Simulate Before Sending: Rethinking Transport in Datacenter Networks

Dan Straussman
Technion, Israel

Isaac Keslassy
Technion, Israel
UC Berkeley, USA

Alexander Shpiner
Nvidia

Liran Liss
Nvidia

Abstract

Existing transport protocols in commodity datacenter networks struggle to provide low collective completion times (CCTs) to AI training collectives, as packet losses and retransmissions significantly degrade performance.

We propose DCSIM, an efficient transport that achieves low CCTs and *practically lossless* performance with commodity switches. In DCSIM, each packet first employs a small simulation probe to traverse the network and explore congestion along a candidate path. Only packets whose simulation probes succeed are then transmitted, expecting to succeed as well. Evaluations confirm that DCSIM achieves faster CCTs than existing schemes, with small queues and virtually zero packet loss. Finally, DCSIM also excels in adverse conditions, including oversubscribed topologies.

2012 ACM Subject Classification Networks → Data center networks; Networks → Transport protocols

Keywords and phrases Datacenter networks, transport protocols, AI training, lossless networks

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.19

Funding *Isaac Keslassy*: This work was partly supported by the Louis and Miriam Benjamin Chair in Computer-Communication Networks.

1 Introduction

Datacenters are increasingly designed for AI training. Large companies such as Google, Meta, OpenAI, AWS and Microsoft are planning to invest hundreds of billions of dollars to support AI workloads in their cloud infrastructure [35, 44, 7, 43, 20, 2, 45].

Unfortunately, due to their computation/communication cycles, AI training applications are very bursty, and can incur significant *packet drop rates* in lossy datacenter networks [47, 13, 46]. In addition, while AI training traffic patterns used to be more symmetric and predictable, recent mixture-of-experts (MoE) models exhibit unpredictable patterns that worsen packet drops [33, 32, 21, 22, 13, 30, 51].

These packet drops harm AI training performance in two ways. First, they cause stragglers due to packet retransmissions, and therefore increase collective completion times (CCTs), i.e., the time for the last packet to complete in a collective communication algorithm like Ring All-Reduce or All-to-All [47, 34]. Second, packet losses can also increase the tail latency of RDMA, a common building block for AI training applications designed for lossless networks [36, 16, 38, 52, 13, 27].

Thus, there is a need to design transports that lower CCTs by making commodity lossy networks practically lossless. To fully exploit the network capacity, many solutions use per-packet load balancing, including oblivious packet spraying, such as in Alibaba Stellar [12, 59, 18, 34]; adaptive packet spraying, such as in STrack, REPS, and the Ultra Ethernet Consortium specification [25, 9, 56, 47]; and adaptive routing, such as in NVIDIA's Spectrum-X [1, 39, 57, 41, 50]. These solutions use several reliability mechanisms to handle losses, e.g., NACK-based and selective-repeat mechanisms in extended RoCE-based protocols [16, 38, 52], or packet trimming [8, 42, 56, 27]. However, these schemes can experience poor performance when scaling [36, 27, 27]. Another approach, adopted by pHost [14] and dcPIM [10], is to rely on a matching algorithm where receivers grant tokens to senders. This approach is promising because it avoids losses due to receiver oversubscription, but it also lacks visibility into the network and is therefore vulnerable to oversubscribed networks and link failures.

Additional schemes show great potential for avoiding losses, but require non-commodity hardware. Rateless erasure coding can mask losses but needs specialized NICs to be implemented at high rates [24, 36]. ExpressPass introduces a receiver-driven credit-based scheme that avoids losses, but it relies on switch modifications, e.g., to ensure symmetric paths, and cannot handle multi-path and link failures [11]. Harmony offers another promising direction: using *reservations* [3]. Harmony uses per-flow fixed-bandwidth reservations to eliminate congestion-related drops while achieving high utilization. Unfortunately, it also needs specialized switches to participate in the reservation process, and struggles with low-rate and variable-rate flows that do not match the fixed reservation rates.

To achieve our goal of a practically-lossless transport running in a lossy network with commodity hardware, we want to use reservations and solve two significant challenges that currently limit their effectiveness. First, the reservations need to be more flexible, with *per-packet reservations* that allow the flow path to change upon congestion. Second, they should be made with *commodity switches* and thus be implementable at any datacenter.

We present DCSIM, an effectively lossless transport mechanism that can achieve low CCT for AI training with commodity switches, while being resilient to adverse network conditions. It relies on two core ideas to address the reservation challenges. The first one is a conceptual shift. Assume we had a *shadow simulation network* that could run at exactly 1/100th the rate of our real network, with load-balanced SIM (simulation) packets that are also 100 times smaller than our real-network DATA packets. Then if each shadow SIM was sent in the shadow network at the same time as its corresponding DATA in the real network, it would also experience the same propagation, transmission and queueing delays. We can exploit this shadow network as follows. As source hosts are about to send DATAS, their corresponding SIMS are sent instead in the shadow network and load-balanced across random paths. Some may be dropped at some congested buffer of size B that is already full of B other SIMS. Others will reach their destination as their buffers were less congested, and an acknowledgment will get back to the source host. Then, a fixed time later, we only send the DATAS that correspond to SIMS that arrived, through the exact same path. We are intuitively guaranteed that the DATAS will also arrive: if a DATA is blocked at some buffer by B DATAS, it means that its SIM would also have been blocked by their corresponding B SIMS, which did not happen. As for any DATAS with a blocked SIM, we simply send a new SIM to probe a new random path.

The above concept is appealing, but not practical. The second key idea is to *approximate the shadow* SIM *network by using dedicated buffers* for SIMS and DATAS. Many commodity switches support partitioning their buffers based on different traffic classes. Now SIMS and

DATAs coexist in the same network and share its capacity. Each SIM effectively implements reservation on a given path for a place in the DATA buffers that will be used when sending a DATA some fixed time later. Intuitively, DCSIM trades off a small portion of the network capacity to ensure a practically lossless network, and thus avoid a more significant loss of capacity due to DATA losses.

In evaluations, DCSIM consistently outperforms existing schemes under several collective workloads. It exhibits better CCTs, higher utilization, *no packet drops*, minimal switch queueing, and negligible reordering. It shines even more under adverse conditions, such as a core-switch oversubscription, as it keeps low CCTs and zero drops while other algorithms suffer from high loss rates.

Contributions. In summary, we make the following contributions.

- We introduce the new conceptual framework of simulating paths before sending packets.
- We design DCSIM to follow this framework while being fully deployable on commodity datacenter networks.
- We show that DCSIM achieves low CCTs using a *practically lossless transport*.
- We show that DCSIM maintains high performance under challenging scenarios, including oversubscribed topologies and small switch queues.

The DCSIM source code is available online [53].

2 DCSIM Algorithm

2.1 Design goals

We design DCSIM to achieve the following goals:

1. No loss. DCSIM should have a near-zero loss probability, despite running in a lossy commodity network. With zero data loss, there is no need to retransmit data, and forward progress is guaranteed.

2. Low queueing. In modern datacenter networks, "queueing delays and buffer overflow are the root cause of unpredictability" [3]. DCSIM should offer a low-queueing solution that enables flexibility to changing patterns.

3. Per-packet load-balancing. DCSIM should be able to offer per-packet load-balancing to fully utilize the network capacity and address challenging conditions such as failed or congested links.

4. Several collectives. DCSIM should be able to handle several concurrent collectives with many flows simultaneously fighting for a chunk of network capacity.

5. Commodity switches. DCSIM should not rely on any non-readily available switch feature.

2.2 DCSIM overview

DCSIM relies on a packet transmission approach that operates in two distinct phases for each DATA packet from source S to destination D:

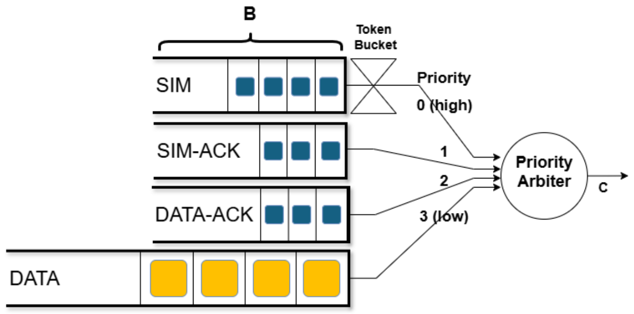


Figure 1 dcSim queueing mechanism

1. A simulation phase, during which a small Sim (simulation) packet is transmitted through a random path from S to D. At each commodity switch along the path, Sims are queued in a Sim queue, distinct from the Data queue for Datas. The Sim queue has strict priority over the Data queue, but its service is rate-limited in a way that allows Sims and Datas to leave in an alternating sequence and fully occupy the line, thus reaching an ideal utilization. Any Sim that arrives at the Sim buffer enters the queue if there is space, and is dropped otherwise. Once a Sim reaches D, D sends back a Sim-Ack to S.

