indicates the unarrived packets with the smallest sequence number. $QP.tail$ indicates arrived packets with the largest sequence number. $QP.wnd$ is the maximum OOO window starting from $QP.tail$. In Figure 4, $QP.wnd$ is 8, thus packets 9-10 and 13-15 are considered out-of-order, and older packets 1-3 are considered lost since they are outside the window. Each out-of-order packet outside the OOO window should trigger NACK for retransmission.

PRO adjusts the size of the OOO window dynamically utilizing an Additive Increase Multiplicative Decrease (AIMD) algorithm to maintain the window around a target size, as shown in Algorithm 2. If the OOO window is too small, true OOO packets may be mistakenly regarded as lost, and retransmitting these true OOO packets induces repeated arriving packets. On the other hand, if the OOO window is too large, lost packets due to failure may be regarded as OOO packets and finally experience a timeout. Since timeout greatly affects the delay performance and can be regarded as a severe "punishment", PRO decreases the size of the OOO window multiplicity to aggressively react to timeout (lines 9-10). The increase of the OOO window is gradual because there may be multiple repeated arriving NACKs (lines 6-7).

C. Out-of-Order Handling

Out-of-order packet placement. PRO as a per-packet load balancing should support OOO deivery at the receiver. We support directly placing OOO packets in application memory. To achieve this, each packet's header is extended with a virtual address, remote key, and DMA length segments. Then RNIC first verifies the remote key for the access authority. Then, it translates the virtual address to the physical address through the memory translation table (MTT) and DMAs OOO packets directly to the final physical address in the application memory. This reduces RNIC memory requirements from 1KB per OOO packet to only a couple of bits compared to solutions to rearrange OOO packets at RNIC on-chip memory [17]. To our knowledge, commercial NVIDIA CX-7 also supports a direct DMA of OOO packets [19]. But in contrast to CX-7 which only supports RDMA *write only* opcode, PRO supports all RDMA *write* and *read* operations for OOO packet placement.
In-order message delivery. Although the data packets in the message can be placed in the host memory in a loose order, to ensure that the completion order of the message is consistent with the application, the receiving side still needs to ensure that the order of the message completion event (CQE) is delivered in the same order as the sending side. In PRO, both sides use multiple bits to record the data packet status. For instance, bit 0 records whether the message has arrived, bit 1 records whether the message is the last message, and bit 2 records whether the message needs to generate a CQE. When each data packet arrives at the receiving side or ACK arrives at the sending side, the RNIC checks whether the relevant packet of the QP's first message has all arrived. If it has arrived, the message will be popped out of the QP, otherwise, it will continue to wait for the next packet. When the message is popped out of the QP, the RNIC checks the CQE flag bit of the message. If the flag is set, the corresponding CQE is placed in the completion queue corresponding to the QP. In this way, the upper-layer application can ensure that the order of message initiation and completion is consistent.

IV. Theoretical Analysis

In this section, we prove that PRO achieves bounded queue length at the network core (i.e., the ToR switch layer and aggregation switch layer). We first presents the network model (§ IV-A). Based on the model, we derive a critical math property that, the queue lengths will reach a steady state. Next, we provide a method to calculate the length distribution and delay estimation in the steady state (§ IV-B).

A. Network Model

Topology. We assume a a two-stage network (ToR-Aggr) topology[5]. There are B_{in} RNICs and one ToR switch in each rack. Each ToR switch has B_{in} ingress ports and B_e egress ports ($B_{in} \leq B_e$). Each ingress port connects to a RNIC within the rack and each egress port connects to an Aggr switch in the upper level. At the aggregation layer, each Aggr switch has B_e ingress ports and B_e egress ports to ToR switches. Each egress port of a switch has a queue, where incoming packets wait to be processed.
Traffic pattern. For the switches, assume each port requires one unit time to process a packet. Packets are synchronous, *i.e.*, all the ports send their packets simultaneously. To avoid overload, no more than B_{in} RNICs across the racks are allowed to send packets simultaneously to a rack. Due to the synchronized feature of LLM traffic patterns, we assume all sender RNICs have the same number of concurrent flows. Let P denote the number of QPs per RNIC. To facilitate the following proof, we assume P is odd. A similar analysis can be used in the case where P is even.

