STrack: A Reliable Multipath Transport for AI/ML Clusters

Yanfang Le[1], Rong Pan[1], Peter Newman[1], Jeremias Blendin[1],
Abdul Kabbani[2], Vipin Jain[1], Raghava Sivaramu[1], Francis Matus[1]

[1]AMD [2]Microsoft

Abstract

Emerging artificial intelligence (AI) and machine learning (ML) workloads present new challenges of managing the collective communication used in distributed training across hundreds or even thousands of GPUs. This paper presents STrack, a novel hardware-offloaded reliable transport protocol aimed at improving the performance of AI /ML workloads by rethinking key aspects of the transport layer. STrack optimizes congestion control and load balancing in tandem: it incorporates an adaptive load balancing algorithm leveraging ECN, while adopts RTT as multi-bit congestion indicators for precise congestion window adjustment. Additionally, STrack facilitates out-of-order delivery, selective retransmission, and swift loss recovery in hardware for multipath environment. The extensive simulation comparing STrack and RoCEv2 demonstrates that STrack outperforms RoCEv2 by up to 6X with synthetic workloads and by 27.4% with collective workloads, even with the optimized RoCEv2 system setup.

1 Introduction

The widely adopted transport technology in the AI/ML cluster today is Infiniband. However, to achieve even larger scale with lower cost, leading CSPs start to adopt Ethernet based technology, such as RDMA over Converged Ethernet, RoCEv2 [25]. In traditional cloud environment, RoCEv2 is proven to provide ultra-low latency and high throughput with little CPU overhead. However, the standard RoCEv2 has many limitations to be adopted widely for AI/ML clusters.

First, AI/ML training networks demand much higher utilization of their network links, around 70%-80% while cloud networks usually only carry a load of 30%-50%. Due to ECMP hash collisions, RoCEv2, as a single-path transport, fails to evenly distribute traffic load across highly parallel paths in the AI/ML network, and can't provide robustness in the face of link failures. As a result, hot spots are congested while other links run empty. The overall performance suffers. Hence, RoCEv2 can't satisfy the high link utilization requirement of AI/ML workloads.

Secondly, as AI/ML cluster networks grow to connect tens of thousands or hundreds of thousands nodes, transport's error resiliency capability is a must in order to achieve high efficiency. Multipathing would make error recovery even more challenging as the transport needs to determine whether a missing packet is a real packet loss or simply a delayed packet due to multipathing. Although RoCEv2's dependency on lossless Ethernet can help prevent congestion packet drops, bit errors or packet delays happen often in a vast AI/ML network. RDMA's go-back-N error recovery mechanism would result in significant performance loss.

Thirdly, highly paralleled paths in AI/ML clusters demand a congestion control algorithm that works jointly with a load balancing scheme. It is a dedicated act of quickly switching paths when there are under-utilized links versus slowing down in face of heavy congestion across paths. This calls for a new approach to congestion control with the consideration about the effect of adaptive load balancing. DCQCN, the de facto congestion control algorithm for RoCEv2, fails to satisfy such a stringent requirement. In addition to being a single path scheme, DCQCN is a rate-based algorithm that interprets lacking of congestion notifications as good network conditions, which often exacerbates congestion level. This would lead to extremely unbalanced network links, not desirable for AI/ML workloads.

Last but not the least, to achieve the low latency requirement of the AI/ML workloads, it would be ideal to continue hardware offloading the transport layer. To support multipathing, we need to track the congestion states in hardware on each path in order to enable congestion-aware load distribution. However, these states grow linearly with the number of sending paths. This could cause a considerable memory overhead even when a modest number of paths are used. In addition, multipathing can cause the out-of-order delivery at NICs, even if we directly DMA data packets to hosts, the memory footprint to track the packet arrival and control information are still needed at the NIC's cache. To minimize the states required, the congestion control and load balancing need to work together and minimize the out-of-order delivery. DCQCN's inability to effectively balance traffic load would require large re-ordering states at receiver NICs.

This paper presents STrack, the first reliable multipath transport that addresses all aforementioned challenges. STrack

employs a novel mechanism that adaptively spray packets to multiple paths without keeping complicated per path state. In addition, we design a window-based congestion control that yields to path selection choice first before cutting down the window in face of pending congestion. We also assume lossy Ethernet as the link layer technology, and design a reliable error recovery mechanism that is based on out-of-order packet counts at receiver NICs to ensure fast packet recovery with minimal spurious retransmission.

STrack uses egress-marked ECN as a congestion signal for the adaptive packet spray algorithm instead of oblivious packet spray that distributes packets across multiple paths evenly. Note that the paths here are not the physical paths, but different entropy values, e.g. different UDP source ports, that a sender NIC's uses to spray traffic. With different entropies in the packet header, ECMP functions at switches would hash them onto different paths. However, oblivious packet spray still experiences hash collisions, and the situation gets worse over time as congested paths accumulate packets as messages continue. In addition, link failures cause full bisectional network becomes asymmetric, which makes oblivious packet spray unbalanced. STrack keeps a simple bitmap for the entropies that have experienced ECN marks as the congestion state. When an ACK comes back without an ECN mark, that entropy is used to clock out the next new packet. If an ACK comes back with an ECN marked, it is marked in the corresponding position in the bitmap. Next non-marked entropy in a round robin manner is used to clock out new packet. Note that S-track uses only a minimal state, bitmap, to keep congestion information. The bitmap is reset after one or two round trip times.

STrack adopts a sender-only congestion control to handle both fabric congestion and last hop incast. Although a receiver-based scheme can handle a large degree of incast, it is well-known that the AI/ML communication collectives are designed to avoid incast at the application layer. The transport layer incasts do occur but to a much less degree, only up to tens of flows, where sender-based algorithm can easily deal with. In addition, receiver-based congestion control requires sender-based design to combat fabric congestion, and adds more memory footprint to the receiver, which conflicts with our goal of reducing hardware offload complexity.

STrack also addresses RoCEv2's inefficiency by using a window-based algorithm that handles the congestion across the paths and works jointly with the adaptive load balancing scheme. As the congestion shows up, ECN marks packets when they are at the egress, exiting the congested queue. This gives us an early indication of pending congestion. ECN-marked entropies are avoided by the sender. Unlike [33], ECN-marked packets don't necessarily cause STrack to cut down its window. STrack's congestion control uses an average RTT, from the instantaneous RTT samples across different paths to decide whether we should cut the congestion window. Only when the average RTT is above a threshold (i.e., target queuing delay), STrack cuts the congestion window. The rational is that when only a few paths congested, we should take advantage of other paths without slowing down; and would cut rate when many paths are congested such as in an incast case. STrack also uses a novel method to quickly converge a flow's window under heavy congestion by utilizing total acknowledged bytes back at the sender. The details of our congestion control design can be found in Section 3.2.

At the receiver, STrack utilizes an coaleasing ACK approach to reduce the packets processing rate pressure on the NIC. It maintains a bitmap to track out-of-order packet arrival and selectively pick a bitmap segment (due to ACK size limit) to inform the sender to about the most-up-to-date view of the packet arrival. It also devises a novel silence packet loss detection mechanism across three different methods: (1) out-of-order packets counters; (2) probing-based approach; (3) timeout. Once packet loss is detected, the retransmited packets are sent if the congestion window is allowed.
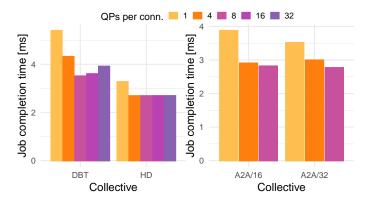
We evaluate STrack with extensive simulations across a broad set of workloads, including permutations, incast traffic and collective workloads with different network scenarios, e.g., different link speed, link down, oversubscription networks, full bi-sectional networks. Our experiments results show that STrack outperforms RoCEv2 up to 6X with synthetic workloads and by 27.4% with collective workloads, even with the optimized RoCEv2 system setup.

2 Background and Motivation

2.1 AI/ML Workloads Challenges over RoCEv2

Collectives play a critical role in distributed AI/ML workloads by enabling efficient communication and synchronization among multiple computational nodes. They help in scaling the training processes, optimizing resource utilization, and reducing communication overhead, which are essential for handling the large-scale data and computational demands of modern AI/ML applications.

