Square: Towards Sharing Internet-Scale GPU Clouds Fairly and Efficiently

Ningxin Su*, Freeman Cheng†, Baochun Li†
*Hong Kong University of Science and Technology (Guangzhou)
†University of Toronto

***Abstract*—The surge in research interests on large language models has led to a significant increase in the demand for GPU computing power, and users increasingly lease GPUs from cloud service providers across multiple datacenters, due to exorbitant costs and limited availability when acquiring high-end GPUs. As multiple users share GPU-powered servers hosted by multiple datacenters in the cloud to train their models, network traffic from their training workloads shares the same inter-datacenter network topology across the Internet, which is often suboptimal with respect to both bandwidth utilization and fairness across the users. In this paper, we design and implement *Square*, a new system framework that allows multiple users to share the inter-datacenter network topology fairly and efficiently. *Square* is theoretically sound, as it is built upon the rigorous yet surprising formulation of a linear optimization problem based on the mathematical theory of *network coding*, and the solution of such a problem constitutes a complete and optimal plan of fairly routing and transmitting traffic in the inter-datacenter network. We have evaluated our implementation of *Square* using an experimental testbed with our own real-world highly performant implementation developed from scratch using the Rust programming language, and show that it has outperformed competing alternatives by a substantial margin.**

I. Introduction

There has been a steady stream of recent interests in the industry on *decentralizing* distributed model training over the Internet, including communication-efficient approaches such as DiLoCo [1], which is designed for low-bandwidth training. As these datacenters from cloud service providers offer GPU compute resources as a shared commodity to all users, each running its own training workload that spans multiple geographical locations, one naturally wonders how such GPU compute resources can be shared fairly and efficiently among these users. If one examines the communication patterns in each training workload, they involve data and pipeline parallelism, which consists of a large number of collective communication operations, or *collectives* in short, such as *all-reduce*. As many users share the same inter-datacenter network, we naturally wish to maximize the utilization of network resources, while at the same time maintaining a basic notion of *fairness*. At a high level, all users should fairly share inter-datacenter bandwidth; but even how such a notion of fairness should be defined in the context of multiple shared datacenters has not been thoroughly explored in the literature.

What constitutes *fairness* when multiple datacenters (or geo-distributed GPU nodes) are shared by multiple users — also called *tenants* — with independent training workloads? Different users may lease vastly different numbers of GPU containers, sharing the inter-datacenter network in a wide variety of different ways. If one user's workload gains access to all the network bandwidth between its sources and destinations in its collectives, while others competing for bandwidth are denied access to any of the bandwidth, it is certainly unfair. But among the extensive literature on fairness in such traffic engineering problems, including max-min fairness and proportional fairness, which notion of fairness should we adopt and enforce, given that different workloads contain its own set of collectives, each of which in turn consists of network flows? In general, how do we utilize the scarce wide-area network bandwidth as fairly and efficiently as possible?

As we attempt to answer these questions, we show several unique challenges involved in choosing the notion of fairness and enforcing it in the context of multi-user workloads. *First*, the definition of *max-min fairness* requires each flow to have an explicit *bandwidth demand*. In our practical case, however, such a *demand* does not exist, as we simply wish to maximize the throughput of each flow in the collectives.*Second*, to maximize resource utilization, we wish to utilize multiple paths for each source-destination pair in our collectives, which include both *unicast* and *multicast* flows. Yet, conventional water-filling algorithms do not support multiple paths [2]; and the most recent state-of-the-art multi-path water-filling algorithms supported unicast flows only [3].

To our best knowledge, in the context of multiple users spanning several datacenters with both unicast and multicast flows in their collectives, there does not exist any related work in the literature that considered fairness across these users, not to mention proposing efficient algorithms to enforce such fairness while maximizing resource utilization.