The most congested situation at a ToR switch happens when all the connected RNICs send packets to other racks in parallel, constituting the upper bound of the queue length.
Characteristics and formalization. Suppose all egress queues are empty at time 0, and packets arrive in some given sequences starting from time 1. If the queue is non-empty at time t, one packet will be popped from the queue, and occupy the port until time $t+1$.

To begin with, we consider queues at any ToR switch. At each unit time, a sum of B_{in} packets arrive at all ingress ports, each of which has a probability $\frac{1}{B_e}$ to be destined for a specific egress port. Let $n_{t,i}$ to be the number of packets in the i-th egress queue arrived at time t, we have

$$\sum_{i=1}^{B_e} n_{t,i} = B_{in} \leq B_e, \forall t > 0 \tag{1}$$

[5]We adopt the two-stage network for modeling and analysis, but the same method can be extended to a three-stage network.

As PRO can achieve orchestrated round robin, each RNIC selects the next egress port to send packets, thus

$$n_{t+1,i} = n_{t,1+(i-2) \bmod B_e}, \forall t, i > 0 \tag{2}$$

Eq. 2 hints at a periodical incoming sequence at i-th egress:

$$\mathcal{S}_i^{ToR} = (n_{1,i}, \cdots, n_{1,1}, n_{1,B_e}, \cdots, n_{1,i+1}) \tag{3}$$

For any Aggr switch, let $n'_{t,i}$ be the number of packets to the j-th egress at time t, we have

$$\sum_{i=1}^{B_e} n'_{t,i} \leq B_e, \forall t > 0 \tag{4}$$

Because RNICs send data from each QP alternately in a fixed order, we define $N_{t,i}$ as the incoming sequence at time $[(t-1)P+1, tP]$ in the j-th egress, *i.e.*,

$$N_{t,i} = (n_{(t-1)P+1,i}, \cdots, n_{(t-1)P+p,i}, \cdots, n_{tP,i}), \forall t, i > 0$$

It is trivial that

$$N_{t+1,i} = N_{t,1+(i-2) \bmod B_e}, \forall t, i > 0 \tag{5}$$

Based on Eq. 5, we know that the i-th egress has a periodic incoming sequence

$$\mathcal{S}_i^{Leaf} = (N_{1,i}, \cdots, N_{1,1}, N_{1,B_e}, \cdots, N_{1,i+1}) \tag{6}$$

Summary. Both Eq. 3 and 6 can be summarized in the following form:

$$\mathcal{S}_i = (\underbrace{n_i, \cdots, n_1, n_\tau, \cdots, n_{i+1}}_{\tau}) \tag{7}$$

where τ is the repeating period, i is an offset in $[1, \tau]$, and n_i represents the number of arriving packets each unit time. Moreover, Eq. 1 and 4 guarantee that

$$\sum_{n \in \mathcal{S}_i} n \leq \tau \tag{8}$$

B. Proof of Theorem 1

Definition 1. *For the arriving sequence $\mathcal{S}_i$ in Eq. 7, $\mathcal{S}_i$ is defined as good if the initial queue length is zero, after τ unit time the queue length will become 0 again.*

From definition 1, it can be concluded that if a queue is initially empty and the packets arrive as $\mathcal{S}_i$, the queue will become empty again after τ unit time.

Lemma 2. *There exists an offset i such that $\mathcal{S}_i$ is good.*

Proof. To prove the existence of a good arrival sequence $\mathcal{S}_i$, we just need to find an offset i which is good. According to the definition of a good arrival sequence, if a queue is initially empty, the queue will become empty again after a good arrival sequence. We first prove that for all $t \in (0, \tau]$, there exists a time t that the queue is empty (**Step 1**). Then, we prove that for different scenarios, we can always find a good arrival sequence (**Step 2**).

Step 1: Consider the sequence $\mathcal{S}_\tau = (n_\tau, \cdots, n_1)$ and an empty queue at time 0. Suppose the queue is never empty in $(0, \tau]$, then one packet is popped from the queue every unit time. Consequently, τ packets are popped after τ unit time and there are still packets in the queue. Since the queue is initially empty, we can derive that $\sum_{n \in \mathcal{S}_i} n > \tau$, which is conflicting with Eq. 8. Thus, it can be concluded that there is a time $t \in (0, \tau]$ when the queue is empty.