Collectives algorithms are known to be designed to avoid heavy incast, e.g., Ring AllReduce, Havingdoubling, DoubleBinary Tree [5, 37], rely either one-to-one or two-to-one traffic patterns. Prior efforts [17, 43] have discovered the importance of the multi-path load balancing for AllReduce collectives. To illustrate the network load balancing impact on the collective communication, we run Allreduce collectives with DoubleBinary Tree (**DBT** in Figure 1) and AlltoAll (**A2A** in Figure 1) with different parallelization, i.e., 8, 16, and 32. The network topology is a 2-tier standard fat-tree topology with 32 NICs to simulate 32GPUs. The link speed is 400Gbps and all the link speeds are the same in the network
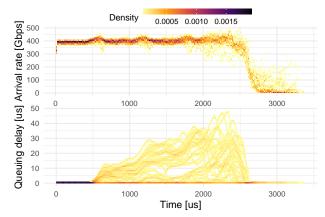
Figure 1: AllReduce collective finishing time.

Figure 2: AlltoAll collective finishing time.

Figure 3: RoCEv2 last-hop link queuing delay and arrival rate of A2A experiment.

topology. The collective size is 128MB. Note that we run one collective at a time, i.e., one collective takes the whole cluster in these experiments.

The collective completion time for Allreduce and AlltoAll operations, as shown in Figure 1, indicates that using more entropies with RoCEv2, such as 4 QPS_PER_CONN (four QPs between source and destination), yields better performance than using single QP (RoCEv2, 1 QPS_PER_CONN). This suggests that ECMP hash collisions underutilize the paralleled path capacities of a network, highlighting the need for a more effective network load balancing scheme for AI/ML workloads.

Furthermore, different source of jitters in the system, e.g., stragglers, NIC scheduler or the GPU thread scheduler, can break the order of the collective algorithms, e.g., AlltoAll, and can cause the moderate incast. To demonstrate this point, Figure 3 monitors the queuing delay at all the last-hop switch queues (total of 32 queues) and traffic arriving rate to each last-hop switch queue as time goes for the AlltoAll experiment with parallel degree of 32 (A2A/32 in Figure 1) for the RoCEv2, 1 QPS_PER_CONN setup. Data shows that arrival rates at the last-hop queues exceed 400 Gbps on multiple ocasions. For example, around 2 ms, the arrival rate peaks at 489.256Gbps at 2357 μs. This high arrival rate, sustained for approximately 500μs, results in switch queuing delays reaching up to 46μs, equivalent to a queue depth of 2.3MB. These significant queuing delays underscore the necessity for a more effective congestion control scheme tailored for AI/ML workloads.

In addition to the issues of single path load balancing and rate-based congestion control, RoCEv2 also requires network to be lossless due to its inefficient loss recovery mechanism, Go-Back-N. Lossless network usually is achieved by enabling priority flow control (PFC). Prior works have been discussed that PFC storms [30], PFC caused network deadlock [22] and head-of-line blocking [13, 35]. In addition, given the vast scale of AI/ML clusters, link errors happen all the time. Adopting go-back-N to recover from link bit errors would be very inefficient. To eliminate the lossless network requirement, a selective retransmission mechanism is desired.

2.2 Hardware Offloaded Transport

RoCEv2 has been proven to achieve high-performance and low-latency by offloading the entire network stack to the NIC hardware. It directly transfers data to and from application buffer (e.g., GPU HBM), which bypasses time consuming processing in the CPU and saves the memory copy between host and GPU memory. To get the same benefits, it is desirable for a new transport for AI/ML workloads is hardware-offloaded as well.

Fortunately, as AI/ML workloads do not have a strict requirement for in-order delivery. Similar as SRD [41], STrack provides reliable out-of-order delivery, without the need for a re-order buffer on the NIC. Nonetheless, the new transport must maintain states to support path congestion information for adaptive multipathing, out-of-order packet arrival information for selective acknowledgements and packet retransmissionson top of a new window-based congestion control algorithm. Accommodating this per-connection state in an efficient manner to allow a huge number of connections to be maintained in an hardware implementation is a key challenge.

2.2.1 Multipath. Over the past decade, significant efforts has been made in improving load balancing in data center networks. There are generally two approaches: 1) one is to break down a single flow into multiple subflows, e.g. 4 or 8, and maintains a separate congestion window for each of the subflows, i.e., MPTCP [19]; 2) the other approach is to spray packets across a large numer of paths, e.g. 128 or 256, and

maintaining one congestion window across all paths [41]. The advantages of the subflow approach is that it keeps in-order packet delivery within each subflow. However, due to the per-path congestion state required, this approach limits the number of subflows that can be efficiently maintained in hardware in parallel.

If packets are sprayed over a large number of entropies (paths), the workload can be distributed more evenly. However, even with larger number of entropies, oblivious packet spraying can still perform poorly if the network is asymmetric and the hash collision can not be eliminated completely. Adaptive packet spray takes into account paths' congestion. The challenge is how to keep the path congestion state simple and efficient. If one congestion window governing traffic sent over all paralleled paths, the additional challenge is how multipathing co-ordinates with the congestion control algorithm as various paths could experience different degrees of congestion.

2.2.2 Congestion Management.
Congestion management for AI/ML clusters is unique: the highly paralleled paths dictate a fine-grained packet spraying approach to achieve high network utilization. It also means that the traditional congestion control algorithms, which apply window control to a single path, can't be adopted here as each path often carries a single packet of a flow during an RTT. Hence, we need to design a mechanism that manages the overall traffic across many different paths or entropies and allows the sender to switch between different paths in order to fully utilize the available capacities across the entire network.

Ideally, the first course of action upon a moderate congestion on a few paths should be shifting some traffic from the over-loaded path to the under-loaded ones. Congestion control should not step in and start cutting the window as reducing the window would limit the number of packets that can be sent to other under-utilized paths. Instead, it diverts some traffic from the over-loaded paths to the under-loaded ones. Congestion control would be necessary during fabric congestion because of over-subscription, network asymmetry, or incast event when multiple senders send to a single receiver.

The design challenge is deciding when to switch paths and when to cut the window. We believe that the earliest congestion signal, indicating onset of congestion, should be used to change path. Without the assumption of network telemetry measures, we make the following key observation: modern switches adopt ECN marking at egress, when a packet exits a queue [49]. This particular packet may not have experienced queueing delay, but its ECN mark indicates the queue behind it is building up. This would give us the earliest congestion signal, much faster than RTT or even the change of RTT. Figure 4 shows a simulation of a transient
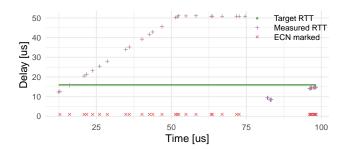

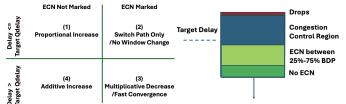**Figure 4: Congestion Signals During an Incast.**

congestion event, when 32 different sources send traffic to one receiver simultaneously. We measure ECN-marking as well as RTT as time progresses at a sender. As shown in Figure 4, the first received ACK packet for this sender is already ECN-marked while the measured RTT is at base RTT, indicating this particular packet has not experienced any congestion but the queue behind it has built up. The measured RTT or any noticeable RTT change does not happen until around 16us, much later after the ECN marked ACK packet has arrived.

With the above observation, we believe that ECN is the most timely signal for adaptive packet spraying. Only when a majority of paths experienced congestion, we would cut the window. As window adjustment is an quantitative decision, that's why we take RTT, a multi-bit signal, as the congestion signal for congestion control.

2.2.3 Reliability.
Packets losses due to bit errors, buffer overflow, or device failures are common in data center networks. Transport layers uses packet retransmission to guarantee reliable packet delivery. A major departure from RoCEv2 in our approach is the assumption of lossy networks. In lossy networks, RoCEv2's Go-Back-N loss recovery mechanism becomes very ineffient. A mechanism that performs selective retransmission, i.e., only dropped packets are retransmited, is required. Selective retransmission requires timely ACK packets in order to track the packet receiving state. Due to multipathing, packet arrivals are often out of order. It is a challenge to detect packet losses quickly while minimizing spurious retransmissions.

Ideally, acknowledging every arrival packet can provide the timely reliability information and does not require tracking states at a receiver. However, due to the NIC packet processing performance pressure as the link speed increases [46], it is more effient to coalease ACKs, i.e., the receivers generate ACKs after receiving a certain number of packets or due to certain special events.