Intuitively, we should certainly start addressing this challenge by formulating a proper mathematical optimization problem, with the objective of maximizing the overall resource utilization in a network, and the constraints of link capacities. If we assume that such an optimization problem can be solved, then the input to this solution is the collection of network flows from all the users, and the output should be how these flows should be routed through the network, and how they share the bandwidth at all the links. However, formulating such an optimization problem is inherently difficult with the presence of both unicast and multicast flows. As an example of such difficulty, it has been well understood in the literature that the problem of maximizing the total throughput of multiple multicast flows in a network is equivalent to the *steiner tree packing* problem, which is NP-hard.

In this paper, we propose a new formulation of such an optimization problem, which achieves all of our objectives in one shot: maximizing resource utilization, enforcing a basic notion of fairness, as well as accommodating the needs of both unicast and multicast flows in training workloads. The foundation of our optimization formulation lies in the mathematical theory of network coding, which facilitates the notion of *conceptual flows*. The theory of conceptual flows with network coding allows multicast flows to be formulated as multiple unicast flows without competing for bandwidth with one another. As a result, our optimization problem can be solved as a linear programming problem, ensuring that the problem can be solved efficiently with off-the-shelf solvers.

But do we really need to activate network coding in our context of multi-user GPU datacenters? As a highlight of this paper, we argue that the conventional notion of network coding is unnecessary to reap the benefits of conceptual flows. Instead, we take advantage of the fact that the data we are communicating is not simply bits; they are floating-point tensors that are routinely processed in collective communication.

Coming back to the original problem of fairness, our optimization formulation incorporates the definition of fairness directly in the optimization objective: by computing both the paths and the sending rate for each flow, we seek to maximize the minimum throughput achieved across all the collectives, subject to link capacity constraints in the network. This is similar to the generalized notion of max-min fairness: by maximizing the minimum achievable throughput, we attempt to share network bandwidth across the network as fairly as we can, while simultaneously allowing some flows to achieve throughput that is higher than the globally fair share. We believe that such a simple notion of fairness is sufficient to capture its basic requirements, yet allowing wide-area network bandwidth to be maximally utilized.

Our original contributions in this paper are three-fold. *First*, based on the theory of network coding, we formulate a linear optimization problem that maximizes resource utilization of wide-area network bandwidth, yet allowing multiple users to share the same datacenter fairly. *Second*, we show that the conventional notion of network coding is unnecessary to achieve the benefits of conceptual flows, and that the same processing can be performed within the network. *Finally*, we design and implement *Square*, a new system to allow multiple users to share the datacenter fairly and efficiently, using a software-defined networking architecture with our linear program solver implemented in the control plane. Our experimental results show convincing evidence that *Square* outperformed competing alternatives by a substantial margin.

II. Motivation and Related Work

Large-scale training workloads increasingly span geographically distributed datacenters, so flows from multiple users compete for scarce inter-datacenter bandwidth over both direct and multi-hop paths. These workloads use collective communication routinely: ring all-reduce can be viewed as reduce-scatter followed by all-gather, both composed of unicast flows, while broadcast and tree all-reduce include one-to-many multicast flows. Treating every multicast as independent unicast traffic wastes bandwidth, so *Square* optimizes how collectives and their underlying unicast and multicast flows are routed and rate-limited without changing ML frameworks or network infrastructure.

Square is related to three lines of work. Collective libraries and topology optimizers, such as NCCL [4], TACCL [5], Blink [6], and TopoOpt [7], improve collective execution or topology selection, but primarily target datacenters or require changes to the collective library, generated schedule, or physical topology.

Datacenter schedulers and WAN traffic engineering systems [8] optimize explicit demand or transfer models, whereas *Square* targets inter-datacenter ML collectives with unicast and multicast flows and no fixed per-flow demands. Overlay systems for bulk transfer and video conferencing [9] are closest architecturally, but *Square* focuses on multi-tenant training workloads and works below unmodified ML frameworks and collective libraries.

III. Problem Formulation with Conceptual Flows

In this paper, our overarching goal is to mathematically formulate and solve the problem of *maximizing the utilization* of network bandwidth in an arbitrary network topology that spans multiple geographically distributed datacenters, while *fairly* sharing such resources across multiple users. To make progress towards this goal, we start with a detailed and rigorous study how both unicast and multicast flows can be best optimized, as training workloads from multiple users share the network topology.