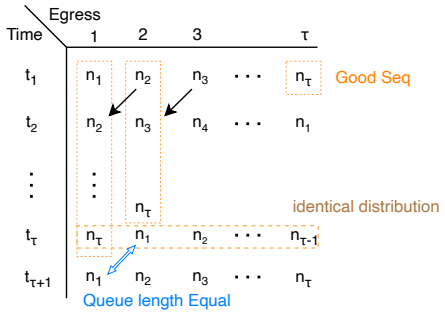


Figure 5: Identical queue length distribution

Step 2: Let $t \in (0, \tau]$ be the last time when the queue is empty, *i.e.*, the queue is never empty in $(t, \tau]$.

If $t = \tau$, the original sequence is a good arrival sequence. The theorem can be proved.

Otherwise (i.e., $t < \tau$), during $(t, \tau]$, the queue is never empty, and there are $\tau - t$ packets popped from the queue. Let $s = \sum_{i=1}^{\tau-t} n_i$, the remaining packets at time t are $s - (\tau - t)$. Under Eq. 8, at most $\tau - s$ packets arrive in $(0, t]$. Since the queue is empty at time t, all these $\tau - s$ packets are popped in this period. In addition, the egress is capable of popping t packets in this period, which also means that there are at least $t - \tau + s$ units of time when the port is idle. Now, consider $\mathcal{S}_{\tau-t} = (n_{\tau-t}, \cdots, n_{1,1}, n_{1,\tau}, \cdots, n_{1,\tau-t+1})$, which is $\mathcal{S}_\tau$ after a rotation of t unit time. The starting period $(0, \tau - t]$ is identical to the time $(t, \tau]$, and consequently, the queue length at time $\tau - t$ is at most $s - \tau + t$. For the trailing period $(\tau - t, \tau]$, all the packets in the queue can leverage the $t - \tau + s$ units of idle time, so that the queue is empty at time τ. In conclusion, $i = \tau - t$ satisfies the theorem. □

Lemma 3. *If a sequence $(x_1, \cdots, x_\tau)$ is good, for all $i \in [1, \tau]$, the sub-sequence $(x_i, \cdots, x_\tau)$ is also good.*

Proof. The main idea is that if the $(x_1, \cdots, x_\tau)$ is good, then when the queue length is initially zero, with the packets arriving as the $(x_1, \cdots, x_\tau)$ sequence, the queue length will become zero again after τ unit time. As for any $i \in [1, \tau]$, the queue length must be no less than zero, and after $(x_i, \cdots, x_\tau)$ arrival sequence, the queue length will become zero. It can be inferred that when the queue length is zero at i ($i \in [1, \tau]$), after $(x_i, \cdots, x_\tau)$ arrival sequence, the queue length will become zero. □

Lemma 4. *Given packet arrival sequence $\mathcal{S}_i$ ($i \in [1, \tau]$), let $q_{t,i}$ represent the queue length at time t. For all time $t \geq \tau$ and packet arrival sequence $\mathcal{S}_i$ ($i \in [1, \tau]$), the distribution of the queue length $q_{t,i}$ are identical.*

Proof. As RNICs pick the initial state uniformly and independently, it is trivial that, at any given unit time, the distribution

Resource Usage			
	Orchestrated Round Robin	Loss Recovery	Total
ALMs	0.3K (0.03%)	20.7K (2.15%)	21K (2.18%)
REGs	0.4K (0.03%)	34.3K (3.19%)	34.7K (3.22%)
RAMs	15KB (0.04%)	53KB (0.16%)	68KB (0.2%)
LOC	317	7794	8111

Table I: Resource usage of PRO prototype at RNIC.

of queue length at all egress ports are identical. This means that the queue length distribution across different columns ($q_{t,j}$ and $q_{t,j+1}$ for any t) of Figure 5 are identical.