These coalesced ACKs relay packet arrival information back to the sender, referred to as **SACK**. Similar to previous works [33, 35], a bitmap to monitor the packet arrival status

Figure 5: Congestion Management Quadrant.

	ECN Not Marked	ECN Marked
Delay <= Target Qdelay	(1) Proportional Increase	(2) Switch Path Only /No Window Change
Delay > Target Qdelay	(4) Additive Increase	(3) Multiplicative Decrease /Fast Convergence

Figure 6: Congestion Management Regions of Actions.

Target Delay

Drops
Congestion Control Region
ECN between 25%-75% BDP
No ECN

Figure 7: Additional fields in ACK packet.

Byte 0	Byte 1	Byte 2	Byte 3
echo_entropy	echo_timestamp		
recv_bytes			
Expected PSN			ooo [0:7]
ooo [8:11]	reserved	SACK base PSN	
SACK bitmap[0:31]			
SACK bitmap[32:63]			

is often adopted. However, a single SACK packet cannot transmit the entire bitmap to the sender due to packet header size limitations. Consequently, selectively choosing which bitmap segment to send back to the sender to provide timely packet arrival information is one of the challenges that we must address.

Packet loss detection poses another issue in a multipath context, as the sender cannot determine whether a missing packet is simply delayed in the network or has been dropped. Timeout is a common method for detecting lost packets. However, timeout periods are often set high enough to account for queuing delays and unpredictable server/NIC processing time; otherwise, senders may experience spurious retransmissions. On the other hand, high timeout values can lead to undesirably long flow completion times. How to quickly and accurately detect silent packet losses is a major obstacle that needs to be solved.

3 STrack Design

This section specifies three components of STrack's congestion management for reliable packet delivery: 1) adaptive packet spray in Section 3.1; 2) sender-based congestion control for fabric and moderate scale incast congestion (up to 100:1) in Section 3.2; and 3) an selective retransmission algorithm in Section 3.3 that takes into account out-of-order packets due to multipath. Our goal is to keep the design as simple as possible and does **not require** any advanced switch capabilities, such as Packet Trimming, Back-to-sender, In-network Telemetry; yet can work with them.

The congestion management in STrack handles congestion-aware multipathing and window-based congestion control jointly. Without the assumption of advanced switch features, STrack has two basic congestion signals to work with: a) egress-marked ECN; and b) RTT; Timely egress-marked ECN information is used to guide which paths shall be used or be avoided. The multi-bit congestion measures, RTT, is used to precisely control the congestion window. Based on these two signals, there are four scenarios that STrack handles as shown in Figure 5: 1) the acked packet is not ECN marked and its RTT is below the target RTT; 2) the acked packet is ECN marked but its RTT is still below the target RTT; 3)

ECN is marked and its RTT is also above the target RTT; 4) ECN is not marked but its RTT is above the target RTT.

In Scenario #1, the network is not congested, which is the most common network state. Hence, STrack would not interfere as long as the congestion window is at the maximum value, roughly the BDP. If the window is not at the max due to prior congestion episode, STrack increases the congestion window proportionally based on how far the RTT is from the target RTT. If the network is slightly congested, reflected by an marked ECN but with low RTT as in Scenario #2, STrack chooses to switch path while keeping the congestion window intact to allow packets flow to other paths. When RTT becomes large indicating network wide congestion or incast scenarios, STrack cuts window multiplicatively in #3. When RTT is much larger than the target, STrack uses additional congestion signal, achieved Bandwidth Delay Product (BDP), total acknowledged bytes in one base RTT, to quickly reaches the operating point in heavy incast scenarios. This allows STrack to converge much faster than the conventional AIMD approach. When congestion goes away, a packet that has experienced long queueing delay will not be ECN marked as no queue built up behind it. In this Scenario #4, STrack actually starts increasing the window additively to avoid starving the link. The ECN marking threshold and the target RTT determine when STrack transitions from Scenario #2 to Scenario #3.

The design has a clear separation between the congestion control and multipathing, i.e., the ECN marking threshold is between 25% and 75% of Bandwidth Delay Product (BDP) and the target queuing delay is one network RTT, as shown in Figure 6. Initially, we only switch paths based probabilistic ECN marking. As the network congestion increases, the majority of packets are ECN marked and their RTT increases which would trigger congestion window cut. The adaptive packet spray is still on-going even if with a window cut. Note that Figure 6 reflects the symbolic queueing view from congestion control's angle while multiple paths' queue are involved in a real network.

STrack assumes that switches drop packets silently when no buffer space is available, i.e., trimming [12, 23] or Back-to-sender[6, 29] is not present. Furthermore, STrack's reliability

Algorithm 1 STrack Overview

1: **procedure** ON_SENDING_PACKET(packet)
2:     packet.path_id = choose_path()
3: **end procedure**
4: **procedure** ON_RECEIVING_ACK
5:     ecn = ack.ecn
6:     path_id = ack.path_id
7:     **update_ecn_bitmap(ecn, path_id)**
8:     **if** measured_rtt < base_rtt **then**
9:         base_rtt = measured_delay
10:     **end if**
11:     measured_Qdelay = measured_rtt - base_rtt
12:     probe_timer_ts = now + 3* net_base_rtt
13:     **if** ack_for_probe *and* delay < 2* net_base_rtt *and* achievedBDP == 0 **then**
14:         **retransmit_packets()**
15:     **end if**
16:     achievedBDP = **update_achievedBDP**(ack, now)
17:     **cwnd = adjust_cwnd**(ecn, measured_Qdelay, achievedBDP)
18:     **loss_detection(ack)**
19: **end procedure**

Algorithm 2 STrack Adaptive Load Balance

1: **INIT:** max_paths=256, bitmap[0...max_paths]=[0], rr=0, next_path_id = -1;
2: **procedure** UPDATE_ECN_BITMAP(ecn, path_id)
3:     **if** ecn **then**
4:         next_path_id = -(path_id)
5:         bitmap[path_id] = 0
6:     **else**
7:         next_path_id = path_id
8:         bitmap[path_id] = 1
9:     **end if**
10: **end procedure**
11: **procedure** CHOOSE_PATH
12:     **if** next_path_id > 0 **then**
13:         rr = next_path_id
14:         next_path_id = -1
15:     **else**
16:         flag = false
17:         paths = min(max_paths, 2*cwnd)
18:         paths = max(8, paths)
19:         rr = (rr + 1)%paths
20:         **while** $bitmap[rr] \neq 0$ **do**
21:                         ▷ one packet only clears one bit.
22:             **if** !flag **then**
23:                 $bitmap[rr] = 0$
24:                 flag = true
25:             **end if**
26:             rr = (rr + 1)%paths
27:         **end while**
28:     **end if**
29:     **return** rr
30: **end procedure**

mechanisms must also align with coalescing ACKs [26, 42] to accommodate the reality that packet processing rates grow at a slower pace than link speeds. STrack provides a packet loss detection mechanism (Section 3.3): (1) out-of-order packets counters; (2) probing-based approach; (3) timeout. Once loss is detected, the retransmited packets are sent if the congestion window is allowed.

Algo 1 shows the overall-design of STrack. We assume an ACK packet echoes back the ecn mark, path_id and timestamp to the sender. Upon receiving an ACK, STrack checks: 1) whether an ECN is marked or not; 2) measured queuing delay (Line #8); 3) Received ACKed bytes (Line #13), which are our congestion signals. STrack updates the ECN bitmap with the echoed path_id and ecn signal (Line #4), which are used by multipath load balancing algorithm (in Algo 2). The main congestion control algorithm **adjust_cwnd** is specified in Algo 3, which we describe separately later. The packet loss detection algorithm is described in Section 3.3. Upon sending out a packet, we choose the right path id (Algo 2), which is covered later.

3.1 Adaptive Load Balancing

Algo 2 explains STrack adaptive load balancing algorithm. It initializes a bitmap with all the paths available, i.e., set bit as 0 and set the *next_path_id* as invalid, i.e., a negative value.

