One to Many: Closing the Bandwidth Gap in AI Datacenters with Scalable Multicast

Sepehr Abdous
Johns Hopkins University
USA

Jinqi Lu
Johns Hopkins University
USA

Jiacheng Wan
Johns Hopkins University
USA

Erfan Sharafzadeh
Meta & Johns Hopkins University
USA

Ying Zhang
Meta
USA

Soudeh Ghorbani
Meta & Johns Hopkins University
USA

Abstract

AI training now floods datacenter fabrics with thousands of simultaneous collectives, yet most frameworks still move data the hard way: $O(N)$ unicasts for a group of N processors. Classic multicast could slash those bytes but has long been deemed unscalable: computing an optimal tree in an asymmetric Clos is NP-hard and group-specific rules quickly exhaust switch TCAM.

We contend that both obstacles disappear once we embrace two mundane facts of today's AI deployments. (i) **Layer regularity.** Even after link failures, Clos paths rise and fall cleanly through layers. We propose a novel layer-peeling heuristic that exploits this to build near-optimal Steiner trees in polynomial time. (ii) **Job locality.** GPU schedulers bin-pack tasks into a few racks, enabling us to design a *power-of-two prefix* method through existing datacenter switch operations. Pre-installing the $k-1$ prefixes per pod, which *shrinks state from exponential to linear* and adds less than 8 B per packet. In a 64-ary fat-tree (65,536 hosts) our prototype uses just 63 rules, down from four billion—and performs within 1.4% of the Steiner optimum. We do not claim a finished system; rather, we argue that multicast is finally within reach for trillion-parameter models and invite the community to revisit the assumption that "multicast simply doesn't scale."

CCS Concepts

• **Networks** → **Traffic engineering algorithms**; **Data center networks**.

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/2025/11
https://doi.org/10.1145/3772356.3772425

Keywords

Scalable multicast, AI clusters, Bandwidth-efficient collective operations

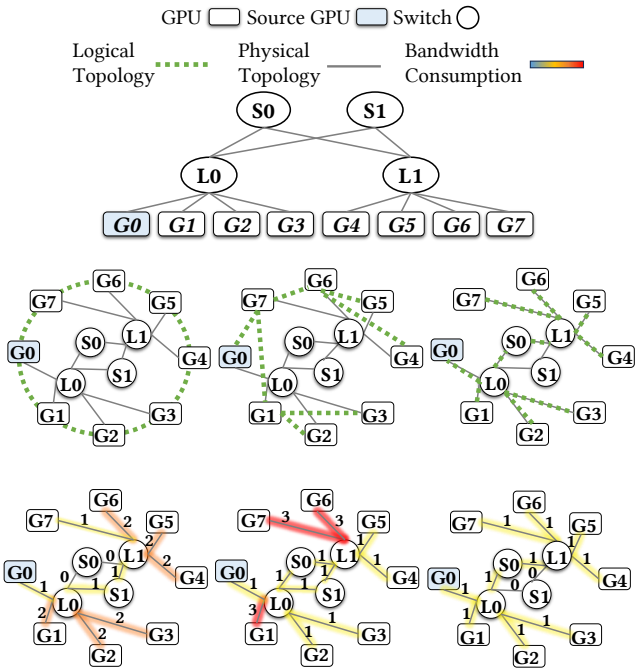
ACM Reference Format:
Sepehr Abdous, Jinqi Lu, Jiacheng Wan, Erfan Sharafzadeh, Ying Zhang, and Soudeh Ghorbani. 2025. One to Many: Closing the Bandwidth Gap in AI Datacenters with Scalable Multicast. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3772356.3772425

1 Introduction

The great AI growth spurt has a networking Achilles' heel. While GPU performance has vaulted *60,000*× in two decades, network links inside the same datacenters have improved barely *30*× [15]. The resulting "bandwidth gap" regularly starves multi-billion-parameter models that stretch across thousands of accelerators. Datacenter networks increasingly emerge as bottlenecks in AI datacenters [6, 33]. Yet, counter-intuitively, production AI frameworks still rely on *application-level unicast* [23]: every GPU individually sprays its updates to every peer. Popular logical topologies—rings, double binary trees, or pipelined meshes—only *schedule* these unicasts; they do not reduce the total traffic. Figure 1 shows a simple Broadcast collective where unicast rings and trees overshoot the minimum bandwidth by 70–80%.

Why not just multicast? Classic IP multicast [10] would slash bytes, latency, and congestion, but two long-standing scalability barriers keep it out of AI clusters:

(1) *Tree construction at cloud scale.* Computing a minimal multicast tree in a Clos with failures—an *asymmetric Clos*—is an NP-hard Steiner Tree problem [25].
(2) *State explosion.* Thousands of concurrent training jobs can spawn thousands of multicast groups, quickly overflowing switch TCAMs if each group needs its own forwarding entries [11, 12, 18]. Past fixes shrink state only at the cost of jumbo headers, per multicast group churn, or



(a) Ring (b) Tree (c) Optimal

Figure 1: Unicast-based Broadcast in a two-tier leaf-spine AI cluster traverses the same core links up to 80% more often than a multicast-optimal solution. Logical rings and trees tame bursts and incast but do not curb total bytes.

added latency—none palatable for microsecond-sensitive RDMA fabrics.[1]

Our thesis. We argue that both barriers crack open once we leverage two unique facts about AI datacenters:

(i) Clos regularity with bounded asymmetry. Even after random link failures or DoR (Disable-on-Repair) maintenance, most paths rise and fall through *layers*. This structure lets us recursively cover receivers layer-by-layer, yielding a $O(min(\mathcal{F}, |\mathcal{D}|))$-approximate tree[2] in polynomial time.