2. A data phase, during which the Datas are sent along the same path previously traversed by their counterpart Sims. To reduce reordering, dcSim associates to each successful Sim the first Data waiting in the queue. This Data is then sent $RTT_{\max}$ time after the Sim was sent, where $RTT_{\max}$ is a fixed datacenter-wide bound on the Sim RTT (cf. § 2.5). Let's focus on a specific Sim_0 and its corresponding Data_0. Assume that Sim_0 competes with other Sims for switch buffer occupancy, and manages to get through. Then when Data_0 later competes with other Datas, we expect it to have no more competitors than Sim_0, and in fact it may have fewer competitors if some Sims got dropped in later switches. We thus expect Data_0 to enter the switch buffers and later reach its destination without any drop. That is, *we expect the Datas not to experience any drop, and in fact to experience slightly less congestion than their corresponding Sims.*

2.3 Switch queueing mechanism

Overview. Fig. 1 illustrates the dcSim queueing mechanism, which is implemented at each switch output and at hosts. It consists of a priority arbiter that implements a strict priority without preemption between four queues, dedicated to the four data types in this paper: (i) Sims, (ii) Sim-Acks, (iii) Data-Acks and (iv) Datas. Sim-Acks and Data-Acks acknowledge reception of Sims and Datas, respectively. Sims (serviced at the highest priority) are rate-limited using a token bucket.

The Sims, Sim-Acks, and Data-Acks are small packets of size ℓ (e.g., $\ell = 64$ bytes), and their queues are stored in small buffers with up to B packets each (e.g., $B = 12$). Datas are larger packets of size L (e.g., $L = 9$ KB for jumbo packets),[1] and their buffer size equals the remainder of the allowed buffer size. Since $\ell \ll L$, the queues for the small packets are extremely unlikely to cause starvation. The packet types can be differentiated in different

[1] Each Sim could also generally represent a set of k Data packets.

ways, e.g., using 2 bits out of the 6-bit ToS field in the IP header. The line rate is assumed to be C throughout the datacenter network.

Intuition with SIMs and DATAs. To set the token-bucket rate, we want to determine the ideal bit rates r_{SIM} for the SIMs and r_{DATA} for the DATAs. We start with a simple case where there is only one switch in the network and we can neglect the SIM and DATA acknowledgments. Assume that the flows are infinite with an infinite stream of SIMs and DATAs, such that the SIM and DATA queues are never empty after the first SIM and DATA appear. Also, let $\alpha = \frac{L}{\ell}$. For example, if $L = 9$ KB and $\ell = 64$ B, then $\alpha \approx 141$. We intuitively want to satisfy two conditions:
(1) *We want to fully utilize the line capacity*, i.e., $r_{\text{SIM}} + r_{\text{DATA}} = C$.
(2) *We also want the two streams of packets to have the same packet rate*, since each successful SIM triggers a later DATA. Formally, $\frac{r_{\text{SIM}}}{\ell} = \frac{r_{\text{DATA}}}{L}$, yielding $r_{\text{DATA}} = \alpha \cdot r_{\text{SIM}}$.

Putting the two conditions together, we get

$$r_{\text{SIM}} = \frac{C}{\alpha + 1}. \tag{1}$$

Thus, by setting a token bucket of rate $\frac{C}{\alpha+1}$ and size B, we expect to achieve these two conditions. This is confirmed in the following theorem (all proofs are in Appendix A).

▶ **Theorem 1** (Ideal SIM and DATA rates). *Under the assumptions above,*
(i) The total rate converges to C.
(ii) The SIMs and DATAs converge to a perfect alternating sequence.

Final scheme. The token bucket above provides a rate of $\frac{C}{\alpha+1}$ to SIM packets, where the α factor accounts for the DATAs and the 1 accounts for the SIMs. However, it neglects the rate of SIM-ACKs and DATA-ACKs. These are harder to account for, as the rates of SIM-ACKs and DATA-ACKs depend on flows that go in the reverse direction. In addition, the DATA-ACK rate is lower as DCSIM only sends back a DATA-ACK every large number of DATAs (e.g., 16), leveraging the lack of packet losses. Since we are interested in AI collectives, we expect a mostly symmetric pattern in which the SIM-ACK rate is close to the SIM rate. Therefore, we heuristically set the token bucket rate at

$$\frac{C}{\alpha_{Data} + 1_{Sim} + 1_{Sim-Ack} + 0.1_{Data-Ack}} = \frac{C}{\alpha + 2.1}. \tag{2}$$

Evaluations show that performance is not sensitive to small variations of this heuristic factor.

2.4 DCSIM description

This section details the various stages of DCSIM, while using the example of Fig. 2 to illustrate each stage.

① **Sending SIM packets.** At each source host S and for each flow, let n_{SIM} denote the number of outstanding SIM and SIM-ACK packets in the network, i.e., the number of sent SIM packets for which S has neither received a SIM-ACK nor timed out. Also, let n_{DATA} denote the number of DATA packets waiting in the host to be sent. Then S always maintains the inequality

$$n_{\text{SIM}} \leq n_{\text{DATA}}, \tag{3}$$

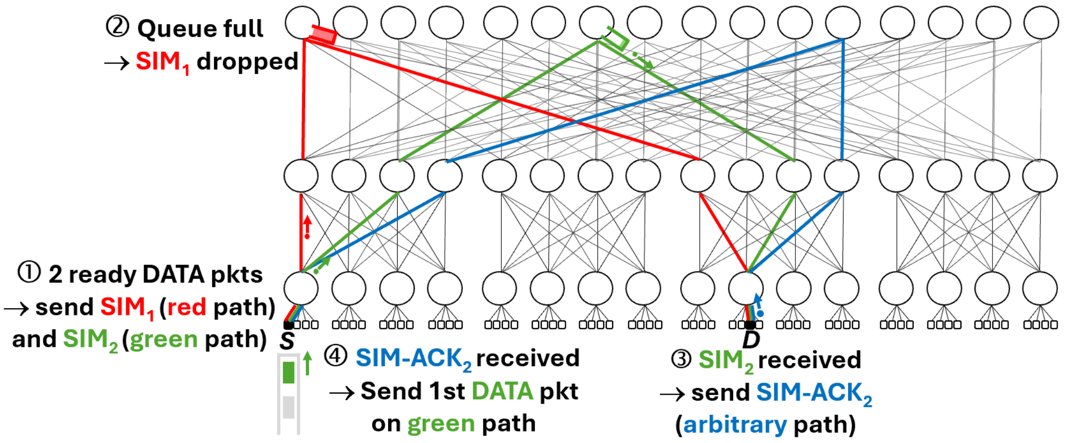


Figure 2 Example of dcSim operation.

i.e., it makes sure that there are no more outstanding Sims (and Sim-Acks) in the network than available Datas waiting to be sent. As previously explained in § 2.3 and illustrated in Fig. 1, S maintains a Sim buffer of size B that is serviced using a token-bucket policy. S then adds Sims to the Sim queue whenever (i) there is space in the Sim buffer, i.e., the Sim queue size is below B and (ii) there are enough Datas to send a Sim (Equation (3)).

We have a degree of freedom in selecting the flows from which to first add Sims. We select flows in round-robin for fairness, but could also have chosen to prioritize flows with the smallest remaining size [5]. Specifically, if there are several collectives and S knows about the collective ID of each of its flows (i.e., roughly speaking, it knows what sets of flows started together), it schedules the next Sim by checking the next collective ID in round-robin order, then choosing a Sim for the next available flow within this collective in round-robin order. If not, it simply picks flows in round-robin order.

Each new Sim is allocated a random path to the destination D. Since the datacenter network relies on commodity switches that use ECMP routing, this random-path allocation is widely implemented by allocating a unique random source port to each Sim [23, 9, 25]. dcSim checks that this source port is distinct from that of currently used Sims. The source port changes the flow five-tuple of the Sim, and therefore it alters its ECMP hashed value at each switch, ultimately changing its path.

Upon transmitting a Sim on the line, S (i) records the Sim transmission time, which will also be used to send a Data after a fixed delay; (ii) starts a timeout mechanism; and (iii) increments n_{SIM}. S also increments n_{DATA} when (i) receiving a new Data from the operating system or (ii) a sent Data times out without a received Data-Ack.

Example. In Fig. 2, assume that at time t at source S, two Data packets arrive at the queue for destination D, so $n_{\text{DATA}} = 2$. S immediately sends two Sim packets: Sim$_1$ with a random source port that leads to the red path after ECMP hashing, then Sim$_2$ with another random source port that leads to the green path. Thus $n_{\text{SIM}} = n_{\text{DATA}} = 2$.

② Switching Sim packets. At each switch, Sim packets go through the buffer mechanism described in § 2.3. If they encounter a full Sim buffer due to simulated congestion, they are dropped. Else, they reach the destination D.

Example. In Fig. 2, the SIM buffer of the leftmost core switch is full, therefore it drops SIM$_1$. In contrast, SIM$_2$ traverses the uncongested green path.