Next, we prove that the queue length distributions across time (i.e., different rows in Figure 5) are also identical. No matter which state each RNIC chooses, according to Lemma 2, there always exists a good packet arrival sequence $\mathcal{S}_i$ ($i \in [i, \tau]$). As shown in Figure 5, we consider a packet arrival sequence of Egress 1, $(n_1, n_2, ...n_\tau)$ is a good sequence. According to the definition of a good sequence in Definition 1, the queue length of Egress 1 will become zero after τ unit time. At $\tau+1$, there are n_1 packets arrive at the empty queue. The arrival sequence of Egress 2 is $(n_2, ...n_\tau, n_1)$. According to Lemma 3, $(n_2, ...n_\tau)$ is also a good sequence. Thus, at τ, there are also n_1 packets arriving at the empty queue. Therefore, we have $q_{\tau+1,1} = q_{\tau,2}$. Similarly, it can be inferred that when $t \geq \tau$, there is $q_{t+1,j} = q_{t,1+(j+1) \mod \tau}$ for any $j \in [1, \tau]$. Thus, the queue length distributions across time (i.e., different rows in Figure 5) are also identical when $t \geq \tau$.

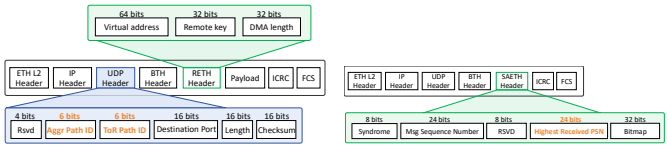
By summarizing the analysis, we can conclude the lemma. □

According to Lemma 4, the steady-state queue distribution of the switch egress is equal to the queue distribution at τ time. The total number of arriving packets up to τ is $HB_{in} \leq HB_e$, where $H = 1$ for ToR switches and $H = P$ for Aggr. Therefore, we conclude that the number of queuing packets at ToR and Aggr switches is also bounded.

Summary. Lemma 4 indicates that queues at both ToR and Aggr switches can converge to a steady state in one time cycle. This also means that the queue lengths at the network core are generally bounded, which we conclude to Theorem 1.

V. Implementation

Packet header format. Figure 6 demonstrates the packet header extension format in PRO, which mainly contains three parts: (i) OOO packet placement in the RETH header. To support direct OOO packet placement, we add 64 bits virtual address, 32 bits remote key, and 32 bits DMA length segments in the RETH header. (ii) Path tagging in UDP header: We utilize the lower 12 bits of the UDP source port field to carry $PathID$, which can support an AI datacenter cluster with 524K GPUs. Since RoCEv2 protocol only uses the destination port field to identify RoCEv2 packets and does not rely on the source port field, the modification of the source port field has no additional impact on RoCEv2 protocol. The lower 6 bits of $PathID$ field indicate ToR egress ports and the higher 6 bits indicate Aggr egress ports. (iii) Maximum PSN number. Loss detection in PRO needs the maximum acknowledged PSN to trigger packet loss judgment.



(a) Data header format (b) ACK header format

Figure 6: Packet header format

RNIC prototype implementation. Our RNIC prototype is built upon an Intel Agilex 7 FPGA [13] that connects to host main memory via PCIe Gen5×16 lanes and the network through a optical module, featuring a dual-port 2 × 200 Gbps programmable RNIC supporting RoCEv2 protocol. The logic of PRO is implemented in ∼8100 lines of Verilog as specific RDMA events, executed independently in hardware threads awakened by a dedicated scheduler. Table I reports the resource usage of PRO prototype in 10K QPs. The current implementation of orchestrated round robin (RR) engine and loss recovery (LR) engine utilizes ∼2.18% ALMs, ∼3.22% REGs, and ∼0.2% RAMs. A 2MB shared cache can be used for QPC/WQE. The low resource consumption is because PRO does not need to maintain path states, and the bounded queues saves memory resources for loss recovery.

Switch functions. The switch runs the control plane based on the SONiC switch OS [26] and configuring the data plane via the Tomahawk4 SDK. We leverage the integrated ACL functionality to implement forwarding path control in ∼2000 LoC. The core procedure is updating the ACL list.

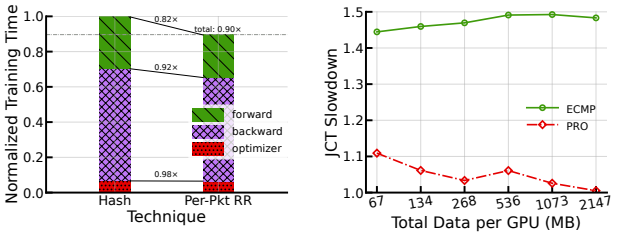
Handling link failures. In normal conditions, the switch maintains the ECMP group (equivalent ports for each destination) in a local routing table. The control plane inserts ACL entries corresponding to the ECMP group to enforce path selection based on $PathID$. In case of link failures, the control plane detects the links that are reachable to a certain destination through the standard BGP process and then updates the ECMP group in the local routing table. Next, the control plane issues corresponding ACL entries according to the new ECMP group. Specifically, for the port that no longer exists in the new ECMP group, the control plane issues a new ACL entry with AR [5] as the forward action.