An overlay network of *Square* nodes. We consider multiple GPU-powered datacenters, each consisting of a large number of physical machines. A user leases a subset of these physical machines across multiple datacenters, each with one or multiple GPUs, to run its distributed training workload. It is routine and cost-effective for multiple users to share these datacenters. In our design, *Square* is to be implemented as a new software-defined overlay network that supports modern machine learning frameworks and their collective communication libraries (*e.g.*, PyTorch over NCCL). As a software layer, *Square* provides its services between the collective communication libraries, such as NCCL, and their underlying transport protocols, such as TCP. A *Square* node runs on each physical machine as the proxy of all its traffic to and from the *Square* overlay network, which spans multiple datacenters.

We can mathematically represent the *Square* overlay network topology as a *bidirected* graph $\mathcal{G} = (V, E)$, where the set V represents all the *Square nodes* (one for each physical machine), and the set E comprises edges connecting these physical machines in the network. In a bidirected graph, each $e \in E$ is associated with a pair of nodes (u, v) from V, and it has an orientation of either (u^+, v^-) or (u^-, v^+), where x^+ means that the edge is directed away from node x, and x^- implies that it is directed towards node x. For each edge e, we use a function $C : E \to R^+$ to represent its bandwidth capacity. It is worth noting that a bidirected graph is the best representation of inter-datacenter overlay networks, where the amount of bandwidth available on (u^+, v^-) and (u^-, v^+) are independent.

An edge in the *Square* overlay network may traverse multiple hops across the Internet, and its bandwidth capacity is subject to linear capacity constraints within the underlying physical topology, which may not be feasible to compute due to the dynamic nature of bandwidth availability over the Internet. Therefore, in this paper, we simply assume that the available bandwidth on each edge in the *Square* overlay can be accurately measured, and does not vary significantly over time.

Optimizing multicast flows with conceptual flows. We begin our journey by considering the fundamental problem of maximizing the total throughput across all the network flows from training workloads in arbitrary network topologies, leaving fairness concerns across multiple users aside (for now).

At first glance, maximizing total throughput in $\mathcal{G}$ resembles the conventional *maximum multi-commodity flow* problem, but that formulation is not sufficiently general for multicast flows: a source can send packets to multiple destinations, and intermediate nodes may replicate packets instead of forcing the source to send one independent copy per destination. Conventionally, multicast throughput corresponds to Steiner tree packing, which is NP-hard [10]; this is unattractive when many users and training workloads share the same inter-datacenter network.

Square instead builds on network coding theory [11], [12]. For a multicast flow, if rate x can be achieved from the source to each destination independently, then rate x can be achieved for the entire multicast flow. Li *et al.* [13] captured this with *conceptual flows*: source-destination unicast flows that do not contend for shared edge capacity when they belong to the same multicast flow and target different destinations. This gives *Square* a tractable linear optimization formulation; later we show that the coding effect can be realized with tensor operations rather than finite-field packet coding.

Maximizing total throughput via linear optimization. Since multicast is a more general category of network flows and unicast is only its special case, to maximize the total throughput of all network flows, it suffices to consider multicast flows only. We allow coding within each multicast flow only; conceptual flows from different multicast flows still compete for the capacities of edges that they share in the network. Built upon this foundation, we formulate the problem of maximizing the total throughput as the following linear optimization problem:

$$
\begin{aligned}
\max \quad & \mathcal{X} \\
\text{s.t.} \quad & \mathcal{X} \le \sum_i f_i && (1)\\
& f_i \le \sum_{p \in \mathcal{P}_j^i} x_j^i(p), j = 1, \ldots, k_i && (2)\\
& \sum_{p \in \mathcal{P}_j^i(e)} x_j^i(p) \le x^i(e), \forall i, j = 1, \ldots, k_i && (3)\\
& \sum_i x^i(e) \le C(e), \forall e \in \mathcal{E} && (4)\\
& x_j^i(p) \ge 0, x^i(e) \ge 0, \mathcal{X} \ge 0, \\
& \forall p \in \mathcal{P}_j^i, \forall i, j = 1, \ldots, k_i, \forall e \in \mathcal{E}. && (5)
\end{aligned}
$$