(ii) Job locality. Collective groups are typically confined to continuous arrays of racks or pods [3], so a compact set of *cover-set prefixes* can label entire subtrees. By pre-installing just $O(k)$ such prefixes per switch in a k-ary fat-tree, we slash data-plane state from $O(2^k)$ to $O(k)$ and encode group membership in only $O(\log k)$ header bits—comfortably within the existing RDMA packet budget. This *exponential-to-linear* cut is transformative: in a 64-ary fat-tree with 65,536 nodes, the required entries plummet from over 4×10^9 to fewer than 64, making multicast practical on today's RoCE hardware.

Deploy-once, touch-never multicast. The resulting system, PEEL (Prefix-Encoded Efficient Layering), requires *no per-group switch updates* and computes near-optimal trees in polynomial time. In large-scale simulations (§4) when performing Broadcast collectives with 8 MB messages, PEEL:

- outperforms Ring and Tree by up to 5× and 12×, respectively, as we change the percentage of failed links from 1% to 10%.
- uses 23% less aggregate bandwidth than unicast rings.
- fits in a fixed 63 TCAM entries in a 64-ary fat-tree network, adding less than 8 B overhead per packet.

Beyond scalability. A deployable multicast service must also provide loss recovery, flow isolation, and rich telemetry. Our prototype inherits RDMA's selective repeat retransmissions and introduces a lightweight heuristic for merging per-receiver congestion signals (§4). All other congestion-control and monitoring hooks reuse the mechanisms already exercised by today's unicast collectives, and are orthogonal to this paper's focus on tree construction and switch state. We therefore zero in on the open scalability gap and reserve a full treatment of reliability engineering for future work.

Take-away. A fresh look at tree algorithms and header/state co-design turns multicast from a 1990s curiosity into a **first-class primitive for tomorrow's trillion-parameter models**.

2 Multicast Tree in Clos

AI clusters typically run on multi-tier Clos fabrics [14, 23]. Because these topologies come in two very different flavors—*symmetric* (no failures) and *asymmetric* (one or more failed links/switches)—we treat them separately.

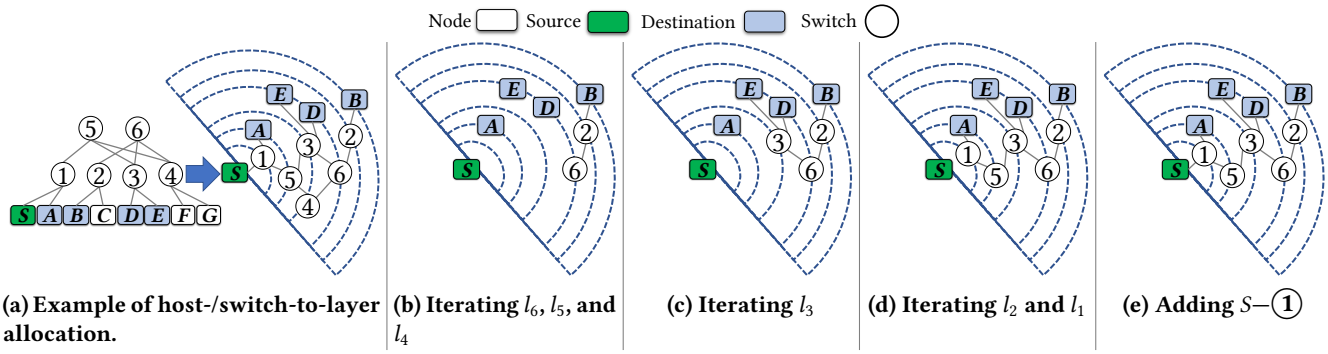
2.1 Symmetric Clos: an Optimal Tree in Polynomial Time

In a failure-free k-ary fat-tree or two-tier leaf–spine, every leaf switch connects to all spine switches with identical link costs. This symmetry collapses the core into a single logical super-node, turning the problem into building a Steiner tree on the host–leaf bipartite graph. Because the graph is now a tree, a breadth-first sweep that starts at the source's leaf and branches down to destination leaves is *optimal* and runs in $O(|\mathcal{D}|)$ time:

Lemma 2.1. *Let $G = (V, E)$ be a symmetric leaf–spine fabric with unit link costs, s the source host, and $\mathcal{D}$ the set of destination hosts. The minimum-cost multicast tree from s to $\mathcal{D}$ can be computed in $O(|\mathcal{D}|)$ time by: (i) lifting all spines into a logical super-node, (ii) adding the unique path $\langle leaf(s),\ super\rangle$, and (iii) attaching each $leaf(d)$, $d \in \mathcal{D}$, to the super-node.*

[1] Other practical hurdles—*e.g.,* loss detection, congestion control, and path observability—already leverage RDMA's selective retransmissions, congestion control, and cluster-wide telemetry used by today's unicast collectives; thus they do not re-inflate switch state or tree-construction cost and are outside this paper's scope.