③ Sending SIM-ACK packets. When D receives a SIM, it immediately sends back a SIM-ACK to S. As usual, the SIM-ACK destination port and IP address are the SIM source port and IP address. Since ECMP hashing is not symmetric, the SIM-ACK can take an arbitrary path after ECMP hashing of its 5-tuple, until it reaches S.

Example. In Fig. 2, D receives SIM$_2$ and sends SIM-ACK$_2$ back to S through the blue path.

④ Sending DATA packets. When a source S receives a SIM-ACK for some SIM, it immediately associates the SIM to the first of the DATA packets waiting in S to be sent. S updates

$$\begin{cases} \text{srcPort}_{\text{DATA}} = \text{srcPort}_{\text{SIM}} \\ t_{\text{DATA}} = t_{\text{SIM}} + RTT_{\max} \end{cases} \tag{4}$$

where the first line implies that the DATA will take the same path as its corresponding SIM, and the second line means that the DATA is scheduled to be transmitted after a fixed delay $RTT_{\max}$ (defined in § 2.5) following the transmission time of its SIM. S also decrements n_{SIM} and n_{DATA}. When sending the DATA, S also sets a large timer and keeps the DATA in a side buffer, so that in the rare event that the DATA times out without an acknowledgment, it will be inserted back at the head of the queue of DATA packets waiting to be sent.

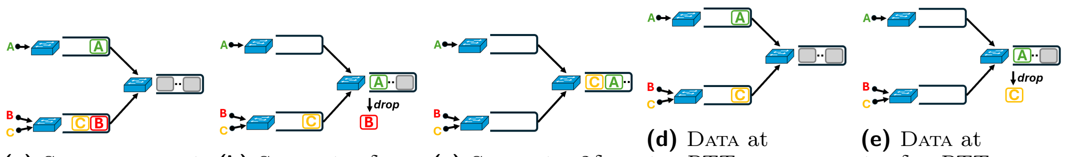
Example. In Fig. 2, S receives SIM-ACK$_2$. Thus, it associates SIM$_2$ with DATA$_1$, the first DATA in the queue. If SIM$_2$ was sent at time t, then DATA$_1$ is later sent at time $t + RTT_{\max}$ along the same green path. In addition, as detailed later, SIM$_1$ times out and a new SIM with a new random path can be sent instead.

Note that this example illustrates well why the association between DATA and SIM packets is not made at the creation of the SIM packets, as would be intuitive. If DATA$_1$ were associated to SIM$_1$, then after SIM$_1$ is dropped, it would need to wait for a timeout and then generate a new SIM, while the following DATA$_2$ packet in the queue would be released upon the arrival of SIM-ACK$_2$. This could cause severe reordering in the system.

⑤ Sending DATA-ACK packets. D sends a DATA-ACK back to S every large number of DATAs (e.g., 16), leveraging the non-existent loss rate in the network (with some minimum frequency, e.g., once every 3 propagation RTTs). (This is not illustrated in Fig. 2.)

Timeouts. When S receives a DATA-ACK, it deletes the DATA. Else, as mentioned, if a DATA times out, it is put back in the queue of DATAs waiting to be sent. In addition, if a SIM timer expires, it decrements n_{SIM}, enabling the transmission of a new SIM.

Example. In Fig. 2, when the timeout for SIM$_1$ that was set in ① expires at $t + RTT_{\max}$, S assumes that SIM$_1$ is lost and decrements n_{SIM}. A new SIM with a new random path can then be sent to D.



(a) Sim queues at t **(b)** Sim at $t+\delta$ **(c)** Sim at $t+2\delta$ **(d)** Data at $t+RTT_{\max}$ **(e)** Data at $t+\delta+RTT_{\max}$

Figure 3 Counterexample showing why Datas may be dropped in multi-stage networks. In (a), the right-side switch Sim buffer is full. In (b), after one slot, only one Sim can access it. The Sim for A enters, so the Sim for B is dropped. In (c), after another slot, the Sim for C enters as well. $RTT_{\max}$ later, in (d), A and C send the corresponding Data packets. But one slot later, in (e), again only one Data can access the limited Data queue. No matter which one, the other Data gets dropped even though its Sim went through.

Data management. To summarize, S needs to manage four Data queues, as Datas can be in four states: (1) at first, waiting for a Sim to be sent; then (2) after receiving a Sim-Ack, waiting for a green light to transmit $RTT_{\max}$ after the Sim; then (3) queued in the Data queue and ready to be transmitted, potentially waiting for other Datas or Sims currently being transmitted; as well as (4) with a copy stored in a side buffer in case a Data-Ack does not come back on time and the timeout expires.

2.5 dcSim computation of $RTT_{\max}$

We want to compute the fixed delay $RTT_{\max}$ between the time S transmits a Sim and the time it transmits its corresponding Data. $RTT_{\max}$ is an upper bound on the Sim RTT, i.e., the time it takes for a Sim then Sim-Ack to get from S to D then back to S. It is a significant parameter, as it delays the transmission of Datas to ensure synchronization, i.e., to make sure that Datas experience the same lack of congestion as their corresponding Sims. To compute $RTT_{\max}$, let RTT_p denote the maximum propagation and processing time in the datacenter network, and let H denote the maximum number of hops for Sims from S to D or for Sim-Acks from D to S. Then we obtain:

▶ **Theorem 2.** *The* Sim *round-trip time is no more than*

$$RTT_{max} = RTT_p + \frac{2H \cdot \ell}{C} \cdot ((B+1) \cdot \alpha + 2.1B + 1) \tag{5}$$

Typically, we would expect this upper bound $RTT_{\max}$ to be within $1-1.5\times$ the propagation RTT for 1.5 KB Data packets, but it can be larger for larger packets.

Example. Assume that $H = 6$ hops in a three-level fat-tree topology, $RTT_p = 7.8$ μs, $C = 800$ Gbps, $L = 1.5$ KB, $\ell = 64$ B, and $B = 12$ pkts. Then $\alpha = \frac{1{,}500}{64} = 23$ and $RTT_{\max} = 1.32\,RTT_p = 10.3$ μs.

3 dcSim Properties

In this section, we present fundamental results about dcSim properties. First, we show that in dcSim, counter-intuitively, Sims may traverse a buffer even though their corresponding Datas will not. This goes against the fact that there are no more Datas than Sims, and therefore the expectation that Datas will experience less congestion. Second, we prove that the token bucket of size B tokens for Sims can be reduced to a size of two tokens only without hurting the property that Sims and Datas can alternate at full rate.

3.1 The limits of emulation

We now want to demonstrate that while in a single switch buffer, SIMs and DATAs can converge to an ideal schedule where SIMs and DATAs alternate without drops and the link becomes fully utilized (Theorem 1), this does not hold in general in a datacenter network.

Counterexample. Let's provide intuition for why we cannot generalize results from a single switch to the whole network. As explained in § 2.2, a SIM denoted SIM$_0$ may compete with other SIMs for a place in the SIM queue. However, these other SIMs may be dropped in later switches, completely changing the timing and effectively canceling the effects of reservation. Thus, when the DATA$_0$ corresponding to SIM$_0$ arrives after $RTT_{\max}$, it may not need to compete with other DATAs anymore, because their SIMs were dropped. Hence, it may quickly exit the switch and arrive earlier than SIM$_0$ at the next switch. However, the next switch may currently be congested, leading to DATA$_0$ being dropped.

Fig. 3 illustrates a counterexample, assuming the DATA buffer can only hold B packets like the SIM buffer. It follows a switch buffer slot-by-slot, where each slot of duration δ corresponds to the time between tokens in the SIM token-buffer mechanism (i.e., $\delta = \frac{L+2.1\ell}{C}$, following Equation (2)). It shows that while the SIM for source host C goes through the switch seamlessly, its DATA actually needs to be dropped.

DATA buffer size. In practice, the above counterexample means that we can only expect a practical near-zero loss rate, not a deterministic zero-loss guarantee. For example, in evaluations (§ 4), we found that if the DATA buffer can hold about 2× as many packets as the SIM buffer, then *we could not see a single* DATA *loss*, no matter the traffic pattern and the network oversubscription.

3.2 Token bucket size

Since SIMs have higher priority in the switches, they have precedence over all other traffic classes. However, because of the non-preemption, they will be delayed if a DATA is currently being sent. Still, they can at most be delayed by the time $\frac{L}{C}$ to send a DATA. The following theorem shows that if we want to reach an alternating sequence of SIMs and DATAs at line rate as proved in Theorem 1 (and under the same assumptions), we cannot use a token-bucket size of 1, as it would reduce the SIM rate, while any size above 2 is fine.