Upon receiving an ACK packet, STrack remembers the entropy value that is returned ecn-free in next_path_id and label the path_id as good path in the bitmap (Line #7-8 in Algo 2). When sending data packets, its initially uses entropy values in a round-robin fashion until an ACK is received. If the ACK is not ECN-marked, its entropy would be used for the packet that is being sent (Line #13 in Algo 2). Otherwise, it checks the bitmap in a round-robin fashion. The total paths are dynamically changed with the congestion window (Line #17 in Algo 2). This is because the path congestion information is fresh within a window. For example, if we maintain a bitmap size of 256, while the congestion window is one, it requires 256 RTTs to update the whole bitmap, and the congestion information is already outdated when the round-robin turns to a particular entropy. While checking the bitmap, if the path is ecn-marked, we skip the path and clear the bit for the first skipped path (Line #23 in Algo 2).

3.2 Congestion Control

Algo 3 shows the design of STrack's congestion control algorithm. Table 1 lists the configuration parameters for the performance tuning and use bdp_sf and $delay\_sf$ as BDP scaling factor and delay scaling factor to adapt to various network speeds and network latency. $target\_Qdelay$ is target queuing delay that we expect the congestion control to

Constant	Meaning	Recommend Values
target_Qdelay		net_base_rtt = 12 μs
target_Qhigh	a higher target queuing delay threshold	3*target_Qdelay
ewma	Used for the RTT averaging	0.125
bdp_sf	bdp scaling factor	BDP/(100Gbps*12μs)
delay_sf	delay scaling factor	base_rtt/12 μs
β	Used for additive increase	5*MTU*bdp_sf
η	Used for fairness	0.15*MTU*bdp_sf
α	Used for the RTT gain increase	4.0*bdp_sf* delay_sf*MTU/base_rtt
γ	Multiplicative decrease parameter	0.8

Table 1: Parameters of STrack.

Algorithm 3 STrack Congestion Control

1: **procedure** ADJUST_CWND(ecn, delay, achievedBDP)
2:     can_decrease = now - last_decrease_ts > base_rtt
3:     can_fairness_shuffle = now - last_selfai_ts > base_rtt
4:     avg_delay = avg_delay*(1 - *ewma*) +*ewma**delay
5:     **if** !ecn and delay > *target_Qhigh* **then**
6:         cwnd ← cwnd + $\frac{\beta}{cwnd}$
7:     **else if** !ecn and delay < target_Qdelay **then**
8:         cwnd ← cwnd + α*(target_Qdelay - delay)/cwnd
9:     **else if** can_decrease **and** avg_delay > target_Qdelay **then**
10:         **if** delay > *target_Qhigh* **and** achievedBDP < max_cwnd/8 **then**
11:             cwnd = achievedBDP
12:             last_decrease_ts = now
13:         **else if** delay > target_Qdelay **then**
14:             cwnd = cwnd* max(1 - γ * (avg_delay - target_Qdelay)/avg_delay, 0.5)
15:             last_decrease_ts = now
16:         **end if**
17:     **end if**
18:     **if** can_fairness_shuffle **then**
19:         cwnd = cwnd + η
20:         last_selfai_ts = now
21:     **end if**
22:     return cwnd
23: **end procedure**

---

be stabilized at. The algorithm maintains one congestion window to handle all the paths congestion.

For efficiency, STrack increases the rate in two different situations. First, when ecn is not marked and current measured queuing delay (**delay** in Algo 3) is below the target_Qdelay. This happens when a congestion episode has just passed, the algorithm increases the rate that is proportional to the difference between current RTT and the target_delay (Line #8). Secondly, we increase the rate by a constant value when

---

Algorithm 4 STrack AchievedBDP

1: **procedure** UPDATE_ACHIEVEDBDP(ack, now)
2:     can_clear_byte = (now - rxcount_clear_ts) > (base_rtt + target_Qdelay)
3:     rx_count += ack_for_probe? 0, acked_bytes
4:     **if** can_clear_byte **then**
5:         achievedBDP = rx_count
6:         rx_count = 0
7:         rxcount_clear_ts = now
8:     **end if**
9: **end procedure**

---

ecn is not marked but measured queuing delay is twice the target delay (Line #6). This is a situation when this packet incurred a long delay, but the queue length has come down drastically (ecn is not marked), we should increase the rate to avoid link starvation. There are two cases where we reduce the window: depending on the current measured queuing delay is greater than *target_Qhigh* or not. If it is greater, we use the achieved BDP (**achievedBDP** in Algo 4) as our window to quickly converge to a lower rate (Line #11). Otherwise, we do multiplicative decrease based on how much average measured queuing delay (**avg_delay** in Algo 3) is over the target_Qdelay (Line #14). To ensure fast convergence of fairness, we periodically add a constant small increase to the congestion window (Line #19). Note that this addition would assist the algorithm to have a slightly bigger window in order for packets to explore different paths.

3.3 STrack Reliability

Without any advanced features (e.g., trimming or BTS), switches drop packets silently. In STrack, we design a packet loss recovery mechanism that can swiftly detect and retransmit silent packet drops. In addition, STrack's reliability mechanism also aligns with coalescing ACK mechanism [26, 42] to accommodate the reality that packet processing rates grow at a slower pace than link speeds. STrack provides a packet loss detection mechanism across three different time scales: (1) one or two RTTs using out-of-order packets counters; (2) multiple RTTs with probes; (3) tens of RTTs using retransmission timeout. Once a loss is detected, the packet is retransmitted as long as the congestion window allows.

3.3.1 Coalescing ACKs and SACKs. Upon the arrival of each packet, a receiver sends an Selective Acknowledgment (SACK) packet if one of the following conditions are met: (1) a sufficient number of bytes have been received; (2) a packet with current expected Packet Sequence Number (EPSN) is received; (3) a probing packet is received.

SACK Packet. Receiver maintains a bitmap that starts from EPSN, and each bit in the bitmap tracks packet arrivals following EPSN. Packets prior to the expected PSN have been received. Due to the constrained bitmap length that can be conveyed in a single SACK packet sent to the sender, the receiver cannot transmit the whole bitmap in one SACK. The complete bitmap is partitioned into segments, each corresponding to the size of a bitmap that a SACK can accommodate. For instance, if the entire bitmap consists of 256 bits and the SACK bitmap size is 64 bits, there will be four segments of bitmaps. The receiver should selectively pick the bitmap segment in SACK in order to collectively represent the comprehensive bitmap as efficiently as possible.

As mentioned before, a SACK packet echoes back entropy (i.e., path_id), ECN mark, and timestamp back to the sender. In addition, a SACK packet is designed to include the following fields: (a) Expected PSN; (b) Selective ACK Base PSN (Base PSN for Selective ACK bitmap field); (c) Selective ACK Bitmap; (d) bytes_recvd, indicating total received bytes at receive with duplicates eliminated; (e) Out-Of-Order (OOO) packets counter, representing the number of packets received at the receiver since the EPSN. These additional fields are shown as in Figure 7.

The receiver also records the lowest PSN received, LPSN, since the last SACK. The next SACK is sent with the bitmap segment that is associated with LPSN as retransmission timeout is always triggered from the lowest unacknowledged PSN. Hence we need to update LPSN as early as possible to prevent the sender from timing out.

3.3.2 Loss Detection. A sender also maintains a bitmap to keep track of packets that have been cumulatively and selectively acknowledged from the SACK bitmap segments. The sender calculates the inflight bytes as follows:

$$inflight\_bytes = bytes\_sent - bytes\_recvd \quad (1)$$
$$- bytes\_claimed\_retransmit. \quad (2)$$

Upon receiving a SACK, the sender updates the inflight bytes based on the SACK information. The inflight should always be smaller than the congestion window. STrack enters loss recovery mode, i.e., loss is inferred, when any of the following conditions meet: (1) the out-of-order packets counter exceeds a threshold; (2) probe-based loss detection; (3) a retransmission timeout occurs.

OOO-based Detection. Loss is inferred if the out-of-order packets counter carried at SACK exceeds a threshold. The threshold is recommended to be set to the maximum of the current congestion window's worth of packet number and min_threshold (say 5). This constant threshold prevents spurious retransmissions when the congestion window is less than min_threshold packets. The rationale for using a

threshold based on the congestion window is that if a packet does not get acknowledged within a whole window's time, it is safe to consider it as lost. This also helps us avoid a large bitmap size at receiver/sender

When the loss detection is triggered, STrack sender enters fast loss retransmission, and records EPSN and current highest received PSN; and considers any unacknowledged packets between EPSN and the highest received PSN are lost.

As long as the congestion window allows, the sender retransmits unacknowledged packets starting from EPSN. Upon receiving new SACKs, it updates EPSN and SACK bitmap and sends the remaining unacknowledged packets. The process continues until all the packets between the original EPSN and the recorded highest received PSN are ACKed. Then, the sender exits the loss recovery mode.