[2] $\mathcal{D}$ is the set of destination nodes and $\mathcal{F}$ is the length of the longest path among the shortest paths from the source to the destinations in $\mathcal{D}$.



(a) Example of host-/switch-to-layer allocation. (b) Iterating l_6, l_5, and l_4 (c) Iterating l_3 (d) Iterating l_2 and l_1 (e) Adding S—(1)

Figure 2: The layer-peeling greedy algorithm on an *asymmetric* leaf–spine fabric: each panel shows the additional switches and links chosen while progressing from the outermost layer (l_6) toward the source (l_0).

Sketch. The super-node abstraction converts G into a tree; optimal multicast on a tree is found by including every destination leaf and the unique connecting paths, which is linear in $|\mathcal{D}|$. □

In future work, we will extend this approach to deeper multi-stage Clos fabrics (such as three-tier fat-trees) and other multi-layer topologies such as rail-optimized topologies [28] which require additional bookkeeping.

2.2 Asymmetric Clos: NP-Hardness

Real fabrics suffer link and switch failures that shatter the symmetry [7, 31]. With arbitrary link removals, no single core may reach all leaves, and finding the minimum-cost multicast tree reduces to the classical *Steiner Tree Problem*—NP-hard even on degree-bounded graphs [19]. Formally:

Theorem 2.2. *Given a k-ary fat-tree G' with an arbitrary subset of failed links, computing the minimum-cost tree that spans source s and destinations $\mathcal{D}$ is NP-hard.*

Proof sketch. We reduce the classical Set-Cover problem to multicast-tree construction. Given a universe U and a family of sets $\mathcal{S}$, we create a Clos fabric in which every element $u \in U$ is mapped to a distinct leaf switch, and every set $S \in \mathcal{S}$ is mapped to a unique core–to–aggregation path that connects exactly the leaves corresponding to the elements in S. We attach the source host to all such core paths. Any multicast tree must select a subset of these paths so that every destination leaf is reachable; thus the cost of the tree equals the number of selected paths, *i.e.,* the number of chosen sets. Finding the minimum-cost multicast tree therefore solves Set-Cover, which is NP-hard, implying that our problem is NP-hard as well. □

2.3 $O(\min\{\mathcal{F}, |\mathcal{D}|\})$-Approximation

Idea: peel layer-by-layer from the outside in. Imagine concentric *"hop layers"* around the source host: hop layer 0 holds the source, layer 1 all neighbors one hop away, and so on until layer $\mathcal{F}$, the farthest destination (B in Figure 2a). Our key observation is that every destination must eventually attach to some ancestor. By always choosing, on each hop layer, the switch that covers the *most* still-unconnected nodes on the next layer, we greedily mimic the classical set-cover heuristic while preserving a layered, tree-shaped structure.

Walk-through example. Figure 2 shows the algorithm on the asymmetric fabric in Figure 2a (left). We start with a graph $\mathcal{T}$ that contains only the source S and all destinations $\{A, B, D, E\}$.

- *Layer l_6:* add destination B.
- *Layer l_5:* B is still unconnected to l_4, so we pick switch (2) (covers B and maximizes reach) and link it to B.
- *Layer l_4:* add destinations D and E, then choose switch (6) to connect upwards toward (2).
- *Layer l_3:* between (3) and (4), (3) covers more yet-unconnected children, so we pick it.
- Continue inward until layer l_0, finally linking (1) to the source S.

The result (Figure 2e) is a loop-free tree that spans all receivers with just five added switches—only one more than the symmetric optimum.

Algorithm description.

(1) Compute $\mathcal{F} = \max_{d \in \mathcal{D}} \text{dist}(s, d)$.
(2) Build hop layers $l_j = \{v \mid \text{dist}(s, v) = j\}$ for $0 \le j \le \mathcal{F}$.
(3) Initialize $\mathcal{T} = \{s\} \cup \mathcal{D}$.
(4) For $i = \mathcal{F}$ down to 0:
  (a) While some node in $l_{i+1} \cap \mathcal{T}$ lacks a neighbor in $l_i \cap \mathcal{T}$, add to $\mathcal{T}$ the switch in l_i that attaches the most such nodes.
(5) Return $\mathcal{T}$.

Lemma 2.3. *Assuming that $\mathcal{T}$ is the outcome of the greedy algorithm, $|\mathcal{T}| = \sum_{i=1}^{\mathcal{F}} |l_i \cap \mathcal{T}| \le |\mathcal{D}| \times \mathcal{F}$*

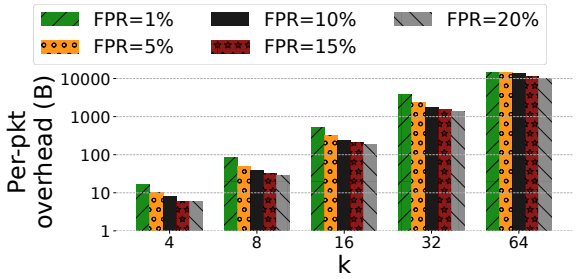


Figure 3: RSBF's Bloom-filter header exceeds one full MTU once $k > 32$; even at a generous false-positive ratio, bandwidth overhead surpasses 100%.