▶ **Theorem 3** (Token bucket). *To achieve an alternating sequence of* SIMs *and* DATAs *at line rate, it is necessary and sufficient to have a token-bucket size of at least 2.*

4 DCSIM Evaluation

We evaluate DCSIM through extensive simulations, which reveal the following key results vs. other algorithms:

- **Zero loss.** Throughout the evaluations with a regular DATA buffer size, DCSIM experiences zero loss.
- **Higher Utilization.** DCSIM achieves a 12% increase in utilization with an all-to-all traffic pattern. Moreover, DCSIM maintains a negligible reordering size.
- **Lower CCT.** DCSIM achieves up to 10% lower CCT under a mix of five all-to-all-v collectives. It still outperforms other algorithms while varying the packet sizes, buffer sizes, flow sizes, collective sizes, and number of collectives.

- **Oversubscribed network.** dcSim shines under adverse conditions. In an oversubscribed scenario with only half the core switches, its CCT is 45% that of other algorithms, and it experiences no losses while their loss rates are typically above 10%.

4.1 Setup

Algorithms. We implement dcSim in the dcPIM simulator [55], and evaluate it against dcPIM, pHost and pFabric.[2] The dcSim source code is available online [53].

Topology. We employ a 3-layer fat-tree topology [4] with a switch radix $k = 8$, resulting in a network consisting of 128 end-hosts, 32 edge switches, 32 aggregation switches, and 16 core switches. All links are configured with 800 Gbps bandwidth, as commercially available today [40], and jumbo frames of 9 KB. We use the simulator default settings for the other parameters: each link propagation delay is set to 200 ns, and each switch is configured with a buffer size of 500 KB per port and a processing latency of 450 ns, yielding a zero-load RTT of 7.8μs. For dcSim, we set a constant Sim buffer size of $B = 12$ packets, reflecting our goal of keeping low queueing occupancies. When evaluating small buffer sizes below 250 KB (equivalent to 28 Data packets), we reduce B proportionally to the buffer size, ensuring that the Sim and Data buffer sizes decrease in lockstep.

Oversubscribed topology. We also test the algorithms using an oversubscribed (blocking) topology [54] where half the core switches are removed.

Collectives. We focus on collective communication patterns that are representative of AI training workloads, and add the functionality to the simulator. A *collective* is defined as a set of flows that begin transmission simultaneously. We implement the following collectives:

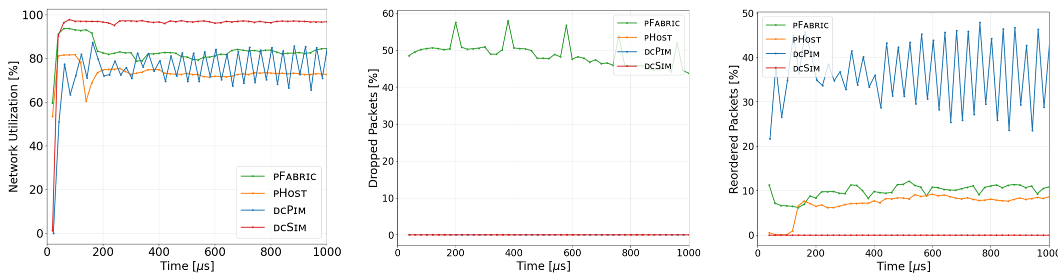
(1) Permutation. The permutation pattern models a *ring all-reduce* collective algorithm. Each sender sends a single flow to a single receiver, and each receiver receives a single flow from a single sender, yielding a total of n flows when there are n hosts in the collective. Each single collective always uses all the hosts ($n = 128$).

(2) All-to-all. The all-to-all pattern models *tensor parallelism.* All hosts in the collective send a flow to all other hosts, yielding $n(n-1)$ flows per collective of size n. We also evaluate an *infinite* all-to-all workload where each flow has an infinite size.

(3) All-to-all-v. The all-to-all variable (all-to-all-v) pattern models *mixture of experts (MoE)* traffic [33, 32, 21, 22, 13, 30, 51]. It is similar to all-to-all, but can be highly unbalanced. To model it, each packet from each sender chooses a random receiver out of the $n-1$ other hosts in the collective.

(4) Set of all-to-all-v. To reflect several competing MoE collectives or tenants, we also focus on a set of several collectives of different sizes that operate in the same datacenter network, as competing collectives are known to be hard to service [49, 32, 26, 58]. We define a baseline set as using all-to-all-v with 5 collectives, with collective sizes randomly selected

[2] dcPIM has been shown [10] to have superior performance vs. Homa [37], Aeolus [19], NDP [17, 48] and HPCC [29] in various settings, therefore we do not repeat comparisons against these algorithms.



(a) Utilization **(b)** Packet drops **(c)** Packet reordering

Figure 4 Single all-to-all collective with infinite flows.

from the set $\{8, 16, 32, 64\}$, possibly with hosts sharing several collectives, and flow sizes of $2\times$ bandwidth-delay product (BDP). We then vary these parameters to study their impact.

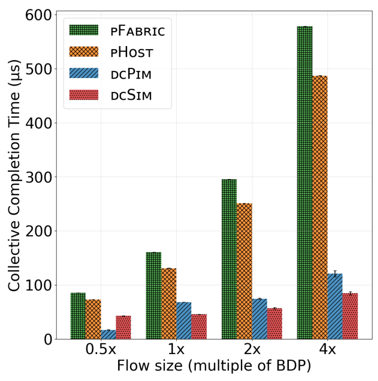
Metrics. We measure the: (1) *Collective completion time (CCT)*, i.e., the time at which the last packet of a collective reaches its destination. When there are several collectives, we average over all CCTs. (2) *Packet loss rate.* (3) *Total queueing delay*, i.e., the total time from the transmission time at the source host to arrival time at the destination host for data packets. (4) *Reordering size*, i.e., the number of data packets per measurement interval that arrive at the destination with non-maximal sequence number. We measure it because in the selective-repeat algorithm and in practical hardware implementations, when there is high reordering, the difference between the sent and received orders of several packets can exceed the transmit window, thus throttling throughput and leading to network underutilization. (5) *Utilization*, i.e., the quantity of data received in a time interval, divided by the total link capacity of the hosts.

We run each simulation 20 times and plot the average result, together with the standard error of the mean (SEM) as error bars.

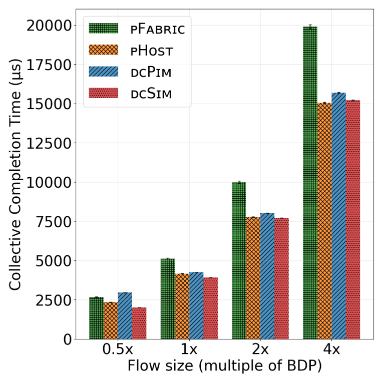
4.2 Performance evaluation

Infinite all-to-all. Fig. 4 illustrates the performance of dcSim compared to dcPIM, pFabric and pHost in an infinite all-to-all traffic pattern. The network utilization of dcSim remains higher than in the other algorithms by at least 14% (Fig. 4(a)). The lower utilization of pFabric may be due to its high level of packet drops. dcSim also impressively achieves 0% loss and 0% packet reordering at all times (Figs. 4(b) and 4(c)). dcSim's low reordering throughout the evaluations may be due to two main reasons. First, its lower queueing delay, as seen in several evaluations. Second, the fact that when sending a burst of Sims through several paths then receiving the Sim-Acks back, Datas are later selected in the order in which Sim-Acks were received, e.g., the first Data is sent on the shortest path. It makes it less likely for a later Data to pull ahead. In contrast, dcPIM and pHost are able to remain lossless, but decrease their sending rate as they detect congestion in the network, based on received token packets.

Single permutation and single all-to-all. Fig. 5 illustrates the CCT with either a single permutation or a single all-to-all collective for different flow sizes. The flow sizes are presented as multiples of BDP, where BDP represents 87 packets. As expected, for all algorithms, CCT increases when the flow size increases. dcSim completes faster than dcPIM, pFabric

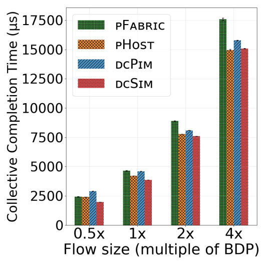


(a) Permutation

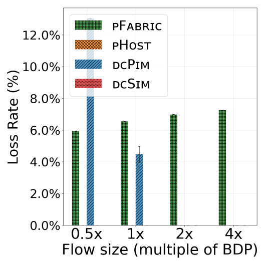


(b) All-to-all

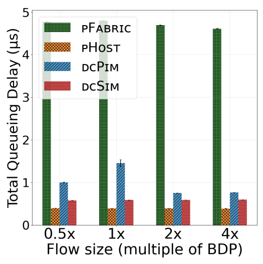
Figure 5 Single permutation or all-to-all collective with different flow sizes.