Probe-based Detection. A probe packet is sent when the sender does not get any SACK over a specific duration, ideally set as n (e.g. n = 3) network wide base RTTs. A probe packet can indicate the SACK bitmap to be returned using a specific SACK base. In common cases, SACK base will be EPSN. Upon receiving a probe packet, the receiver is expected to issue a SACK using the SACK base immediately. A dedicated bit is used to indicate that the SACK is triggered by the probe packet.

If an SACK packet for probe packet comes back and the measured RTT is small, and no other ACK is received after the probe is sent (Line #10 in Algo 1), the sender gets into the loss recovery mode and using the same way as ooo-based method to recover from packet losses. In case the probe packets are also lost, the sender will send another probe packet transmission if it doesn't receive the SACK within the specified period, n times network wide base RTTs.

Timeout-based Detection. A single timestamp per flow is maintained at the sender to monitor retransmission timeouts. This timestamp is initialized upon the creation of the connection and is updated whenever a higher EPSN is received. If the timeout period has passed without the sender receiving the acknowledgement for current EPSN packet, all the unacknowledged packets are declared lost and retransmitted. The timeout value is generally configured in hundreds of microseconds or, at the very least, tens of microseconds.

4 Evaluation

We evaluate STrack through large-scale simulations using synthetic traffic and collective workloads to answer the following questions:

- How does STrack's adaptive packet spraying impacts the tail latency (Section 4.2)?
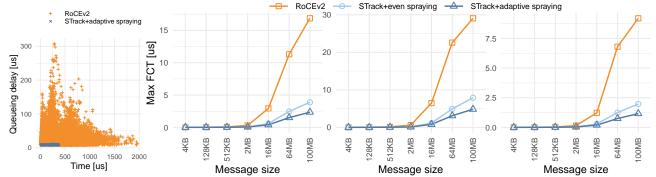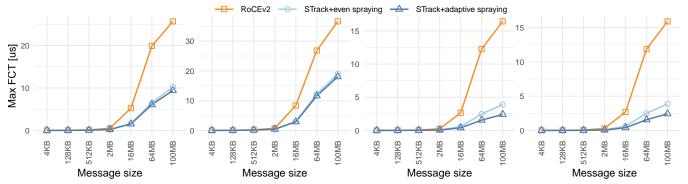
Figure 8: Switch link queuing delay.

Figure 9: 8K nodes permutation, 400Gbps.

Figure 10: 8K nodes permutation, 200Gbps.

Figure 11: 8K nodes permutation, 800Gbps.

Figure 12: 8K nodes permutation, 4:1 over-subscription.

Figure 13: 8K nodes permutation, 8:1 over-subscription.

Figure 14: 8K nodes permutation, 64 links down.

Figure 15: 8K nodes permutation, 256 links down.

- How does STrack's joint optimization of adaptive load balancing and congestion control perform under oversubscribed as well as asymmetric networks (Section 4.2)?
- How does STrack handle loss recovery upon high incasts and what is its impact on switch buffer occupancy (Section 4.2)?
- How does STrack impact the AI/ML workloads' communication performance (Section 4.3)?

4.1 Experiment Setup

We compare STrack with RoCEv2 [25], which is widely deployed transport over Ethernet in AI/ML network. We configure the constants for STrack congestion control as suggested in Table 1 and set the maximum number of paths as 256 for adaptive load balancing and switch ECN mark threshold $K_{min} = 25\%BDP$ and $K_{max} = 75\%BDP$. Switch is operated without any advanced features enabled, and drops packets if queuing exceeds 5 BDP. RoCEv2 utilizes DCQCN [49] as its primary congestion control mechanism and supports single-path operation. Additionally, it necessitates Priority Flow Control (PFC) to be enabled throughout the entire network to prevent packet loss caused by buffer overflow. We implemented shared buffer architecture for PFC to best demonstrate the performance of RoCEv2. We configured the total switch buffer size to be 256MB for a total capacity of 51.2Tbps and scaled the buffer size accordingly for switches with different radices. For instance, the switch buffer size is set to 128MB for a 25.6Tbps switch. We set the ECN threshold to one BDP for DCQCN and configure Lossless for RoCEv2.

Switch has ECMP [15] as the routing strategy. We extend htsim [9] simulator to implement the RoCEv2 and STrack as well the switch models.

Performance metrics. For the synthetic workloads, we use tail flow completion time (max FCT) as our application performance metric. The FCT is measured from the time the message is ready to be pushed to the networking stack to the time when the last packet of that message gets acknowledged at the sender. For the collective workloads, we measure the collective completion time, which is measured when the first message of the collective sent to the time when the last message finishes.

9

4.2 Synthetic Traffic

We use a 2-tier standard fattree topology [17, 18, 40] with 8192 nodes for the evaluation with MTU size of 4KB. Our tests vary the link speed for 200G, 400G and 800G and the default link speed is 400G. The synthetic workloads include permutation traffic pattern and incast traffic pattern. The network topology is configured with (1) full-bisectional network; (2) over-subscription ratio of 4:1 or 8:1; (3) asymmetric network. The asymmetric network is created by disabling 0.78% or 3.1% of inter-switch links. The network wide RTT is 8μs, and thus, the network bandwidth-delay-product (BDP) for 400Gbps is 400KB. The message size varies from 4KB, 128KB, 512KB 2MB, 16MB, 64MB and 100MB.

Permutation. We utilize the permutation traffic pattern to demonstrate effective load balancing. This pattern is created by randomly pairing two nodes for each flow, ensuring that any given node serves as the receiver for one flow and the sender for another. For this traffic pattern, if the network load balancing is perfect or nearly perfect, congestion control should not reduce the window size. This workload is ideal for showcasing the effectiveness of the network load balancing scheme, and demonstrate how congestion control interacts with load balancing. Given there are a total of 8K nodes in the network topology, the permutation workload has a total of 8K flows and we report the maximum flow completion time over the 8K flows. Figure 9 shows the maximum flow completion time of the various message size for STrack, STrack's congestion control with oblivious packet spraying and RoCEv2.

Figure 9 shows the max flow completion time of the various message size for STrack, STrack's congestion control with oblivious packet spraying and RoCEv2 with the default link speed of 400G. STrack improves the max FCT over RoCEv2 up to 6.3X, as RoCEv2 traffic uses one path and is highly impacted by hash collisions. As a result, RoCEv2 is not able to leverage the idle parallel paths to achieve higher throughput. In addition, adaptive packet spray improves the max FCT over oblivious packet spraying up to 33%, when the message size is 100MB. STrack's adaptive packet spraying requires at least one RTT to get the congestion signal, i.e., ECN, back to sender to skip the bad path. Thus, the performance differentiation for oblivious packet spraying and adaptive packet spraying starts to show only for larger message sizes, e,g, after 16MB.

Figure 8 samples the switch queuing delay as the simulation time progresses in the 16MB experiment in Figure 9 for STrack and RoCEv2. There are a total of 16K switch queues, 8K queues from ToR to Spine Switch and the other 8K queues from Spine to ToR. To avoid too many logs to be processed, we only generate logs if the queuing delay is larger than 8us (the base RTT). Hence, if we do not see any points at
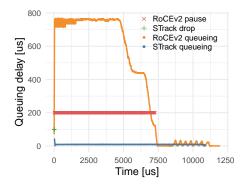
one time moment in Figure 8, it means that across the 16K queues, there is not any queue whose queue depth exceeds the BDP at that time point. Figure 8 shows that, initially before 370us, STrack's adaptive packet spraying experiences mild congestion due to hash collisions, but it adapts to skip the bad paths, and finds the good paths. Hence after 370us, all the flows settle down to the good paths, and thus, the switch queuing delays do not exceed 8us after 370us. In contrast, RoCEv2 continues experiencing hash collisions, which leads to the queuing delay up to 250us until the end of experiment.

To demonstrate the robustness of STrack, we also run the same workloads with link speeds of 200Gbps and 800Gbps. Figure 10 and 11 shows the maximum FCT for different message sizes for 200Gbps and 800Gbps, respectively. STrack adaptive packet spraying outperforms oblivious packet spraying as message size increases. STrack adaptive packet spraying consistently outperforms RoCEv2 starting from message size 128KB and gets the performance improvement by up to 6X when the message size is 64MB and 100MB.