VI. Evaluation

A. Evaluation Setup

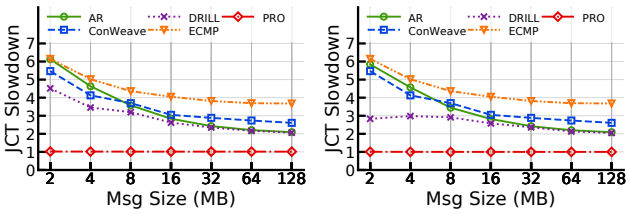
Testbed: Our testbed consists of 4 servers connected via 16 switches. Each server is equipped with 4 AMD EPYC CPU, 2300GB RAM, 8 H800 GPU model, and 4 2×200GbE RNIC with PRO prototype, and runs CentOS 7 with Linux kernel v5.4.203. All switches are 128×200Gbps Broadcom Tomhawk4, where 8 Top-of-Rack (ToR) switches are connected with 8 Aggregation (Aggr) switches. Each link bandwidth is 200Gbps. We use the default shared buffer settings in the switch.

Simulation: We develope our simulator based on an open-sourced NS3 simulator [4]. We use Fat-Tree topology [1] with 8 Aggr switches, 16 ToR switches, and 128 servers. All switch-to-switch and switch-to-server links are 200Gbps. The link delay is 1 µs. The switch buffer size is 100MB and the dynamic



(a) MoE AI model training (b) NCCLtest

Figure 7: JCT slowdown performance [Testbed].



(a) Ring (b) Tree

Figure 8: JCT slowdown performance [NS-3].

threshold with $\alpha = 1$ [7]. PFC is enabled in our experiments. The whole network is a single RDMA domain.

Traffic: We generate traces according to ring-based and tree-based collective communication patterns of LLM traffic. (i) Ring: each server randomly selects its neighborhood and forms a ring communication mode, which is the basic unit of collective communications including reduce-scatter,all-gather, and ring-allreduce [20]. (ii) Tree: each server randomly selects its pattern/child server and then all servers represent a double binary tree [21], which is also the basic unit of collective communications including reduce, broadcast, and tree-allreduce. The message size is 16MB by default.

Baselines: We compare PRO with four LB schemes, *i.e.*, ECMP, Adaptive Routing (AR) [5], DRILL [11], and ConWeave [25]. We refer to Broadcom Tomhawk4 ASIC to implement AR [5]. Specifically, AR forwards each packet at switches according to the quality of egress port, *i.e.*, link load and queue length updated every 1µs. In terms of loss retransmission, we compare fast loss detection mechanism in PRO with MPRDMA [16]. MPRDMA selects packet sending path to control the amount of out-of-order arrival packets, striving to making it smaller than the length of bitmap.

Parameter settings: In simulations, we configure DCQCN's parameter based on the recommendation in [25]. Specifically, we set $K_{min} = 100KB$, $K_{max} = 400KB$ and $P_{max} = 20\%$ as ECN marking threshold. We set 80 μs RTO for the corner case of tail packet loss. Other parameters related to ECMP, AR, DRILL, and ConWeave are as recommended in their papers and vendor devices [5], [11], [25].

Metric: We use job completion time (JCT) slowdown as the primary metric. JCT is the ratio of the actual completion time of the last flow of ring/tree traffic to the ideal completion time. Besides, we measure the queue length imbalance, link load imbalance, and out-of-order degree for analysis. In detail, queue imbalance is the difference between maximum queue length and average queue length among all egress ports, which is sampled per 0.1 µs. Load imbalance is calculated by the highest link load divided by the average link load, recorded per 10 µs. The out-of-order degree is measured by the current packet's PSN minus the maximal in-order packet's PSN.