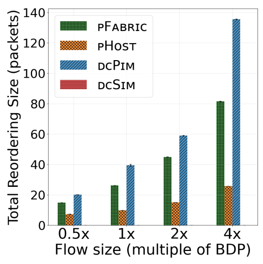
(a) CCT



(b) Loss rate



(c) Total queueing delay



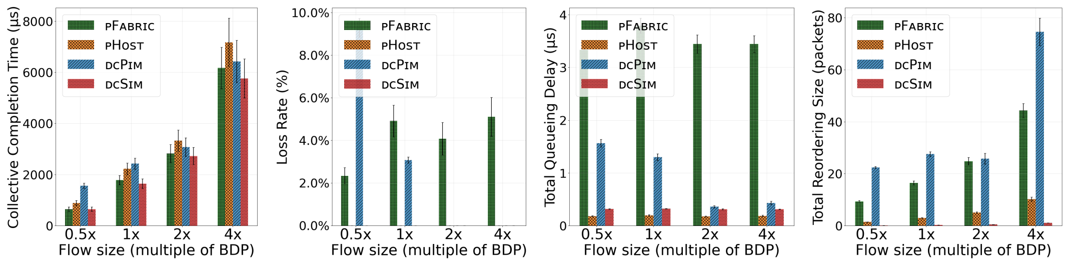
(d) Reordering size

Figure 6 Single all-to-all-v collective with different average flow sizes.

and pHost in all experiments, except for an half-BDP size in the permutation traffic. This evaluation is the least congested of all. Under 1 BDP, dcPIM does not activate its full algorithm for small flows, and therefore simply sends the flows without any control packets. dcSim's focus is on larger flows for AI training, but it could adopt the same behavior for small flows by sending them at a lower priority without using Sim packets. Above 1 BDP, the dcPIM algorithm is activated, worsening the CCT. On the other hand, with the heavier all-to-all traffic (Fig. 5(b)), the non-activation of dcPIM for short flows of size $0.5 \times$ BDP increases dcPIM's CCT even beyond that of pFabric and pHost.

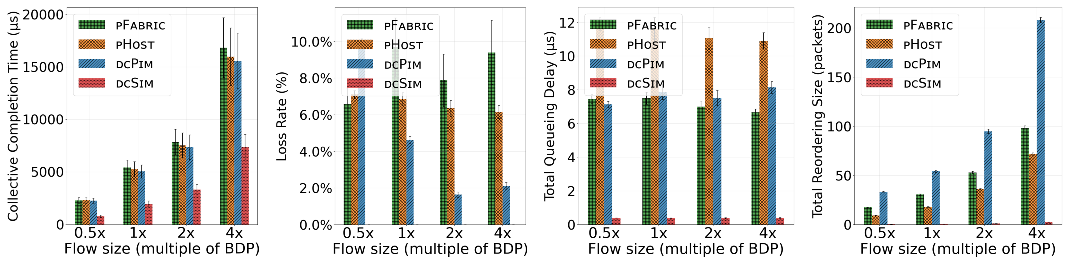
Single all-to-all-v. Fig. 6 presents the results of running a single all-to-all-v collective as a function of different average flow sizes. Since flow sizes vary, we only consider their average, and express it again as a multiple of BDP. dcSim uniformly achieves lower CCTs, with an average improvement of 12.5%, together with a zero loss rate and reordering size, and a small queueing delay.

Several all-to-all-v. Fig. 7 presents an evaluation with five concurrent collectives of random sizes. Performance is largely similar to a single all-to-all-v, with a non-zero yet negligible reordering size. Given the lower total load, dcPIM's queueing delay also gets lower and similar to dcSim. pHost achieves the lowest queueing delay due to its conservative schedule, which comes at the cost of a higher CCT.



(a) CCT **(b)** Loss rate **(c)** Total queueing delay **(d)** Reordering size

Figure 7 Five all-to-all-v collectives with different average flow sizes.



(a) CCT **(b)** Loss rate **(c)** Total queueing delay **(d)** Reordering size

Figure 8 Adverse scenario with an oversubscribed network using half the core switches, given five all-to-all-v collectives.

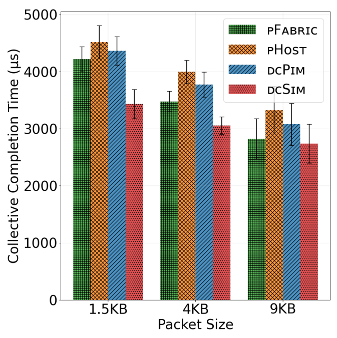
Oversubscribed all-to-all-v. Fig. 8 illustrates an adverse scenario with an oversubscribed network in the baseline scenario of five all-to-all-v collectives. We effectively halve the number of core switches by retaining only the odd-indexed switches and reducing the rate of the remaining one to 1% of their initial rate. dcSim excels in this case, with markedly better performance than the other algorithms. The simulation phase enables dcSim to detect and avoid congested paths by dropping Sim packets early, thereby preventing subsequent Data packet losses. In contrast, dcPIM schedules end-to-end links between two hosts but cannot identify whether one path is better than another, since it has no visibility inside the network. Some aggregation switches are much more congested than others, therefore load-balancing cannot be made blindly. Once congestion is detected, dcPIM reduces the total sending rate to limit losses, but then does not utilize the full network capacity.

4.3 Sensitivity analysis

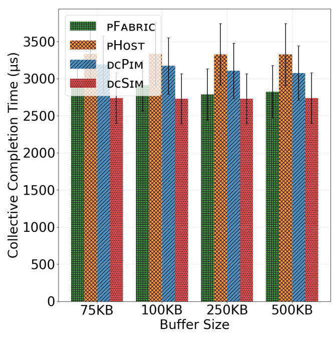
We now perform a sensitivity analysis for the topology parameters by varying a single parameter each time.

Packet size. Fig. 9(a) compares the CCT when using some of the most common Data packet sizes in datacenter networks: 1.5 KB (Ethernet), 4 KB (RDMA), or 9 KB (Jumbo). As expected, for all algorithms, the CCT is lower when L is higher, i.e., larger packets help with bulky transfers. While we used 9 KB as the default packet size in our evaluations, dcSim actually outperforms other algorithms even more for lower packet sizes.

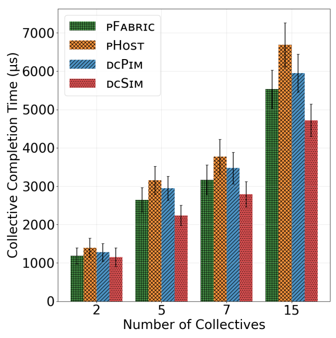
Underbuffering. Fig. 9(b) illustrates the impact of reducing the switch buffer sizes. dcSim is resilient to this reduction, and results are largely similar to previous evaluations.



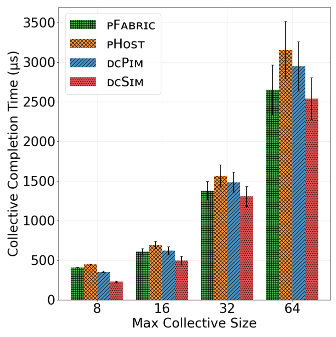
(a) Packet size



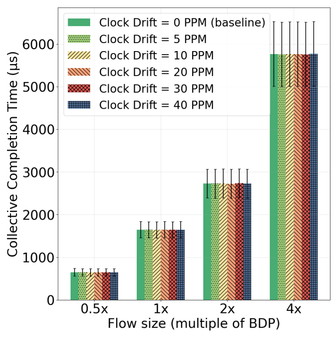
(b) Buffer size



(c) Number of collectives



(d) Max collective size



(e) Clock drift

Figure 9 Sensitivity analysis to topology parameters

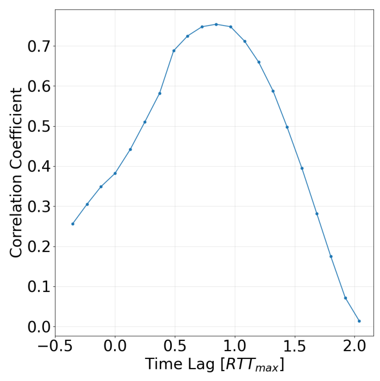
Number of collectives. Fig. 9(c) varies the number of all-to-all-v collectives. dcSim keeps outperforming, and outperforms even more at high loads with many collectives.

Max collective size. Fig. 9(d) illustrates the impact of the maximum collective size, given five random all-to-all-v collectives. dcSim keeps outperforming in all cases.

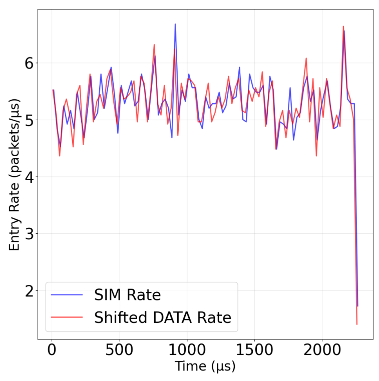
Clock drift. We rely on an $RTT_{\max}$ delay given by an internal clock. We evaluated the resilience of our system to clock drift using a methodology similar to that employed in Firefly [28]. Fig. 9(e) demonstrates that the impact on CCT remains negligible, with a performance degradation of less than 1% even under a static drift of 40 PPM.