The objective of this linear program is to maximize the sum of flow rates in all the multicast flows, $\sum_i f_i$, as shown in (1). In each multicast flow i, its flow rate is the minimum of the flow rates that can be achieved from the source S^i to each of its destinations D_j^i. Due to the notion of conceptual flows and the use of network coding, the flow rates to each of its destinations D_j^i can be independently achieved, as they do not compete for edge capacities. This observation leads to (2), where f_i is the minimum of conceptual flow rates to each of the destinations. The conceptual flow rate from S^i to D_j^i is represented by $\sum_{p \in \mathcal{P}_j^i} x_j^i(p)$, where $x_j^i(p)$ represents the conceptual flow rate along an acyclic path p in the set of feasible paths $\mathcal{P}_j^i$. Since the flow rate $x_j^i(p)$ is specified along a particular path p, we do not need to explicitly specify the flow conservation constraint, as in the common practice of formulating multi-commodity flow problems and linear optimization problems with network coding [13].

Constraint (3) focuses on the *effective flow rate*, $x_i(e)$, within each flow i on an edge e. Since conceptual flows destined to different destinations within the same multicast flow do not compete with one another for edge capacity, the effective flow rate $x^i(e)$ should be the maximum rate of all conceptual flows in flow i that traverse edge e, which is computed by $\sum_{p \in \mathcal{P}_j^i(e)} x_j^i(p)$, where $\mathcal{P}_j^i(e)$ represents the set of paths in $\mathcal{P}_j^i$ that uses edge e. Due to intra-flow network coding, different flows *do* compete for edge capacities that they share, implying that the total of all effective flow rates from different flows should not exceed the edge capacity, as shown in constraint (4). The constraints that guarantee all flow rates to be non-negative, (5), round out this formulation.

For each multicast flow i, the optimal solution to this linear program provides the conceptual flow rates $x_j^i(p)$ along all feasible paths for each destination D_j^i. It is worth pointing out that this encapsulates both routing and flow control, as it precisely computes all the paths where flow rates are positive, as well as what these rates are.

IV. Square: Fairness, Efficiency, and Coding

The notion of fairness. With multiple users sharing the same inter-datacenter network of *Square* nodes with their training workloads, we must consider enforcing a certain notion of fairness, rather than simply maximizing the total throughput of all network flows. But the question that we raised as a core challenge in this paper was: How do we choose among the existing definitions of fairness in this context? Such a notion of fairness would represent the perspective of the users who lease GPU resources across multiple datacenters: the inter-datacenter network bandwidth should be divided among users without discrimination.

As a core challenge in network resource allocation, fairness has been extensively studied when allocating bandwidth across multiple unicast flows. Assuming that the bandwidth demand from each flow is known, *max-min fairness* ensures that no flow can receive a higher rate without hurting another flow with an equal or lower rate. Exact max-min programming repeatedly solves $\max\min_i f_i$ over the remaining flows and residual capacities, which can be expensive at scale. Recent

one-shot approximations [3] reduce this cost, but they require explicitly sorted allocations and target unicast flows only.

Most of the existing work on fairness focused on unicast flows only, and multicast flows are rarely considered. Though our optimization formulation so far considered multicast flows using network coding theory and conceptual flows, it has not yet incorporated any notion of fairness. Since *Square* is, first and foremost, a practical software framework designed to support real-world inter-datacenter networks, and given that *demands* for our flows do not exist in practice, we intentionally choose to pursue a more practical and simpler alternative: we only solve our optimization problem over the first k steps, each with the objective of $\max\min_i f_i$, and using residual link capacities and the remaining set of flows from the preceding step as input. The number of steps, k, that we choose to perform in max-min programming is a tunable hyperparameter, as is subject to our budget with respect to computation costs. In other words, rather than a more complex one-shot approximation algorithm [3] that involved sorting allocations with unicast flows only, our k-shot approximation algorithm simply carries out the *first* k steps of lexicographical optimization with max-min programming. Though our k-shot approximation is not optimal, it strikes a balance between maximizing resource utilization and the incurred computational complexity in doing so.