Lemma 2.4. *Assuming that OPT is the optimal multicast tree in the asymmetric Clos,* $|OPT| \geq max(|\mathcal{D}|, \mathcal{F})$.

Theorem 2.5. *The algorithm's approximation factor is* $O(min(\mathcal{F}, |\mathcal{D}|))$.

Proof. Given that $|OPT| \geq max(\mathcal{F}, |\mathcal{D}|)$ and $|\mathcal{T}| \leq |\mathcal{D}| \times \mathcal{F}$, we conclude that $|\mathcal{T}| \leq |OPT| \times (\frac{|\mathcal{D}| \times \mathcal{F}}{max(\mathcal{F}, |\mathcal{D}|)})$. Therefore, the approximation factor is $O(\frac{|\mathcal{D}| \times \mathcal{F}}{max(\mathcal{F}, |\mathcal{D}|)}) = O(min(\mathcal{F}, |\mathcal{D}|))$. □

We also empirically show that our greedy algorithm outperforms Ring and Binary Tree [3] in asymmetric Clos (§4).

Open question: multicast vs. multipath. A single Steiner tree funnels traffic onto one set of links, whereas load balancers' goal is to stripe bytes across many paths. How should a fabric reconcile these opposing objectives, *e.g.,* by building *multiple* near-optimal trees, or by re-hashing prior to branch points—and what performance trade-offs does that create?

3 Keeping Per-Switch State Small

Commodity switches expose only a few thousand multicast entries [12, 18], yet large AI clusters need orders of magnitude more. Existing solutions fail for at least one reason: they overflow TCAM outright [2, 10]; inflate headers and exhaust bandwidth [9, 11, 18, 20, 29]; create redundant traffic [9, 18, 24]; impose multi-millisecond setup delays [12, 23]; rely on unsupported switch operations [26]; or demand full-fabric programmability, driving up cost [12, 18, 20, 24]. We quantify these drawbacks for two state-of-the-art schemes in §3.1. We then introduce PEEL, a power-of-two prefix aggregation scheme that compresses per-switch state from exponential to linear (§3.2). If some tiers are programmable, PEEL optionally performs a two-stage refinement (§3.3): packets launch immediately with static prefixes, and a background controller optimizes the steady state for lowering bandwidth overhead with no start-up latency.

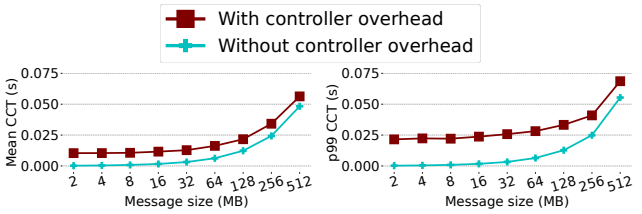


Figure 4: Orca's SDN flow-setup delay inflates collective completion time; the 99th-percentile CCT for a 32 MB Broadcast rises by 8×.

3.1 Why Existing Schemes Fail to Scale

RSBF [18] is a recent Bloom-filter (BF) proposal that pushes multicast state into the packet header: each switch encodes all outgoing ports in a BF, and the header size is enlarged just enough to hit a target false-positive ratio (FPR). In principle, this trades TCAM (switch state) for packet header bits, but in practice, header growth is explosive. Even with an aggressive 20% FPR, RSBF already exceeds one full 1500 B MTU once the fat-tree degree passes $k = 32$ (Figure 3); Elmo [29], LIPSIN [20], and Yeti [11] incur still larger headers [18]. The false positives that remain also spray redundant traffic onto links outside the multicast tree, consuming yet more bandwidth.

At the other extreme, Orca [12], an SDN-based scheme, improves scalability by installing rules only when needed (*i.e.,* when the multicast groups are active) via a centralized controller. It also reduces header size by offloading each ToR's last hop fan-out to a host-side agent and then relies on an SDN controller to push per-tree blacklist rules that mask Bloom-filter false positives. Modelling the controller's flow setup time as $N(10\,\text{ms}, 5\,\text{ms})$ [16, 17] on an 8-ary fat-tree with 1024 GPUs (128 hosts, 8 GPUs / host) in OMNet++ [1], we find that the 99th-percentile collective-completion time for 32 MB Broadcast collectives inflates by 8× (Figure 4). These two case studies illustrate a broader pattern: BF schemes hemorrhage bandwidth as the fabric grows, while controller-driven techniques inject intolerable latency.