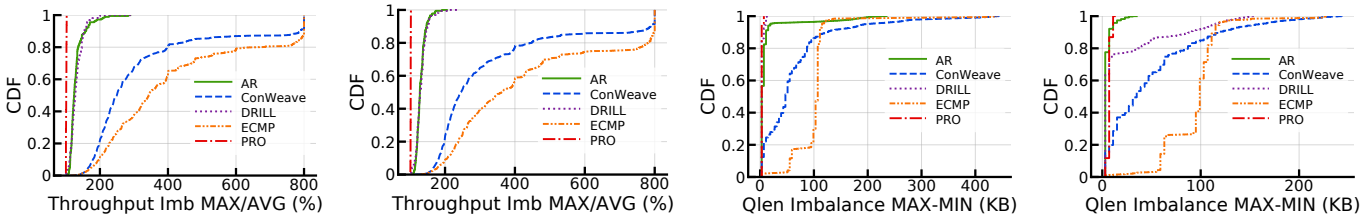
B. Performance Comparisons

PRO improves LLM applications. We first compare our PRO with widely-used ECMP in the testbed. Specifically, 4 servers run MoE model [23] and NCCL test application [18] in which each GPU generates network traffic from 67MB to 2147MB. The results of the JCT slowdown are in Figure 7(a). Compared to ECMP, PRO reduces the iteration time of each round by 10.4% in model training, i.e., the time of forward computation and backpropagation are reduced by 17.9% and 7.6%, respectively. We dive into the communication process of MoE model training, which contains all-to-all and all-reduce communication. We notice that most traffic follows an all-to-all communication pattern. Then, we use the NCCL test application to further demonstrate the performance under all-to-all communication. As Figure 7(b) shows, PRO improves 1.3∼1.48× JCT slowdown performance compared to ECMP. Moreover, PRO can achieve near-optimal JCT slowdown for large messages since PRO achieves balanced queues in the network core.

PRO achieves near-optimal JCT slowdown. Next, we compare PRO with existing state-of-the-art alternatives, including ECMP, AR, DRILL, and ConWeave, in terms of JCT slowdown in simulations. As Figure 8 shows, compared to these alternatives, PRO reduces the JCT slowdown by 50.9%∼83.6%. Both ECMP and Conweave are per-flow granularity, where long flows stick to their path once sent out (or only rerouting after severe congestion), thus resulting in collisions and significant imbalance in link utilization. Especially, links between ToR switches and Aggr switches are under-utilized. Since the simulation has a larger topology than the testbed, ECMP is prone to more severe collisions. Although AR and DRILL are per-packet granularity and improve the balance degree of link utilization, however, under heavy LLM traffic loads, AR and DRILL can induce a microscopic queue imbalance in switches and trigger ECN marking.

As listed in Table II, existing alternatives all experience ECN marks except PRO. We also notice that AR suffers from queue imbalance in the ToR uplinks while DRILL suffers from queue imbalance in the Aggr downlinks. As a result, they cause unnecessary rate decrease and thus link utilization is occasionally lost. Fortunately, our PRO adopts a global-aware per-packet round-robin which guarantees precise balance, *i.e.*, even link utilization across links, and uniform queue length across switch ports. Overall, our PRO maintains 1.0∼1.02 JCT slowdown across all message sizes, which is near-optimal.

PRO achieves highly-efficient load balancing. To understand where improvements in JCT slowdown come from, we take a closer look at the effect of load balancing (including link utilization and queue length) in the switch. We monitor these metrics of all links between ToR switches and Aggr

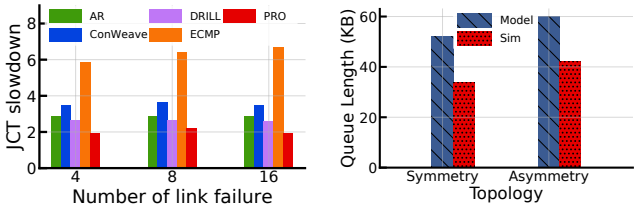


(a) Load imbalance of ToR uplinks (b) Load imbalance of Aggr downlinks (c) Qlen imbalance of ToR uplinks (d) Qlen imbalance of Aggr downlinks

Figure 9: Comparisons with state-of-the-arts [NS-3, traffic pattern: Ring].