Revising the problem formulation. With the objective of $\max\min_i f_i$ in each step of max-min programming, our preceding formulation becomes the following alternative linear optimization problem:

$$\begin{aligned} \max \quad & \mathcal{X} \\ \text{s.t.} \quad & \mathcal{X} \leq f_i, \forall i \\ & \text{Constraints (2), (3), (4), and (5).} \end{aligned} \tag{6}$$

For example, in a two-flow case, the first step may allocate both flows the same minimum rate; after one minimum-rate flow and its consumed capacities are removed, the second step reoptimizes over the residual network and can assign additional conceptual flows to the remaining flow. The final rate of a flow is the sum of the rates allocated to it across these max-min programming steps.

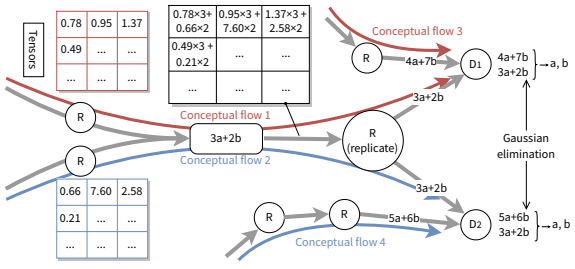


Fig. 1. Merging tensor-backed conceptual flows with a random linear code.

Implementing conceptual flows over tensors. The optimal solution to our linear program contains a number of conceptual flows in each multicast flow, each involving a flow rate and a path from a source to one of its destinations. To implement conceptual flows in practice, we may need to handle the case where packets from multiple flows from the same source to different destinations need to be *merged* to one flow. With conventional random network coding, a linear code with random coefficients needs to be applied on corresponding bytes from multiple flows, in a finite field such as GF(256).

In the context of machine learning training workloads, however, conceptual flows can be realized using *tensor operations* instead: with flows originating from collective communication primitives such as ring all-reduce, all the data traversing *Square* nodes in the inter-datacenter overlay consists of tensors only, which can easily be merged using the same linear code with random coefficients, without resorting to finite field operations. As illustrated in Fig. 1, tensors from two inbound conceptual flows, a and b, can be merged using a random linear code such as $3a + 2b$ before being forwarded to the outbound link. This computation can be efficiently performed on inbound tensors, then decoded further downstream at the destinations using Gaussian elimination.

V. Square: Design and Implementation

From the onset, *Square* has been designed with *practicality* in mind, with its focus on a production-quality real-world implementation. As our linear optimization problems need to be solved in a centralized fashion, our design follows a software-defined networking principle, where all the control-plane decisions are made in a centralized controller, and once a complete transmission plan — involving the paths and rates of all unicast or multicast flows — has been computed, it will be implemented in each of the *Square* nodes in the data plane. Conforming to a software-defined design, statistics about performance metrics in the data plane are periodically measured and reported to the controller. *Square*'s design is rather unique in the context of distributed ML systems. Existing distributed ML frameworks are built primarily for datacenter settings, and their assumptions may fail when the underlying network becomes a wide-area inter-datacenter network over the public Internet. Distributed ML training workloads often experience severe performance degradation in these scenarios due to limited bandwidth that is unevenly distributed across the Internet. Google DeepMind's DiLoCo [1] was motivated by this observation, and mitigated the problem with a communication-efficient optimization algorithm in the training workload.

In contrast, we believe that, for better practicality and generality, *Square*'s data plane must work correctly with any training workload and distributed ML framework. Instead of relying on heavyweight virtual switches or generic user-space tunnels, *Square* operates *purely in the user space* and is implemented in Rust, offering multi-Gbps throughput with direct control-plane integration.