3.2 Hierarchical Power-of-Two Cover Sets

Paths in a Clos fabric split naturally into an *upward* funnel and a *downward* fan-out. From the source host up to the spine, only a single packet copy exists; load balancers such as ECMP may choose among equal-cost links, but no switch replicates the packet, so one rule suffices and state never explodes. State blow-up occurs *only* on the way down [12]. Once the packet leaves the highest common ancestor, it may be replicated—first across aggregation switches, then across ToRs, and finally to the destination hosts. Receiver sub-trees can grow exponentially at each tier, so each branch point either needs per-group forwarding entries (IP multicast or

SDN-installed) or must encode multiple outgoing ports in the header (Bloom-filter schemes). Thus, all replication, state growth, and control-plane churn arise in the downward segment, and optimizing this segment is essential for scalable multicast. The remainder of this section therefore focuses on the downward path; although we use the aggregate-to-ToR tier in a fat-tree for concreteness, the same principles apply to other downward segments (*e.g.,* spine-to-leaf in a leaf–spine).

Key idea. PEEL replaces per-group multicast entries with a compact set of *power-of-two prefix* rules, exactly as CIDR [13] coalesces contiguous IP addresses. Concretely, a single rule can forward to *all* ToRs in a pod, two rules can each cover half of the ToRs, four rules a quarter, and so on—every rule's mask length is a power of two. Because AI collectives are bin-packed, *i.e.,* job placements are localized [3], their receivers tend to cluster in a handful of racks within the same pod. We therefore assign every ToR in a pod a $\log_2(k/2)$-bit identifier[3] and pre-install in each aggregate switch exactly one forwarding entry for *every* power-of-two prefix of that identifier space.

Packets carry a small header containing a tuple ⟨prefix value, prefix length⟩ which selects one of the pre-installed power-of-two prefix rules at the aggregate switch. Upon receipt, the switch parses the header, applies the indicated prefix rule, and replicates the packet to all ports in the corresponding block. These operations are already supported by commodity datacenter switch hardware, requiring no new ASIC features.

Packet generation and header overhead. The sender emits *one packet for each selected prefix*. Each packet carries a single ⟨prefix, len⟩ tuple, whose size is

$$\text{header bits} = \underbrace{\log_2(k/2)}_{\text{prefix value}} + \underbrace{\left\lceil \log_2(\log_2(k/2)+1) \right\rceil}_{\text{prefix length}} = O(\log_2 k),$$

which is well under 8 B even for k=128, a fat-tree with 500+K hosts.

Switch-state overhead. Because the prefix space is fixed, we can pre-install every power-of-two rack block once and keep the data plane *fully static*. Let $m = \log_2(k/2)$ be the number of bits in a ToR identifier. For each prefix length $\ell \in \{0, \ldots, m\}$ there are 2^ℓ disjoint blocks (*e.g.,* length 0 covers all ToRs, length m covers a single ToR), so an aggregate switch needs $1 + 2 + 4 + \cdots + 2^m = 2^{m+1} - 1 = k - 1$ TCAM entries—linear in the port count, *e.g.,* only 127 for k=128, a fat-tree with over 500K hosts, versus the $O(2^{k/2})$ blow-up of naïve IP multicast, roughly $2^{64} \approx 1.8 \times 10^{19}$ entries for the same network!

Example. Consider an 8-ary pod whose ToRs are numbered `000`–`111`. A Broadcast collective targets racks `010`, `011`, `100`, `101`, `110`, and `111`. PEEL builds a small trie and selects the outermost complete sub-trees, yielding prefixes `1**` (four ToRs) and `01*` (two ToRs). The source injects *two* packets: one carrying the 3-bit prefix `1**`/1 and one carrying `01*`/2.

Upon arrival, the aggregate switch matches the first packet on its `1**` rule and multicasts to the four upper ToRs; the second packet matches `01*` and reaches the two lower ToRs.

3.3 Optional Two-Stage Refinement with Programmable Cores

The power-of-two overlay eliminates switch-state blow-ups, yet its coarse prefixes may over-cover a pod when resources are fragmented, causing redundant packets that the ToRs later discard. When even a *single* tier—the core layer—offers limited programmability, PEEL can trim this bandwidth waste without sacrificing the zero-latency start-up enjoyed by static prefixes.

Fast start. As before, the source immediately launches one packet per cover prefix; the core forwards these untouched, guaranteeing that training begins within microseconds.

Background optimization. In parallel, a centralized controller (such as an SDN controller) computes the exact set-cover tree for the active collectives (*e.g.,* using the algorithms in §2). Once that computation finishes, it programs only the core switches with a small number of per-group replication rules—typically one rule per destination pod. From that moment on, the source transmits a *single* copy of each packet through the core; the programmable core duplicates the packet on the fly, appending the appropriate ⟨prefix, len⟩ tuple before forwarding it to the correct aggregates.

3.4 Open Questions

While PEEL closes the long-standing scalability gap by bounding both tree state and per-packet overhead, its design also exposes several new research frontiers.

- **Congestion signals.** Can multicast trees remain both scalable and congestion-aware? A promising direction is a *tree-aware controller* or lightweight in-network marking scheme that curbs synchronized queue build-ups *without reintroducing per-group switch state*. The key challenge is detecting congestion early on shared multicast links—before queues synchronize across branches—while keeping the data plane restricted to limited prefix rules. This tension between visibility and statelessness raises broader questions about what minimal feedback (e.g., aggregated ECN marks or probabilistic telemetry) is sufficient for stable multicast flow control at scale.