Permutation with over-subscribed network. Network over-subscription is common in data center networks, as it can reduce the cost for the network devices, e.g., switches and links. We change the full bi-sectional 2-tier network topology to be the 4:1 or 8:1 oversubscribed, where the total bandwidth from the ToR switches to the spine switches is 1/4 or 1/8 of total bandwidth from host to ToR switches. We use the permutation traffic pattern as the workloads. This setup evaluates the effectiveness of joint optimization of congestion control and multipath load balancing mechanism.

Figures 12 and 13 display the maximum flow completion time (FCT) for various message sizes using STrack and RoCEv2 in 4:1 and 8:1 over-subscription networks, respectively. STrack consistently outperforms RoCEv2 by up to 3X across all message sizes. STrack's adaptive packet spraying mechanism effectively balances the load across inter-switch links, fully utilizing all available path bandwidth. Additionally, its congestion control reduces the window size by factors of 4 or 8 to accommodate the 4:1 or 8:1 over-subscription. In contrast, RoCEv2's single path strategy suffers from heavy hash collisions, leading to network congestion, a reduction in sending rate by DCQCN, and under utilization of the network. As congestion control plays the main role in driving the performance in over-subscription scenarios, we observe that adaptive packet spraying and oblivious packet spraying exhibit similar behavior due to their use of the same congestion control algorithm.

Permutation with Asymmetric network. Network link failures are common in data centers [40]. To simulate an asymmetric network, we disconnect the links between ToR and spine switches. We fixed the number of switches experiencing link failures at 16 and disconnected either 64 (0.78%)
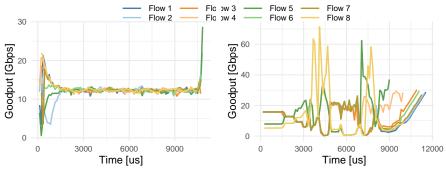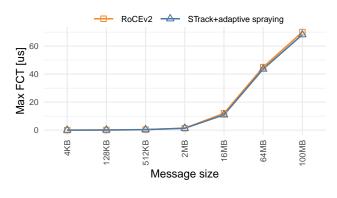
Figure 16: 32->1 incast, Queuing delay, drop and pause as time goes.

Figure 17: 32->1 incast, STrack throughput over time.

Figure 18: 32->1 incast, RoCEv2 throughput over time.
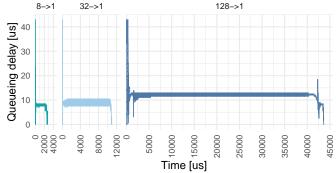

Figure 19: 32->1 incast, message completion time

Figure 20: STrack last-hop switch queuing delay for 8->1 incast, 32->1 incast, 128->1 incast as time goes.

or 256 (3.1%) out of a total of 8,192 inter-switch links. This corresponds to 4 links per ToR switch for the 64-link down scenario and 16 links per ToR switch for the 256-link down scenario. We ran the same permutation traffic pattern on this network to validate the efficiency of STrack multipath load balancing.

Figures 14 and 15 show the maximum flow completion time for an $8K$-flow permutation with 64 and 256 link failures. Both oblivious packet spraying and adaptive packet spraying consistently outperform RoCEv2's single-path transport by up to 6X, particularly when 64 links are down and the message size is 100MB. We do not observe a significant performance decrease despite these link failures. Additionally, adaptive packet spraying gains up to a 60% performance improvement over oblivious packet spraying when 64 links are down and the message size is 100MB.

Incast. We run experiments for 7 message sizes the same as Figure 9-15 for the 8-to-1 incast, 32-to-1 incast, and 128-to-1 incast, respectively. Figure 16 shows the last-hop switch queuing delay, packets drop for STrack and pause generation time for RoCEv2 as time goes in a 32-to-1 incast experiment

with 16MB. RoCEv2 shows persistently high switch queuing delay up to 7500us (worth of 37.5MB switch queue depth) for 5000us, and takes 7800us to converge to the right rate, while STrack takes 95us to reach to the stable state, which shows the fast convergence of STrack. We also label the pause generation time (marked red) and packets drop time (marked green) in Figure 16. The switch keeps sending the pause frames until the convergence point with RoCEv2, which is due to the slow convergence of DCQCN. Because of its fast convergence, STrack only drops packets at the first RTT, and there is not any more packet drops after the first RTT.

Figure 17 and Figure 18 show the instantaneous throughput of 8 flows, averaged over 100 microsecond intervals, for STrack and RoCEv2 respectively, same as the experiment depicted in Figure 16. Figure 17 demonstrates that the 8 flows with STrack can achieve their fair share of a bottleneck link, and converge to a new, stable bandwidth share. Conversely, RoCEv2 exhibits fairness issue in Figure 18, with PFC pauses, whose on and off behavior also prevents each flow from quickly attaining its fair share.

Figure 19 shows the maximum flow completion time of STrack and RoCEv2 for each message size with 32->1 incast.

As RoCEv2 relies on lossless link support, the bottleneck bandwidth of the last hop is fully utilized. Although pauses are generated, they tend to cause collateral damage to victim flows that share the uplink bandwidth. For the flows that go to the incast destination, RoCEv2 can achieve near-optimal flow completion time. As a lossy protocol, STrack incurs losses during an incast event, especially for the first RTT. The challenge is to recover quickly from these losses. As shown in Figure 19, STrack can match RoCEv2's performance by finishing about the same time across various message sizes, indicating that STrack's fast packet loss recovery mechanism allows its end-to-end performance to align with that of the lossless network.

Figure 20 shows the queuing delay of last-hop ToR switch for 8->1, 32->1, 128->1 incast experiments with message size of 16MB. STrack is able to stabilize at the target queuing delay across different incast degrees, which shows the robustness of the STrack's congestion control. Note that the stabilized point of 128->1 incast is is slightly higher than the target queuing delay, i.e., 8μs, this is because the ideal congestion window in this case is 1.3 packets. Due to rounding error, the inflight packets tend to vary between 1 or 2 packets, which causes slightly elevated queueing delay.

4.3 AI/ML Workloads

Collective algorithms in machine learning training systems are designed to avoid network incast [11, 12, 27]. However, network imbalance and system jitter can affect the ideal message scheduling and degrade system performance. In this section we evaluate STrack on a representative set of ML workloads. To model the communication scheduling of Deep Learning (DL) training workloads, especially collectives [5, 37], we generate a message trace based on the three AllReduece algorithms, i.e., DoubleBinaryTree (**DBT**), Ring (**RING**), and HalvingDoulbing (**HD**), and a AlltoAll collective (**A2A**).

The AllReduce traces express message dependency, where messages from later steps are sent only after messages in previous steps are received. For instance, in the DoubleBinary Tree algorithm, a node in the middle layer of the tree only sends messages to its parent after receiving messages from its two children. The AllToAll traces are sequenced to prevent incast. For example, the first message from rank n is sent to rank $(n + 1)\%N$, and the second message at rank n is sent to rank $(n + 2)\%N$. For AlltoAll communication, we restrict the number of active parallel connections at both the sender and receiver to various levels, such as 16, 32, and 64. This helps to control the degree of incast and outcast from a network perspective.

The simulated topology is a full-bisection 2-tier Clos network with all links operating at the same speed as the default. We vary the link speed to demonstrate the robustness of

STrack. We only show the results for the network link speed of 400Gbps as the experiment results of different link speeds are similar. We assume one NIC maps to one GPU. Each message chunk size is 128KB to utilize the pipeline. We primarily assess a multi-job scenario involving 64 identical collectives running concurrently on a 2048-NIC cluster, with each collective group randomly distributed throughout the cluster.