The centralized controller is a JavaScript `node.js` front-end server backed by a *PostgreSQL* database. All linear optimization problems are formulated and solved in Python, and each optimal solution is translated into paths and rates for multicast flows before being stored for retrieval by the data plane. This database-centric API equips *Square* with our optimization-based algorithm while still supporting custom routing and monitoring applications.

Each *Square* node is a user-space Rust overlay switch, operating between collective communication libraries and transport

protocols in the kernel. It exposes virtual network interfaces via TUN/TAP so distributed ML frameworks can run without modifications, and establishes overlay links between *Square* nodes using QUIC.

The data plane uses stream-oriented routing: by assigning one independent path to each stream in a network flow, *Square* supports multi-path routing between a source-destination pair. The controller chooses the routes, while Rust stackless coroutines and ownership passing reduce copying enough to reach several Gbps of achieved throughput purely in user space.

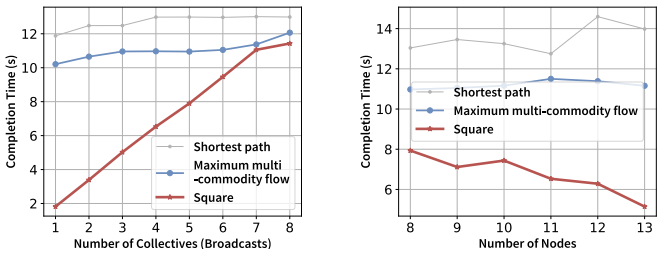
VI. Square: Performance Evaluation

We evaluated *Square* on two testbeds: an emulated wide-area overlay and a real inter-datacenter overlay over Digital Ocean. The emulated overlay was deployed in one datacenter in British Columbia, Canada, with up to 16 *Square* data-plane nodes, each running in its own Docker container across four VMs; each VM had 8 vCPUs and 30 GB of memory. To emulate heterogeneous WAN links, we implemented rate limiting for QUIC overlay links and sampled link capacities from a normal distribution centered at 10 Mbps with a 10% standard deviation. The Digital Ocean testbed used one VM in each of Frankfurt, San Francisco, Sydney, and London, each with 4 vCPUs and 8 GB of memory.

We compared *Square* with two baselines. *Shortest-path routing* merges the shortest paths from a multicast source to its destinations into one tree, and the maximum throughput of the constituent paths determines the multicast rate. This simple baseline captures the core behavior of ECMP-style routing over selected best paths, even though ECMP itself does not directly optimize multicast collectives. *Maximum multi-commodity flow* routing converts multicast traffic into source-destination unicast flows and solves the classic unicast optimization problem. It is optimal for unicast-only demands in arbitrary graphs and is a common WAN traffic-engineering benchmark, but it can over-count bandwidth when a single multicast payload should be shared across destinations. For both maximum multi-commodity flow and *Square*, we use the top $n-1$ shortest paths between each source-destination pair as feasible paths, where n is the number of nodes in the topology.

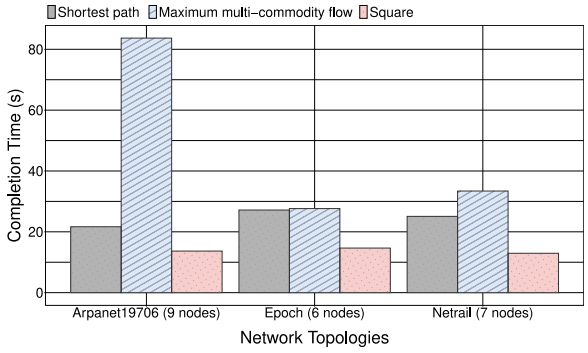
Our experiments primarily focus on full-mesh overlay topologies, which are particularly relevant for inter-datacenter deployments where operators can provision logical links between regions. We measured completion time for broadcast and ring all-reduce collectives, using tensor sizes of 10^8 and 10^9 bits in the emulated and Digital Ocean testbeds, respectively.