[3] A k-ary fat-tree has $k/2$ ToRs per pod.

- **Resource fragmentation.** How should prefix aggregation evolve as job placement becomes less compact? Even in bin-packed clusters, GPU placement often leaves small gaps across racks or pods, fragmenting prefix ranges and preventing complete sub-trees in the prefix trie. Minor fragmentation can thus reduce PEEL's aggregation efficiency and increase redundant transmissions. Future designs could explore adaptive prefix packing, workload-aware trie pruning, or hierarchical aggregation that tolerates sparse job layouts while preserving low overhead.
- **Incremental deployment.** If only a subset of switches can be reprogrammed, which tier yields the highest return on investment—cores that replicate, aggregates that rewrite prefixes, or ToRs that filter? What roll-out sequence minimizes transient imbalance or excess bandwidth during transition? Studying the trade-off between deployment cost, bandwidth savings, and stability could guide how operators gradually adopt multicast support in heterogeneous fabrics.

Together, these questions mark the next stage of the journey: *from feasibility to deployability.* Solving them could turn multicast from a long-dismissed curiosity into a core substrate for large-scale distributed AI training.

4 Evaluation

We simulate PEEL in OMNet++ [1] and evaluate its latency under Broadcast collectives. Our key findings are: (a) PEEL performs closely to the optimal Steiner tree baseline, while outperforming Ring, Binary Tree, and Orca. (b) With programmable core switches, PEEL's performance becomes even closer to the optimal case; with 512 MB messages, its mean latency is only 1.4% above the bandwidth-optimal Broadcast.

Experimental setup. We simulate an 8-ary fat-tree with 4 servers per ToR; each server hosts 8 GPUs attached to a dedicated NIC, and GPUs communicate via NVLink/NVSwitch at 900 GBps [5], while all physical links run at 100 Gbps [8, 30]. Traffic consists of Broadcast collectives whose arrivals follow a Poisson process (CPS) [32], each parameterized by its scale (GPU count) and message size; GPU selections honor job locality [3]. We report the mean and 99$^{\text{th}}$-percentile (tail) collective-completion time (CCT)—the interval from collective initiation until the message has reached all GPUs.

Baselines. We evaluate PEEL against two widely deployed collective communication algorithms, *i.e.,* Ring and Binary Tree, and Orca [12], a state-of-the-art scheme that uses centralized controllers to realize scalable in-network multicast. We choose Orca as it does not face scalability challenges under large networks, unlike RSBF [18], and does not require operations unsupported by today's programmable switching fabric, unlike Cepheus [26]. We model the controller's flow setup delay as a normal distribution ($N(10\,\text{ms}, 5\,\text{ms})$) [16, 17]

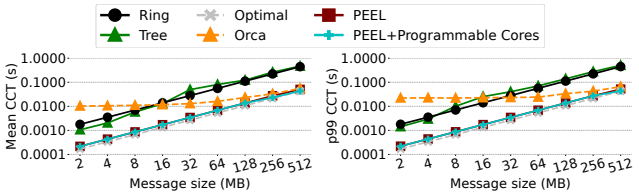


Figure 5: PEEL performs closely to the bandwidth-optimal baseline.

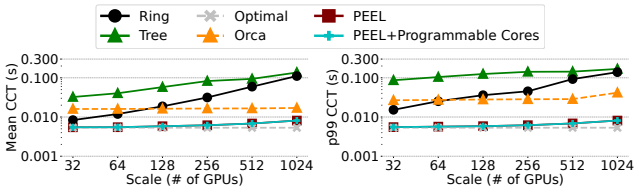


Figure 6: PEEL is faster than Orca, Tree, and Ring.

for both Orca and PEEL. When distributing data among nodes using Ring or Binary Tree topologies—similar to existing libraries such as NCCL [4]—we divide messages into chunks to enable pipelined forwarding. This allows nodes to forward fully received chunks while concurrently receiving the remaining ones, thereby increasing parallelism and overall performance. In our implementation, each message is divided into eight chunks. Lastly, we add the bandwidth-optimal Broadcast mechanism (using an optimal Steiner tree) as a baseline.

Congestion control. All schemes run atop DCQCN+PFC configured as in prior work: 12MB switch buffers, ECN marks between 5kB and 200kB (1% marking probability), and PFC Stop/Resume at 11% free buffer space with a 5-MTU hysteresis [27, 34]. Multicast makes a single ECN mark fan out into many CNPs, so PEEL replaces DCQCN's receiver-side rate limiter with a sender-side guard timer (one reaction every 50μs). This small change slashes 99$^{\text{th}}$-percentile CCT by 12× for a 64-GPU Broadcast with 32 MB messages. Interactions with other congestion signals are deferred to future work.

PEEL performs closely to bandwidth-optimal Broadcast. As the first experiment, we evaluate the CCTs of distinct baselines in performing Broadcast collectives among 512 nodes with various message sizes. We set the collective arrival rates in a way that the average network offered load in every scenario is 30%. The results, presented in Figure 5, illustrate that PEEL outperforms Orca, Ring, and Tree while performing closely to the optimal scheme regardless of the message sizes. For instance, under 2 MB and 512 MB messages, PEEL's tail CCT is 101× and 21% lower than Orca, respectively. Meanwhile, under 2 MB and 512 MB messages, PEEL's mean CCT is only 23% and 18% higher, respectively, than the bandwidth-optimal Broadcast.