Full Bisectional Network. Figure 21 displays the maximum collective finishing time for 64 allreduce collectives using different allreduce algorithms with STrack and RoCEv2 on a full bisectional network. STrack with adaptive packet spraying outperforms RoCEv2 with 1 QPS_PER_CONN by up to 3x and RoCEv2 with 4 QPS_PER_CONN by up to 27.4%, especially with the DoubleBinaryTree algorithm. The DoubleBinaryTree algorithm exhibits a 2:1 incast property, where the limitations of DCQCN become evident. RoCEv2 with 4 QPS_PER_CONN performs better than RoCEv2 with 1 QPS_PER_CONN across various allreduce algorithms, as the increased entropy improves performance, albeit with higher CPU cost. Figure 25 plots the CDF of 64 collective finishing time of DoubleBinaryTree of Figure 21 experiment. It shows that the 64 collectives achieves similar finishing times with STrack, while there is 26.6% difference between the smallest finishing time and the maximum finishing time with RoCEv2 with 4 QPS_PER_CONN, which indicates STrack not only has better tail latency of each collective, but also better fairness compares with RoCEv2. Figure 22 shows the maximum collective finishing time for 64 AlltoAll collectives across different levels of parallelism using STrack and RoCEv2. STrack with adaptive packet spraying outperforms RoCEv2 with 1 QPS_PER_CONN by up to 79.2% and RoCEv2 with 4 QPS_PER_CONN by up to 18.9%, particularly at a parallel degree of 32. Figure 26 shows the CDF of finishing times for 64 alltoall collectives with a parallel degree of 32. This further demonstrates that STrack offers better fairness compared to RoCEv2.

Oversubscribed Network. Figure 23 shows the maximum collective finishing time for 64 allreduce collectives using different allreduce algorithms with STrack and RoCEv2 on a 4:1 oversubscribed network. STrack with adaptive packet spraying outperforms RoCEv2 with 1 QPS_PER_CONN by up to 4.86X and RoCEv2 with 4 QPS_PER_CONN by up to 4.13X, especially with the DoubleBinary Tree algorithm. We observed that the collective completion time of STrack does not significantly increase in the oversubscribed network compared to that in the full bisectional network (Figure 21). This indicates that the traffic generated by the allreduce algorithm does not fully utilize the full bisectional network. However, in this scenario, the RING and HalvingDoubling configurations of RoCEv2 with 1 QPS_PER_CONN and 4
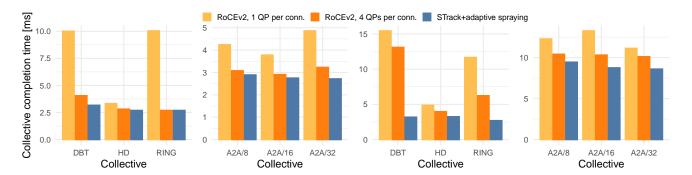
Figure 21: 64 AllReduce Collectives, full bisectional network.

Figure 22: 64 AlltoAll Collectives, full bisectional network.

Figure 23: 64 Allreduce Collectives, 4:1 oversubscribed network.

Figure 24: 64 AlltoAll Collectives, 4:1 oversubscribed network.
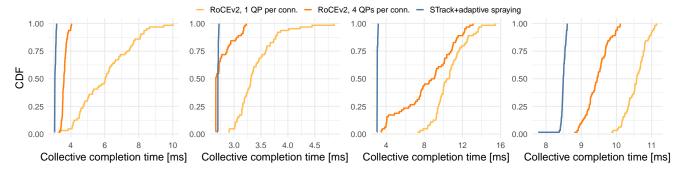

Figure 25: DBT, full bisectional network.

Figure 26: 32, A2A, full bisectional network.

Figure 27: DBT, 4:1 oversubscribed network.

Figure 28: 32, A2A, 4:1 oversubscribed network.

QPS_PER_CONN exhibit worse performance than that of the full bisectional bandwidth due to its poor load balancing and congestion control mechanisms. Although RoCEv2 with 4 QPS_PER_CONN performs slightly better than RoCEv2 with 1 QPS_PER_CONN across various allreduce algorithms, it does not match STrack's load balancing efficiency, triggering DCQCN to cut rate that leads to poor performance. Figure 27 shows the CDF of finishing times for 64 allreduce collectives with DoubleBinary Tree algorithm. Comparing to RoCEv2, STrack shows much lower collective finishing time for each collective and offers better fairness.

Figure 24 shows the maximum collective finishing time for 64 AlltoAll collectives across different levels of parallelism using STrack and RoCEv2 on a 4:1 oversubscribed network. STrack with adaptive packet spraying outperforms RoCEv2 with 1 QPS_PER_CONN by up to 51.5% and RoCEv2 with 4 QPS_PER_CONN by up to 17.8%, particularly at a parallel degree of 16. Compared to the allreduce collectives (Figure 21), alltoall collectives require higher bandwidth for inter-switch links. Consequently, we observe approximately four times the maximum collective finishing time in the full bisectional network scenario. Figure 28 displays the CDF

of finishing times for 64 alltoall collectives with a parallel degree of 32 for STrack and RoCEv2. It demonstrates that STrack completes one all-to-all collective in 7.78 ms, while other collectives complete in 8.5 ms. This is because 16 of the 32 NICs for the earlier-finishing collective are located on the same ToR, thereby reducing its traffic to the bottleneck inter-switch links by one-fourth. As a result, STrack consistently shows superior performance and greater fairness compared to RoCEv2.

5 Related Work

Our work draws on the long history of congestion control and multipath load balancing mechanisms. Below, we classify various congestion control techniques based on the type of congestion signal they employ and their approaches to regulating transmission rates, distributing network traffic across multiple paths, and achieving joint optimization of these aspects.

5.1 Congestion Signals

The choice of the signal to use to detect congestion largely impacts the *responsiveness* of the algorithm.

Delay-based. The simplest signal is measured RTT, where either the sender measures the round trip time (e.g., Swift [28], Timely [34], TCP Vegas [8]) or the receiver annotates ACK packets with one-way delay information (e.g., OnRamp [32]). Accurate RTT measurement is critical and may require special OS or hardware support such as Snap in Swift [28]. Delay-based signals are often described as *"multi-bit"* as they provide rich information for exact queuing delay along the path in the network.

Explicit Congestion Notification (ECN). ECN is widely used in datacenters and is available in all modern switches. Algorithms like DCTCP [3], Hull [4], and D^{2}TCP [44] all rely on ECN marking. For example, DCQCN [49] combines ECN and *Priority Flow Control* (PFC) to react to congestion. However, although egress-marked ECN-based algorithm can react quicker than delay-based ones [50], they perform poorly when dealing with incasts and are usually hard to tune [31].

In-Network Telemetry. While ECN is a simple and widely available example for in-network telemetry (INT), other protocols rely on more complex signaling such as maximum queue occupancy along the path [45]. Modern switches offer a plethora of such signals and protocols using those are in active development. HPCC [31], for example uses precise load information along the path to adjust its rate at the sender.

Packet-Trimming and Back-to-Sender. Contemporary programmable switches offer queuing stats at their ingress points, enabling the application of cutting payload techniques to data packets [12, 23]. Additionally, switches can duplicate header packets and send them back to the endpoints [6, 29]. This approach requires the notification of network congestion status to endpoints even without experiencing ongoing congestion.

5.2 Congestion Management

5.2.1 Congestion Control Algorithm.
The decision on how to react to congestion can occur in different parts of the network, and this has a direct impact on the ***visibility*** and on which type of congestion the algorithm can react to.

Receiver-based. Some algorithms exploit the knowledge of the receiver about concurrently incoming flows to schedule the transmission by the different senders directly. Many of these schemes such as NDP [23], EQDS [38], pHost [20], ExpressPass [14], and Homa [36] use end-to-end credits back to the sender to control flows rates. However, whereas these schemes can deal very well with incast traffic, they usually fall short when dealing with congestion happening in the network fabric [23, 38].

Sender-based. Algorithms such as Swift [28], DCQCN [49], DCTCP [3], HPCC [31], Timely [34], Poseidon [45], and

others instead rely on the sender to adjust its transmission rate based on congestion information received from the receiver. In some schemes like Bolt [6], switches can directly indicate congestion back to the sender and thus shortcut the path to the receiver. These sender-based congestion control algorithms can respond to congestion occurring anywhere in the network, which is why they are widely used in data center networks today.

5.2.2 Multipath load balancing.
The multipath load balancing mechanisms can be categorized into host-driven solutions [24, 33, 48], switch-driven solutions [2, 21, 47] and centralized solutions [1, 7, 16, 39]. Here, we focus on closely host-driven related works.

Today's data centers primarily rely on single-path transport methods, such as TCP [11] and RoCEv2 [25], which are standard practices. However, these methods often encounter well-documented ECMP hash collisions. In contrast, MPTCP [19] addresses these challenges by breaking down a single flow into multiple sub-flows and intelligently distributing packets among these sub-flows based on network congestion conditions. Presto [24] breaks a flow even further into flowcells and picks paths in a round-robin fashion.