4.4 dcSim properties

Correlation of Sim and Data. Fig. 10(a) illustrates the correlation in a random last-hop switch between the rate of Sim packets and the rate of Data packets with shifted times, in order to verify that the rate of Data packets corresponds indeed to the earlier rate of Sim packets. We consider a last-hop (edge switch $\rightarrow$ host) switch queue to reduce the impact of Sims that may be later dropped. The correlation achieves a maximum value of 0.77, confirming that the Data rate indeed reflects the Sim rate, even though the match is not entirely perfect. In addition, the corresponding time lag is 0.86 $RTT_{\max}$. The intuition for a lag lower than $RTT_{\max}$ is that the Sim drops along the path lead to lower queueing delay for the Datas, which reach the last-hop queue faster.



(a) Correlation



(b) Dynamic behavior

Figure 10 SIMs vs. lagged DATAs at the last hop. (a) Correlation between the SIM and time-shifted DATA rates. (b) Dynamic behavior of SIM queue and time-shifted DATA queue.

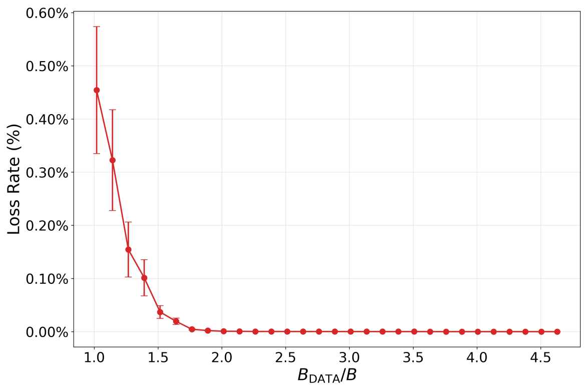


Figure 11 DCSIM loss rate as a function of $\frac{B_{Data}}{B}$.

In addition, Fig. 10(b) shows the dynamic behavior of the SIM and (lagged) DATA queues. They clearly tend to move together consistently, indicating that it is not unreasonable to assume in general that if the SIM buffer is not congested, then neither should the DATA buffer.

DATA buffer. Fig. 11 shows the impact of the DATA buffer size on the loss rate, given a constant SIM buffer size that can hold $B = 12$ SIM packets. Let B_{Data} denote the number of DATAs that can fit in the DATA buffer size. Then the figure shows that for

$$\frac{B_{Data}}{B} \geq 2, \tag{6}$$

there is no loss, i.e., if the DATA buffer can fit 24 DATAs, no DATA will be lost despite the high load of five all-to-all-v collectives. While the lossless threshold is not at an ideal $\frac{B_{Data}}{B} = 1$ that would correspond to an exact emulation, it is still an impressive result that the network can run lossless with so little buffering.

Async DCSIM. DCSIM sends each DATA packet $RTT_{\max}$ after its corresponding SIM was sent. We introduce an *Async* DCSIM version that acts impatiently and immediately sends the DATA packet after a SIM-ACK arrives at the source. Async DCSIM is intriguing, because

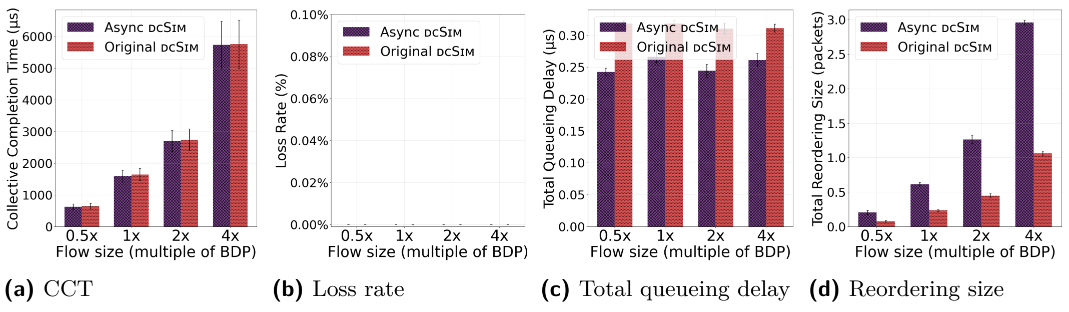


Figure 12 Async dcSim vs. original dcSim.

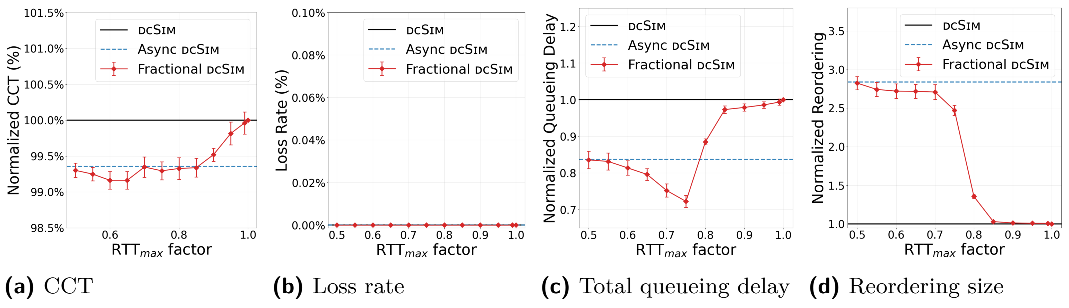


Figure 13 Fractional dcSim vs. baseline dcSim and Async dcSim

on the one hand, it loses the theoretical synchronization, but on the other, it is more reactive to changes in network conditions.

Fig. 12 shows how the CCT of Async dcSim is slightly lower than the CCT of the regular dcSim. Its loss rate is also 0%, and its queueing delay is lower by up to 24%, as it is more likely to quickly exploit a low queue size. However, since it is not synchronized anymore, its number of reordered packets becomes much higher, indicating a higher disparity in the queue sizes between different paths. Thus, this Async version offers different tradeoffs for the datacenter operator.

Fractional dcSim. dcSim waits $RTT_{\max}$ to send Data packets, where $RTT_{\max}$ is computed according to the worst-case formula of Theorem 2. Motivated by the findings in Async dcSim, we analyze the impact on system performance of only waiting for a fraction of the worst-case $RTT_{\max}$. More specifically, we send the Data packet at the later of (1) this fractional delay following the Sim transmission and (2) the Sim-Ack arrival time.

Fig. 13 compares this fractional dcSim for several fractional $RTT_{\max}$ values against the original dcSim and Async dcSim. It shows that the CCT of the fractional dcSim is marginally lower than for the baseline dcSim, while its packet loss remains at 0%. In addition, its queueing delay decreases. However, packet reordering increases, maybe because we are slowly losing the guarantee provided by $RTT_{\max}$. We observe that queueing and reordering behavior remain comparable to the baseline dcSim from $RTT_{\max}$ down to $0.85\,RTT_{\max}$, while the behavior converges towards that of Async dcSim as $RTT_{\max}$ is further reduced.

5 Related Work

Table 1 provides a qualitative comparison of dcSim against existing transport paradigms, as detailed below.

Type	Handles network congestion	Practically lossless	Handles low-rate flows	Uses commodity switches
Load balancing (Oblivious spraying, REPS)	✓	✗	✓	✓
Credits (ExpressPass)	✗	✗	✓	✓
Scheduling (dcPIM, pHost)	✗	✗	✓	✓
Reservation (Harmony)	✓	✓	✗	✗
Simulation probe (dcSIM)	✓	✓	✓	✓

Table 1 Comparison of datacenter transport designs.

Per-packet load-balancing. Many solutions use per-packet load balancing with commodity switches [12, 59, 18, 34, 25, 9, 8, 56, 47, 1, 39, 57, 41, 50]. They rely on diverse reliability mechanisms [16, 38, 52, 8, 42, 56]. However, the RoCE-like recovery schemes can experience poor performance with high per-flow rates, and trimming-based schemes can degrade with many flows [36]. Such load-balancing schemes can also suffer from the interaction with congestion control [15].

Receiver-driven credits. ExpressPass [11] introduces a receiver-driven credit-based scheme that attempts to avoid losses, but it relies on switch modifications, e.g., to ensure symmetric paths, and cannot handle multi-path and link failures. Additional credit-based algorithms, like Homa [37], Aeolus [19] and FlexPass [31], often have little visibility into the network.

Scheduling. pHost [14] shifts scheduling decisions to end hosts using Request-to-Send (RTS) and token-based coordination, avoiding switch modifications. It can be seen as implementing a single stage of matching. While simpler to deploy, its coordination mechanism can incur overhead under bursty or high fan-in patterns. dcPIM [10] replaces $\log(n)$ matching rounds from classical PIM with constant-time matching, achieving high utilization and scalability. However, it does not have visibility within the network.

Priorities. pFabric [5] is a seminal design that prioritizes packets from flows with the smallest remaining size. In contrast, dcSim adopts a round-robin policy.