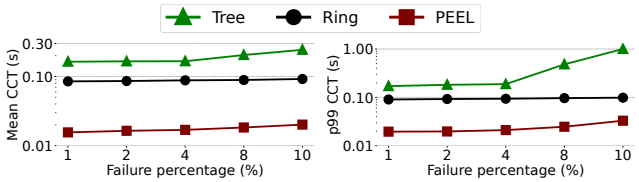


Figure 7: PEEL is fast in asymmetric Clos.

With programmable core switches, PEEL's performs even closer to the optimal scheme. We also simulate PEEL with programmable cores. Figure 5 shows that having programmable cores further reduces PEEL's CCTs when message sizes are large and completion times are larger than controller setup time. For instance, with 512 MB messages, the tail latency of PEEL+Programmable Cores is 14% better than PEEL and only 1.4% higher than optimal Broadcast.

Scale independence. With a fixed 64 MB message, we vary the Broadcast scale from 32 to 1024 GPUs as shown in Figure 6. Across the entire range, PEEL surpasses Ring, Tree, and Orca while remaining closest to the optimal baseline. At 256 GPUs, for example, PEEL's mean CCT is 5× lower than Ring, 13× lower than Tree, and 2.5× lower than Orca.

Robustness to failures. To gauge our greedy tree in an asymmetric fabric, we simulate a two-tier leaf–spine with 16 spines, 48 leafs, two servers per leaf, and eight GPUs per server; every NIC and link runs at 100 Gbps [14]. A 64-GPU Broadcast of 8 MB messages is repeated while 1–10% of spine–to-leaf links are randomly failed. As Figure 7 shows, Ring's latency grows more gently than Tree's because it spreads load, yet PEEL remains faster than both across the board. Even with a high 10% failure rate, PEEL's 99^{th}-percentile CCT is 3× lower than Ring and 30× lower than Tree, confirming that the non-optimal trees produced by our greedy algorithm still deliver superior performance under realistic fault conditions.

5 Related work

The prior work on realizing scalable multicast in modern networks can be broadly categorized into three groups:

1) IP multicast: While IP multicast [2, 10] has been traditionally used to facilitate message multicasting, it is impractical in today's datacenters [2, 12, 18, 26]. Specifically, IP multicast requires significant state-keeping at the switches and continuous control messages exchanged between them, which can delay the process of a receiver joining a multicast session by up to 23 seconds [2, 12].

2) Bloom Filter-based approaches: To shrink the per-switch state, a group of proposals [12, 18, 20, 24], move the multicast overhead to packet headers. Explicitly encoding the forwarding information of every switch in the multicast tree into packet headers significantly increases packet header size [12, 18]. To avoid this, a group of papers [9, 18, 20, 24] use Bloom Filters (BFs) to shrink the packet header space overhead of encoding multicast information. Unfortunately, BF is by nature prone to false positives, resulting in redundant traffic transmission on links that are not part of the multicast tree [12]. Also, existing BF-based approaches [9, 18, 20, 24] still create significant per-packet overhead and do not scale to large networks, *e.g.,* 64-ary fat-tree datacenters [12, 26].

3) Controller-based schemes: Another group of proposals [12, 23] rely on central controllers for updating routing tables when performing multicast. Unfortunately, schemes that exploit SDN-based centralized controllers [12, 14] experience millisecond-scale flow setup delays [16, 17, 21, 22], *e.g.,* flow setup delays can be as high as $50ms$ depending on traffic characteristics [16, 17]. To avoid this, Cepheus [26] uses *initiation messages*. Carrying the multicast information, *initiation messages* traverse the switches in the multicast tree and update their routing tables. However, operations such as updating the routing tables upon receiving initiation messages are not widely deployable in today's datacenter switching fabric. Also, existing proposals [11, 12, 18, 24, 26] typically assume programmability in every datacenter switch, making their deployment extremely costly for the operators.

6 Conclusion

We argue that it is time to rethink the long-held view that "multicast doesn't scale." Our position rests on two early results: (i) a layer-peeling heuristic that turns the NP-hard tree problem in an asymmetric Clos into a polynomial computation with bounded approximation, and (ii) a power-of-two prefix method that collapses switch state to k-1 static rules while adding <8 B of header. Preliminary simulations in a fat-tree place these ideas close to the Steiner optimum and ahead of Orca, Ring, and Binary Tree. Yet several questions remain open, including how multicast trees interact with multipath load-balancing, how to achieve congestion isolation and control, and which tiers should be upgraded for programmability. Tackling these issues could open a rich line of inquiry for the community.

Acknowledgments

We would like to thank our shepherd, Sujata Banerjee, the anonymous HotNets reviewers, and Sana Mahmood for their insightful feedback. This work was supported by NSF CNS NeTS Grant No. 2313164, the Intel Fast Forward Award, and a Meta Faculty Research Award. ChatGPT was used solely for language editing and quality control; all ideas and content are original. This research did not use any Meta resources and was conducted at Johns Hopkins University.