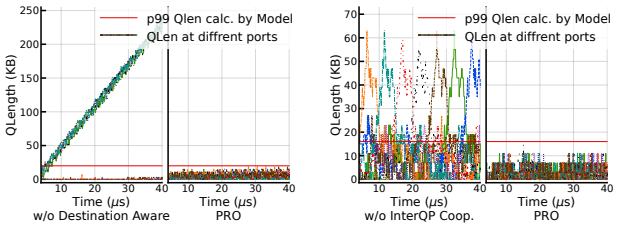
Table II: ECN count [NS-3].

	ToR switches	Aggr switches
ConWeave	2020578	619500
ECMP	1859745	1453340
AR	2760	0
DRILL	0	218
PRO	0	0



(a) Asymmetry topology (b) Asymmetric topology and silent packet drops

Figure 10: Handling network failure [NS-3].



(a) Queue length of Aggr downlink under Tree traffic. (b) Queue length of ToR uplink under 32-concurrency all-to-all traffic.

Figure 11: PRO breakdown [NS3].

switches. Figure 9(a) and Figure 9(b) report the balance degree of link utilization. As we can see, PRO, DRILL, and AR perform better than Conweave and ECMP since packet-level granularity load balancing can respond to link conditions more timely and make fine-grained decisions. Specifically, PRO achieves 6.8× improvements on link utilization balance at the 99th percentile. Figure 9(c) and Figure 9(d) depict the balance degree of queue length. The results show that, for ToR uplink, DRILL achieves a better balance performance than AR due to its delay-free awareness of queue length. But for Aggr downlinks, since traffic forwarded by AR is imbalanced in the uplinks, the load in downlinks is relatively slight and exhibits uniform falsehoods. Compared to all these alternatives, PRO stands out due to its nature of orchestrated round-robin and well-balanced ability at both uplinks and downlinks. Specifically, PRO achieves better queue length balance, improving by 0.6×∼55× at the 99th percentile.

PRO is efficient to handle failures. We evaluate the performance of PRO under the following two failure scenarios. We let 4, 8, and 16 links down in the topology (a total of 1024 links among all switches) and repeat the experiments. Figure 10(a) demonstrates that PRO can handle link failure induced asymmetry topology well, achieving 1.9∼2.2× better JCT slowdown performance. ECMP achieves the worst performance. AR, ConWeave, and DRILL can adapt to asymmetric links by reactive sensing thus achieving similar performance when the number of failed links increases. Although PRO temporarily resorts to AR when needs rerouting, the comparison confirms that PRO can still bring performance benefits when link failures are severe.

PRO has minimal packet reordering. Since reordering may (i) increase MTT overhead, (ii) affect the packet loss detection and (iii) consume bitmap resources, an ideal LB mechanism should introduce reordering as little as possible. Figure 12 shows out-of-order degree under ring traffic pattern. As we can see, AR and DRILL suffer from severe reordering, whose 99th-percentile out-of-order packets are 44 and 22 respectively. By contrast, benefiting from well-balanced queue length, PRO only introduces 4 out-of-order packets at the 99th percentile even under heavy load (i.e.,94% link load).

C. Performance Breakdown of PRO

Benefits from destination-aware equivalent path set and inter-QP orchestration. Figure 11(a) shows the queue length from all Aggr switches to the same ToR switch under the tree traffic pattern. RR without destination-aware may introduce a continuous queue length increase. This is because one TX NIC can simultaneously communicate with two RX NICs in this case. And then once two RX NICs are under the same ToR, RR without destination-aware may not fully utilize Aggr downlinks, thereby causing load imbalance and even congestion. Figure 11(b) demonstrates the queue length from all Aggr switches to one ToR switch under a 32-concurrency all-to-all traffic pattern. The result shows that RR without inter-QP orchestration would cause micro imbalance at ToR uplinks since there is traffic contention due to synchronization between QPs from the ToR switches. Such uncoordinated QP issues can be further exacerbated with the increase of network scale and number of concurrent QPs, which finally causes high reordering at hosts and impacts the effect of the fast loss recovery mechanism.

Benefits from cautious loss retransmission. The cautious loss retransmission in PRO can efficiently distinguish out-of-

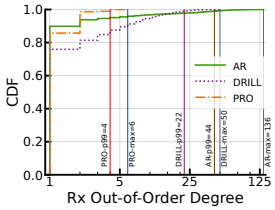


Figure 12: Out-of-order degree [NS-3, Ring traffic].