Non-commodity hardware. Additional schemes show potential for avoiding losses, but require non-commodity hardware. HPCC [29] uses in-network telemetry to provide fine-grained, real-time congestion feedback for precise end-host rate control, but relies on programmable switch support and accurate timestamping, which may not be universally available. Rateless erasure coding can mask losses but needs specialized NICs to be implemented at high rates [24, 36]. Harmony [3] relies on per-flow fixed-bandwidth reservations to eliminate congestion-related drops while achieving high utilization, but needs specialized switches to participate in the reservation process, and struggles with low-rate and variable-rate flows that do not match the fixed reservation rates.

Lossless networks. Large lossless networks have been deployed in datacenter networks [6] and constitute an alternative to lossy networks.

6 Conclusion

In the paper we introduced dcSim, a novel transport algorithm that achieves low CCTs and *practically lossless* performance with commodity switches. dcSim relies on a paradigm shift, by simulating the result of going through a path before doing it. In dcSim, each packet first employs a small simulation probe to traverse the network and explore congestion along a candidate path. Only packets whose simulation probes succeed are then transmitted, expecting to succeed as well. Evaluations confirmed that dcSim achieves faster CCTs and higher utilization than existing schemes, with small queues and virtually zero packet loss. Finally, evaluations showed how dcSim remains effective under adverse conditions that are highly challenging and cause many losses in other algorithms.

References

1 D. Abts and J. Kim. *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Synthesis Lectures on Computer Architecture. Springer International Publishing, 2022. URL: `https://books.google.ca/books?id=NYZyEAAAQBAJ`.

2 Sharon Adarlo. Amazon Is Building a Gigantic Computing Facility to Match the Human Brain. `https://futurism.com/the-byte/amazon-anthropic-ai-data-center`, 2025.

3 Saksham Agarwal, Qizhe Cai, Rachit Agarwal, David Shmoys, and Amin Vahdat. Harmony: A congestion-free datacenter architecture. pages 329–343, 2024.

4 Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. pages 63–74, 2008. `doi:10.1145/1402958.1402967`.

5 Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal Near-Optimal Datacenter Transport. In *SIGCOMM*, 2013.

6 Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. Empowering Azure Storage with RDMA. 2023.

7 Matthias Bastian. OpenAI's Stargate pivot highlights rift with Microsoft over future AI computing needs. `https://the-decoder.com/openais-stargate-pivot-highlights-rift-with-microsoft-over-future-ai-computing-needs/`, 2025.

8 Tommaso Bonato, Abdul Kabbani, Daniele De Sensi, Rong Pan, Yanfang Le, Costin Raiciu, Mark Handley, Timo Schneider, Nils Blach, Ahmad Ghalayini, et al. FASTFLOW: Flexible Adaptive Congestion Control for High-Performance Datacenters. *arXiv preprint arXiv:2404.01630*, 2024.

9 Tommaso Bonato, Abdul Kabbani, Ahmad Ghalayini, Michael Papamichael, Mohammad Dohadwala, Lukas Gianinazzi, Mikhail Khalilov, Elias Achermann, Daniele De Sensi, and Torsten Hoefler. REPS: Recycled entropy packet spraying for adaptive load balancing and failure mitigation. In *EuroSys*, 2026.

10 Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. dcPIM: Near-optimal proactive datacenter transport. pages 53–65, 2022.

11 Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. pages 239–252, 2017.

12 Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. pages 2130–2138, 2013.

13 Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. RDMA over Ethernet for distributed training at Meta scale. pages 57–70, 2024.

14 Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: Distributed near-optimal datacenter transport over commodity network

fabric. pages 1–12. ACM, 2015. URL: `https://dl.acm.org/doi/10.1145/2716281.2836086`, `doi:10.1145/2716281.2836086`.

15 Barak Gerstein, Mark Silberstein, and Isaac Keslassy. Making congestion control robust to per-packet load balancing in datacenters, 2025. arXiv:2509.07907. URL: `https://arxiv.org/abs/2509.07907`.

16 Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over commodity ethernet at scale. pages 202–215, 2016.

17 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. pages 29–42, 2017.

18 Jinbin Hu, Jiawei Huang, Wenjun Lv, Yutao Zhou, Jianxin Wang, and Tian He. CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center. 27(6):2338–2353, 2019.

19 Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. Aeolus: A building block for proactive transport in datacenters. pages 422–434, 2020.

20 Gadi Hutt and Bob Evans. AWS Launches Project Rainier: Massive AI Supercomputing Cluster for Anthropic. `https://podcasts.apple.com/bb/podcast/aws-launches-project-rainier-massive-ai-supercomputing/id1437752008?i=1000717513092`, 2025.

21 Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

22 Zewen Jin, Shengnan Wang, Jiaan Zhu, Hongrui Zhan, Youhui Bai, Lin Zhang, Zhenyu Ming, and Cheng Li. Bigmac: A communication-efficient mixture-of-experts model structure for fast training and inference, 2025. URL: `https://arxiv.org/abs/2502.16927`, `arXiv:2502.16927`.

23 Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. pages 323–335, 2017.

24 Mikhail Khalilov, Siyuan Shen, Marcin Chrapek, Tiancheng Chen, Kenji Nakano, Peter-Jan Gootzen, Salvatore Di Girolamo, Rami Nudelman, Gil Bloch, Sreevatsa Anantharamu, et al. SDR-RDMA: Software-defined reliability architecture for planetary scale RDMA communication. *arXiv preprint arXiv:2505.05366*, 2025.

25 Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabbani, Vipin Jain, Raghava Sivaramu, and Francis Matus. Strack: A reliable multipath transport for AI/ML clusters. *arXiv preprint arXiv:2407.15266*, 2024.

26 Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, 2023.

27 Wenxue Li, Xiangzhou Liu, Yunxuan Zhang, Zihao Wang, Wei Gu, Tao Qian, Gaoxiong Zeng, Shoushou Ren, Xinyang Huang, Zhenghang Ren, et al. Revisiting RDMA reliability for lossy fabrics. pages 85–98, 2025.

28 Yuliang Li et al. Firefly: Scalable, ultra-accurate clock synchronization for datacenters. pages 434–452, 2025.

29 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. pages 44–58, 2019.

30 Xudong Liao, Yijun Sun, Han Tian, Xinchen Wan, Yilun Jin, Zilong Wang, Zhenghang Ren, Xinyang Huang, Wenxue Li, Kin Fai Tse, et al. MixNet: A runtime reconfigurable optical-electrical fabric for distributed mixture-of-experts training. pages 554–574, 2025.

31 Hwijoon Lim, Jaehong Kim, Inho Cho, Keon Jang, Wei Bai, and Dongsu Han. Flexpass: A case for flexible credit-based transport for datacenter networks. In *EuroSys*, pages 606–622, 2023.

32 Wei Liu, Kun Qian, Zhenhua Li, Tianyin Xu, Yunhao Liu, Weicheng Wang, Yun Zhang, Jiakang Li, Shuhong Zhu, Xue Li, et al. SkeletonHunter: Diagnosing and localizing network failures in containerized large model training. pages 527–540, 2025.

33 Xinyi Liu, Yujie Wang, Fangcheng Fu, Xupeng Miao, Shenhan Zhu, Xiaonan Nie, and Bin Cui. NetMoE: Accelerating MoE training through dynamic sample placement. In *ICML*, 2025.

34 Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, et al. Alibaba Stellar: A new generation RDMA network for cloud AI. pages 453–466, 2025.

35 Saf Malik. Zuckerberg: Meta to spend 'hundreds of billions' on AI data centres for superintelligence push . `https://www.capacitymedia.com/article-zuckerberg-meta-ai-data-centres`, 2025.

36 Sarah McClure, Sylvia Ratnasamy, and Scott Shenker. Load balancing for AI training workloads. *arXiv preprint arXiv:2507.21372*, 2025.

37 Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. pages 221–235, 2018.

38 Yang Nie, Zheng Shi, Xinyi Chen, and Liguo Qian. An out-of-order packet processing algorithm of RoCE based on improved SACK. In *IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1402–1408, 2022.

39 NVIDIA. NVIDIA InfiniBand Adaptive Routing Technology Accelerating HPC and AI Applications. `https://www.amax.com/content/files/2023/12/NVIDIA_InfiniBand_Adaptive_Routing_Technology_Insights_Whitepaper.pdf`, 2023.

40 NVIDIA. Connectx-8 supernic datasheet. `https://nvdam.widen.net/s/pxsjzhgw6j/connectx-datasheet-connectx-8-supernic-3231505`, 2024. Datasheet.

41 NVIDIA. NVIDIA Spectrum-X Network Platform Architecture. `https://resources.nvidia.com/en-us-networking-ai/nvidia-spectrum-x`, 2024.

42 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. pages 761–777, 2022.

43 OpenAI. Stargate advances with 4.5 GW partnership with Oracle. `https://openai.com/index/stargate-advances-with-partnership-with-oracle/`, 2025.