References

[1] 2020. *OMNeT++ Simulator. https://omnetpp.org/*.

[2] 2022. *Multicast Group Capacity: Extreme Comes Out on Top. https://bit.ly/2H5sQ1n.*

[3] 2024. *Double Binary Tree Documentation. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/*.

[4] 2024. *NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.*

[5] 2024. *NVLink and NVLink Switch. https://www.nvidia.com/en-us/data-center/nvlink/*.

[6] 2025. *State of AI Infrastructure Report. https://www.flexential.com/resources/report/2025-state-ai-infrastructure.*

[7] Sepehr Abdous, Senapati Diwangkara, and Soudeh Ghorbani. 2024. Tempus: Probabilistic Network Latency Verification. *IEEE Access* (2024).

[8] Sepehr Abdous, Erfan Sharafzadeh, and Soudeh Ghorbani. 2023. Practical Packet Deflection in Datacenters. *PACMNET* (2023).

[9] Zihao Chen, Jiawei Huang, Qile Wang, Jingling Liu, Zhaoyi Li, Shengwen Zhou, and Zhidong He. 2023. MEB: An Efficient and Accurate Multicast using Bloom Filter with Customized Hash Function. In *APNeT*.

[10] Stephen E Deering and David R Cheriton. 1990. Multicast Routing in Datagram Internetworks and Extended LANs. *TOCS* (1990).

[11] Khaled Diab and Mohamed Hefeeda. 2022. Yeti: Stateless and Generalized Multicast Forwarding. In *NSDI*.

[12] Khaled Diab, Parham Yassini, and Mohamed Hefeeda. 2022. Orca: Server-assisted Multicast for Datacenter Networks. In *NSDI*.

[13] Vince Fuller, Tony Li, Jessica Yu, and Kannan Varadhan. 1993. *Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy*. Technical Report.

[14] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *SIGCOMM*.

[15] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. AI and Memory Wall. In *IEEE Micro*.

[16] Keqiang He, Junaid Khalid, Sourav Das, Aaron Gember-Jacobson, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. 2015. Latency in Software Defined Networks: Measurements and Mitigation Techniques. In *SIGMETRICS*.

[17] Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. 2015. Measuring Control Plane Latency in SDN-enabled Switches. In *SOSR*.

[18] Jiawei Huang, Zihao Chen, Yiting Wang, Hui Li, Zhaoyi Li, Qile Wang, Sitan Li, Zhidong He, and Wanchun Jiang. 2024. Achieving High Efficiency for Datacenter Multicast using Skewed Bloom Filter. In *ICPP*.

[19] Frank K Hwang and Dana S Richards. 1992. Steiner Tree Problems. *Networks* (1992).

[20] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. 2009. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. *SIGCOMM Computer Communication Review* (2009).

[21] Murat Karakus and Arjan Durresi. 2017. A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN). *Computer Networks* (2017).

[22] Ramin Khalili, Zoran Despotovic, and Artur Hecker. 2018. Flow Setup Latency in SDN Networks. *JSAC* (2018).

[23] Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler. 2024. Network-Offloaded Bandwidth-Optimal Broadcast and Allgather for Distributed AI. In *SC*.

[24] Dan Li, Henggang Cui, Yan Hu, Yong Xia, and Xin Wang. 2011. Scalable Datacenter Multicast using Multi-class Bloom Filter. In *ICNP*.

[25] Dan Li, Yuanjie Li, Jianping Wu, Sen Su, and Jiangwei Yu. 2011. ESM: Efficient and Scalable Datacenter Multicast Routing. *ToN* (2011).

[26] Wenxue Li, Junyi Zhang, Yufei Liu, Gaoxiong Zeng, Zilong Wang, Chaoliang Zeng, Pengpeng Zhou, Qiaoling Wang, and Kai Chen. 2024. Cepheus: Accelerating Datacenter Applications with High-performance RoCE-capable Multicast. In *HPCA*.

[27] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High Precision Congestion Control. In *SIGCOMM*.

[28] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, and Rui Miao. 2024. Alibaba HPN: A Datacenter Network for Large Language Model Training. In *SIGCOMM*.

[29] Muhammad Shahbaz, Lalith Suresh, Jennifer Rexford, Nick Feamster, Ori Rottenstreich, and Mukesh Hira. 2019. Elmo: Source Routed Multicast for Public Clouds. In *SIGCOMM*.

[30] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: A Network Architecture for Disaggregated Racks. In *NSDI*.

[31] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2020. Probabilistic Verification of Network Configurations. In *SIGCOMM*.

[32] Yongji Wu, Yechen Xu, Jingrong Chen, Zhaodong Wang, Ying Zhang, Matthew Lentz, and Danyang Zhuo. 2024. MCCS: A Service-based Approach to Collective Communication for Multi-Tenant Cloud. In *SIGCOMM*.

[33] Zhen Zhang, Chaokun Chang, Haibin Lin, Yida Wang, Raman Arora, and Xin Jin. 2020. Is Network the Bottleneck of Distributed Training?. In *NetAI*.

[34] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-scale RDMA Deployments. In *SIGCOMM*.