Another approach, known as Hermes [48], leverages ECN and RTT to differentiate among good, gray, and bad paths, although it depends on pre-installed switch routing to enforce specific paths. Lastly, Protective Load Balancing (PLB [27]) utilizes RTT and ECN to detect ECMP hash collisions and dynamically adjusts the flow entropy field to avoid congested paths.

5.2.3 Joint Optimization.
There have been limited prior work focused on jointly optimizing congestion control and multipath load balancing, which is the primary objective of our system. One example of such an approach is MP-RDMA [33], which employs packet spraying during the initial window, and handles fabric and incast congestion using ECN only. As a result, the algorithm converges slowly as it requires multiple RTTs to reach stability upon a congestion event. In a similar vein, Scalable Reliable Datagram (SRD [41]) utilizes congestion control techniques similar to BBR [10] and incorporates multipath load balancing through packet spraying. However, as a proprietary scheme, it is not clear the detailed algorithms and its loss recovery scheme.

STrack utilizes ECN for path selection and switches to paths that are devoid of ECN markings. It combines ECN and RTT in the congestion control algorithm to effectively tackle network congestion.

6 Conclusion

We tackle the challenge of managing collective communication for AI/ML workload without advanced switch supports, such as packet trimming or in-network-telemetry. We

propose and demonstrate that STrack, a NIC-side only and hardware-offloaded reliable transport protocol, can achieve ultra-low latency, very high bandwidth utilization and well balanced networks. By taking advantage of egress-marked ECN, STrack swiftly changing path upon early congestion detection and adaptively load balancing to utilize the overall available capacity in a multipath network. Congestion window control only comes in when a majority of the paths are congested when average RTT is elevated. With a unique error recovery mechanism, STrack also ensures rapid packet recovery within a multipath context. Our evaluation results show that highly utilized and well balanced network links are the key to ideal performance for AI/ML workloads. STrack outperforms RoCEv2 up to 6X with synthetic workloads and by 27.4% with collective workloads, even with the optimized RoCEv2 system setup.

Acknowledgments

References

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, 2010. USENIX Association.

[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*, SIGCOMM '14, page 503–514, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (DCTCP). In *SIGCOMM*, 2010.

[4] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, 2012.

[5] AMD. RCCL: Radeon Collective Communication Library, 2024. Accessed: 2024-07-06.

[6] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt: Sub-RTT congestion control for Ultra-Low latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 219–236, Boston, MA, 2023. USENIX Association.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, 1994.

[9] BROADCOM. htsim Network Simulator, 2023. Accessed: 2023-04-06.

[10] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016.

[11] Vinton G. Cerf and Robert E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

[12] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. Catch the whole lot in an action: Rapid precise packet loss notification in data centers. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, 2014.

[13] Wenxue Cheng, Kun Qian, Wanchun Jiang, Tong Zhang, and Fengyuan Ren. Re-architecting congestion management in lossless ethernet. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 19–36, Santa Clara, CA, February 2020. USENIX Association.

[14] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, 2017.

[15] Rob Coltun, Dennis Ferguson, and Joel Moy. The ospf not so stubby area (nssa) option. RFC 1587, March 1994.

[16] Andrew R. Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637, 2011.

[17] Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng, Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, Yiqun Guo, Xiaowei Jiang, Lingbo Tang, Yin Du, Yingya Zhang, Pan Pan, and Yuan Xie. Eflops: Algorithm and system co-design for a high performance distributed training platform. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 610–622, 2020.

[18] Jianbo Dong, Shaochuang Wang, Fei Feng, Zheng Cao, Heng Pan, Lingbo Tang, Pengcheng Li, Hao Li, Qianyuan Ran, Yiqun Guo, Shanyuan Gao, Xin Long, Jie Zhang, Yong Li, Zhisheng Xia, Liuyihan Song, Yingya Zhang, Pan Pan, Guohui Wang, and Xiaowei Jiang. Accl: Architecting highly scalable distributed training systems with highly efficient collective communication library. *IEEE Micro*, 41(5):85–92, 2021.

[19] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[20] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, 2015.

[21] 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 Special Interest Group on Data Communication*, SIGCOMM '17, page 225–238, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16.

[23] 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 ACM SIGCOMM 2017 Conference*, SIGCOMM '17, 2017.

[24] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based Load Balancing for Fast Datacenter

Networks. In *Proc. ACM SIGCOMM*, 2015.

[25] InfiniBand Trade Association. Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.1 Annex A17: RoCEv2. https://cw.infinibandta.org/document/dl/7781, 2014.

[26] Emily Johnson and Michael Brown. Analysis of tcp coalescing acknowledgment mechanism in high-speed networks. *Journal of Network and Computer Applications*, 120:56–67, 2018.

[27] Abdul Kabbani, David J. Wetherall, Gautam Kumar, Junhua Yan, Kira Yin, Masoud Moshref, Mubashir Adnan Qureshi, Qiaobin Fu, Van Jacobson, and Yuchung Cheng. Plb: Congestion signals are simple and effective for network load balancing. 2022.

[28] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. 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, page 514–528, 2020.

[29] Yanfang Le, Jeongkeun Lee, Jeremias Blendin, Jiayi Chen, Georgios Nikolaidis, Rong Pan, Robert Soule, Aditya Akella, Pedro Yebenes Segura, Arjun singhvi, Yuliang Li, Qingkai Meng, Changhoon Kim, and Serhat Arslan. Sfc: Near-source congestion signaling and flow control, 2024.

[30] Qiang Li, Yixiao Gao, Xiaoliang Wang, Haonan Qiu, Yanfang Le, Derui Liu, Qiao Xiang, Fei Feng, Peng Zhang, Bo Li, Jianbo Dong, Lingbo Tang, Hongqiang Harry Liu, Shaozong Liu, Weijie Li, Rui Miao, Yaohui Wu, Zhiwu Wu, Chao Han, Lei Yan, Zheng Cao, Zhongjie Wu, Chen Tian, Guihai Chen, Dennis Cai, Jinbo Wu, Jiaji Zhu, Jiesheng Wu, and Jiwu Shu. Flor: An open high performance RDMA framework over heterogeneous RNICs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 931–948, Boston, MA, July 2023. USENIX Association.

[31] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *PProceedings of the ACM SIGCOMM 2019 Conference*, SIGCOMM '19, 2019.

[32] Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. Breaking the Transience-Equilibrium nexus: A new approach to datacenter packet transport. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 47–63. USENIX Association, April 2021.

[33] 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 Networked Systems Design and Implementation (NSDI 18)*, pages 357–371, Renton, WA, April 2018. USENIX Association.

[34] Radhika Mittal, Terry Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-based congestion control for the datacenter. In *SIGCOMM*, 2015.

[35] 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 Special Interest Group on Data Communication*, SIGCOMM '18, 2018.

[36] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM*

[37] NVIDIA Corporation. NCCL: NVIDIA Collective Communications Library, 2024. Accessed: 2024-07-06.

[38] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, April 2022.

[39] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *SIGCOMM*, 2014.

[40] 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. Alibaba hpn: A data center network for large language model training. SIGCOMM '24, 2024.

[41] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A cloud-optimized transport protocol for elastic and scalable hpc. *IEEE Micro*, 40(6):67–73, 2020.

[42] John Smith. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Boston, MA, 2nd edition, 2010.

[43] Feng Tian, Yang Zhang, Wei Ye, Cheng Jin, Ziyan Wu, and Zhi-Li Zhang. Accelerating distributed deep learning using multi-path rdma in data center networks. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, SOSR '21, page 88–100, New York, NY, USA, 2021. Association for Computing Machinery.

[44] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, 2012.

[45] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. Poseidon: Efficient, robust, and practical datacenter CC via deployable INT. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 255–274, Boston, MA, April 2023. USENIX Association.

[46] 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. SRNIC: A scalable architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, Boston, MA, April 2023. USENIX Association.

[47] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 139–150, New York, NY, USA, 2012. Association for Computing Machinery.

[48] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient datacenter load balancing in the wild. SIGCOMM '17. Association for Computing Machinery, 2017.

[49] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale RDMA deployments. In *SIGCOMM*, August 2015.

[50] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. ECN or delay: Lessons learnt from analysis of DCQCN and TIMELY. In *Proceedings of 2016 ACM Conference on Emerging network experiment and technology (CoNEXT 2016)*, December 2016.