44 PA Governors Press Office. Governor Josh Shapiro Announces Amazon Plans to Invest $20 Billion in Pennsylvania for AI Infrastructure. `https://dced.pa.gov/newsroom/governor-josh-shapiro-announces-amazon-plans-to-invest-20-billion-in-pennsylvania-for-ai-infrastructure-in-largest-capital-investment-in-commonwealth-history/`, 2025.

45 Dylan Patel, Daniel Nishball, and Jeremie Eliahou Ontiveros. Multi-Datacenter Training: OpenAIs Ambitious Plan To Beat Googles Infrastructure. `https://semianalysis.com/2024/09/04/multi-datacenter-training-openais/`, 2024.

46 Chenchen Qi, Wenfei Wu, Yongcan Wang, Keqiang He, Yu-Hsiang Kao, Zongying He, Chen-Yu Yen, Zhuo Jiang, Feng Luo, Surendra Anubolu, et al. SGLB: Scalable and robust global load balancing in commodity AI clusters. pages 626–644, 2025.

47 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. pages 691–706, 2024.

48 Costin Raiciu and Gianni Antichi. NDP: Rethinking datacenter networks and stacks two years after. *ACM SIGCOMM Computer Communication Review*, 49(5):112–114, 2019.

49 Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. CASSINI: Network-aware job scheduling in machine learning clusters. pages 1403–1420, 2024.

50 Peter Rizk. Turbocharging Generative AI Workloads with NVIDIA Spectrum-X Networking Platform. `https://developer.nvidia.com/blog/turbocharging-ai-workloads-with-nvidia-spectrum-x-networking-platform/`, 2023.

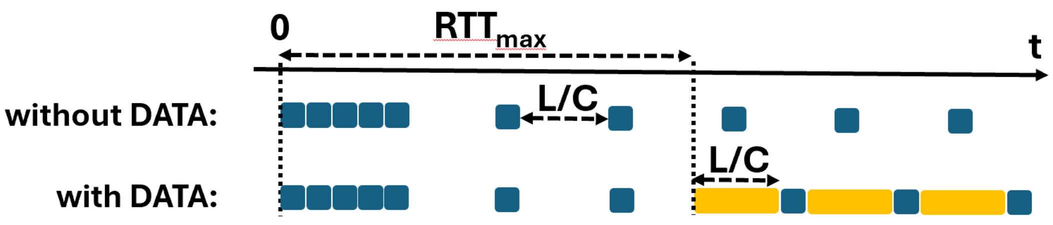


Figure 14 DCSIM token-bucket example

51 Chenchen Shou, Guyue Liu, Hao Nie, Huaiyu Meng, Yu Zhou, Yimin Jiang, Wenqing Lv, Yelong Xu, Yuanwei Lu, Zhang Chen, et al. InfiniteHBD: Building datacenter-scale high-bandwidth domain for LLM with optical circuit switching transceivers. pages 1–23, 2025.

52 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. pages 816–831, 2023.

53 Dan Straussman et al. dcSim repository. `https://github.com/danstr1/dcsim`, 2025.

54 Lide Suo, Yiren Pang, Wenxin Li, Renjie Pei, Keqiu Li, Xiulong Liu, Xin He, Yitao Hu, and Guyue Liu. PPT: A pragmatic transport for datacenters. pages 954–969, 2024.

55 Terabit Ethernet. dcPIM repository. `https://github.com/Terabit-Ethernet/dcPIM`, 2025.

56 Ultra Ethernet Consortium. Ultra EthernetTM Specification v1.0. `https://ultraethernet.org/wp-content/uploads/sites/20/2025/06/UE-Specification-6.11.25.pdf`, 2025.

57 Moshe Voloshin. Introduction to Congestion Control for RoCE. Technical report, Broadcom Inc., 2023. URL: `"https://docs.broadcom.com/doc/NCC-WP1XX"`.

58 Yongji Wu, Yechen Xu, Jingrong Chen, Zhaodong Wang, Ying Zhang, Matthew Lentz, and Danyang Zhuo. MCCS: A service-based approach to collective communication for multi-tenant cloud. pages 679–690, 2024.

59 Jie Zhang, Dafang Zhang, and Kun Huang. Improving datacenter throughput and robustness with Lazy TCP over packet spraying. *Computer Communications*, 62:23–33, 2015.

A Proofs

Proof of Theorem 1. Consider first a case where there are no DATAs, starting from a time $t = 0$ where many SIMs appear. As illustrated in Fig. 14 (without DATA), the token bucket first releases a batch of SIMs until it has no more tokens, then releases the following SIMs periodically as in a simple leaky bucket. These next SIMs are sent every $\frac{\ell}{\frac{C}{\alpha+1}} = \frac{\ell+L}{C}$, allowing for the transmission of an ℓ-sized SIM and then for an interpacket gap equivalent to an L-sized DATA.

Now, assume we also get the first DATA $RTT_{\max}$ after the SIM (bottom row of Fig. 14). The start is the same. However, after $RTT_{\max}$, a DATA arrives at the DATA queue. It will be serviced either immediately (if there is no currently serviced SIM) or just as the SIM departs. Then, when the DATA departs, (a) there is a SIM in the SIM queue by assumption, and (b) since the DATA lasts $\frac{L}{C}$ time, at least $\frac{L+\ell}{C}$ has passed since the last SIM used its token, so there is also a token for the SIM packet. Thus, a SIM packet is serviced. Since we assumed that there are always DATA packets in the queue after the first one, it is followed by a DATA packet. And so on periodically. Since the SIM packet is serviced exactly at the rate of the token bucket, tokens do not accumulate and there is never more than one token available between the end of a DATA packet and the start of the next one.

As a result, SIMs and DATA are serviced in an *alternating sequence*, and they full occupy the line, i.e., *their total rate is C*. ◀

Proof of Theorem 2. The sum of the time it takes for a SIM to get from S to D and the time for its corresponding SIM-ACK to get from D to S, is equal to the sum of the total (1) propagation time, (2) transmission time and (3) queueing time.

1. Propagation time. The total propagation round-trip time is at most RTT_p.

2. Transmission time. The SIM and then the SIM-ACK cross $2H$ hops. The transmission time at each hop is $\frac{\ell}{C}$, yielding a total of $2H \cdot \frac{\ell}{C}$.

3. Queueing time. A SIM entering the SIM buffer encounters at most $B-1$ SIMs in the queue. If it is the first SIM in the queue, it will leave in at most $\frac{\ell\cdot(\alpha+2.1)}{C} + \frac{L}{C}$, where the first term accounts for the maximum time it takes for the token bucket to allow departure, and the second term accounts for the maximum time a non-SIM packet may block the SIM packet (non-preemption), following Equation (2). Since it finds up to $B-1$ SIMs in the queue upon arrival, it will leave in at most $\frac{B\cdot\ell\cdot(\alpha+2.1)}{C} + \frac{L}{C}$. Accounting for $2H$ hops and using $L = \alpha \cdot \ell$, we get an upper bound of $2H \cdot \frac{(B+1)\cdot\alpha\cdot\ell+2.1B\cdot\ell)}{C}$.

Finally, after summing all three terms,

$$RTT_{\max} = RTT_p + \frac{2H\cdot\ell}{C}\cdot((B+1)\cdot\alpha + 2.1B + 1) \tag{7}$$

◀

Proof of Theorem 3. (i) First, let's explain why a token bucket of size 1 does not work. We saw that SIMs can be delayed by at most $T = \frac{L}{C}$ time when a lower-priority DATA is currently being transmitted. When using a bucket of size 1, it may set a worst-case pattern where after each SIM is sent, the next SIM (1) first waits for the token-bucket gap time of $T' = \frac{\ell}{\frac{C}{\alpha+2.1}} = \frac{L+2.1\ell}{C}$ (using $L = \alpha\cdot\ell$); (2) then whenever this next SIM is ready to receive the token, a DATA just starts transmission and delays it for another T. Thus, the time between two SIMs will be up to $T + T'$, yielding only about half the needed line rate.

(ii) If the token bucket size is 2, assume that at least one SIM is in the SIM queue. Then while a first SIM may wait for some DATA transmission to complete and then expects to receive its token, the token for the next SIM can still keep coming. Note that $T' = \frac{L+2.1\ell}{C} > \frac{L+\ell}{C} = T + \frac{\ell}{C}$. Thus, after the first SIM departs, the second one will not have received its token yet in such a case. Therefore, there is no lack of token for the SIM behind it, proving that two tokens are sufficient. In other words, once there are several SIMs in the queue and assuming an infinite stream of SIMs, then after an initial period the SIMs will not differentiate between two tokens and any higher number, e.g., $B \geq 2$.

Note that if the queue is empty, a token bucket of size 2 may lead to a small burst of 2 SIMs. It is still better than a burst of B SIMs. Also, if we had a small buffer of one SIM in front of the SIM queue and before the arbiter, we could have used a token bucket of size 1, or a simple leaky bucket. This is the small cost of relying on a commodity switch. ◀