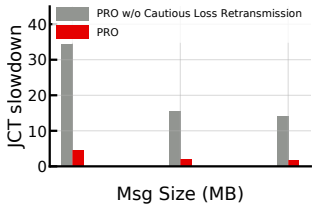


Figure 13: Cautious loss retransmission [NS-3, Ring traffic]

order arrivals and packet losses, thus enabling fast recovery for failure loss. Figure 13 reports the benefits of cautious loss retransmission. Specifically, with the assistance of cautious loss retransmission, PRO further reduces the JCT slowdown by 86.9%∼88.8%.

VII. Conclusion

In this paper, we emphasize the importance of deterministic load balancing for LLM workloads in AI datacenter networks. We present PRO, a novel load balancing scheme that enforces a deterministic round-robin order on RNICs while accounting for both intra-QP and inter-QP traffic. PRO achieves balanced queue occupancy at both the ToR and aggregation switch layers. We implement PRO on FPGA hardware. Extensive experimental results demonstrate that PRO reduces JCT slowdown by 50.9%–83.6% compared with state-of-the-art mechanisms. By revisiting load balancing design tailored for LLM workloads, we believe PRO represents a promising direction for future AI datacenter networks.

Acknowledgements

This work is supported in part by the National Key Research and Development Program of China (No. 2024YFB2907000), by the National Natural Science Foundation of China (NSFC) under Grant Nos. 62302055, 62372061, 62502197, and U25B2034, by the Basic Research Program of Jiangsu under Grant No. BK20251206, and by the Shandong Provincial Natural Science Foundation under Grant No. ZR2024LZH006. Corresponding author: Qingkai Meng.

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the 2008 ACM Conference on SIGCOMM*, page 63–74, 2008.

[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. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, page 503–514, 2014.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, page 63–74, 2010.

[4] Conweave authors. Ns-3 simulator for rdma network load balancing. https://github.com/conweave-project/conweave-ns3, 2023.

[5] Broadcom. Tomahawk4. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series, 2024.

[6] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *Proceedings of the ACM CoNEXT*, page 49–60, 2013.

[7] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking*, 6(2):130–140, 1998.

[8] Ultra Ethernet Consortium. Ultra ethernet specification v1.0.1. https://ultraethernet.org/wp-content/uploads/sites/20/2025/10/UE-Specification-1.0.1.pdf, 2025.

[9] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. In *Proceedings of IEEE INFOCOM*, pages 2130–2138. IEEE, 2013.

[10] Facebook. Llama: Open and efficient foundation language models. https://research.facebook.com/file/1574548786327032/LLaMA--Open-and-Efficient-Foundation-Language-Models.pdf, 2023.

[11] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM SIGCOMM*, page 225–238, 2017.

[12] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM SIGCOMM*, page 29–42, 2017.

[13] Intel. Intel agilex 7 fpga. https://www.intel.com/content/www/us/en/products/sku/210676/intel-agilex-7-fpga-iseries-027-r29a/specifications.html, 2024.

[14] Sugi Lee, Mingyu Choi, Ikjun Yeom, and Younghoon Kim. Perf: preemption-enabled rdma framework. In *Proceedings of the 2024 USENIX Conference on USENIX ATC*, 2024.

[15] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[16] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. Multi-Path transport for RDMA in datacenters. In *15th USENIX Symposium on NSDI*, pages 357–371, 2018.

[17] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM SIGCOMM*, page 313–326, 2018.

[18] Nvidia. Nccl tests. https://github.com/NVIDIA/nccl-tests, 2021.

[19] Nvidia. Nvidia connectx-7 400g ethernet. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectx-7-datasheet-Final.pdf, 2021.

[20] Nvidia. nccl communication pattern. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/operations.html, 2024.

[21] Nvidia. nccl source code. https://github.com/NVIDIA/nccl/tree/v2.17.1-1, 2024.

[22] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, 2024.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[24] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

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

[26] Sonic. https://github.com/sonic-net/sonic-buildimage/tree/201811, 2024.

[27] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on NSDI*, pages 407–420, 2017.

[28] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. {SRNIC}: A scalable architecture for {RDMA}{NICs}. In *20th USENIX Symposium on NSDI*, pages 1–14, 2023.

[29] WikiPedia. Extended euclidean algorithm. https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm, 2024.