Emulated wide-area overlay experiments. As shown in Figs. 2a and 2b, *Square* achieves better resource utilization than both alternatives on full-mesh overlays as the number of collectives and overlay nodes varies. With an increasing number of collectives on an 8-node full-mesh overlay, both shortest-path and maximum multi-commodity flow routing under-utilize bandwidth and suffer prolonged average completion times. In contrast, *Square* balances fairness with utilization and is able to saturate overlay links as the number of collectives scales up.

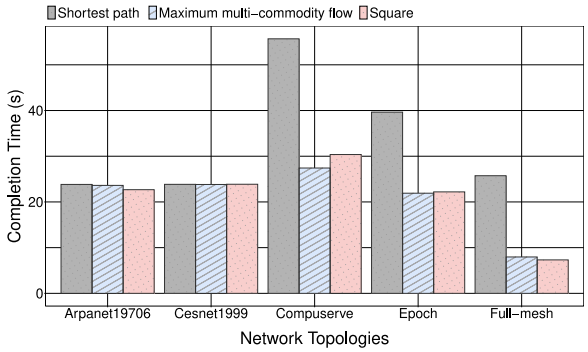


(a) Varying collectives. (b) Varying overlay nodes.

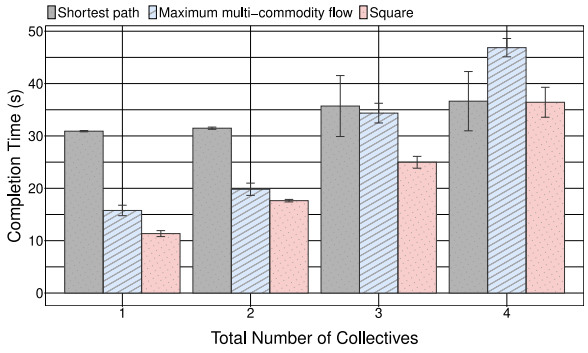
Fig. 2. Full-mesh emulated overlay experiments with broadcast collectives.



(a) Emulated topologies.



(b) Unicast-only collectives.



(c) Digital Ocean overlay.

Fig. 3. Experimental results over emulated real-world topologies, unicast-only collectives, and a real Digital Ocean inter-datacenter overlay.

With 5 concurrent broadcast collectives, increasing the number of nodes in the full-mesh overlay significantly decreases *Square*'s average completion time, while shortest-path and maximum multi-commodity flow routing do not show a gradual decrease despite the additional link capacity. This confirms that *Square*'s computation of optimal routing

and flow rates with multicast flows leads to more effective bandwidth utilization as the overlay scales up.

Beyond full-mesh topologies, Fig. 3a shows results on three Topology Zoo networks: `Arpanet19706`, `Epoch`, and `Netrail`, with two broadcast collectives and the top-2 widest paths as feasible paths. *Square* significantly outperforms its competitors, reducing completion time on `Arpanet19706` from 83.68 s with maximum multi-commodity flow routing to 13.67 s. This result highlights the cost of decomposing multicast traffic into unicast flows in topologies with bottleneck links. Even with only two collectives, *Square* reduces average completion time by nearly 50% compared with shortest-path and maximum multi-commodity flow routing.

The full-mesh results also show that path diversity alone is not sufficient. Maximum multi-commodity flow has access to the same candidate path set as *Square*, but once multicast collectives are expanded into independent unicast demands, the optimizer can reserve capacity for duplicate payloads. *Square* avoids this artifact by preserving multicast dependency before rate allocation, which is why it tracks added overlay capacity more directly as the topology grows. Shortest-path routing is simpler but leaves even more capacity unused because it commits each receiver to a single merged tree before solving the rate-allocation problem.

Across the arbitrary-topology experiments, the magnitude of the gain depends on where bottlenecks lie. The strongest case, `Arpanet19706`, has bottleneck links that are repeatedly selected by decomposed unicast paths, while *Square* spreads the conceptual flow over feasible alternatives. The smaller but consistent gains on `Epoch` and `Netrail` show the same effect when contention is less extreme: *Square* does not need a complete full mesh to improve collective completion time; it only needs enough feasible path diversity to avoid unnecessary duplicate traffic.

Unicast-only ablation. Fig. 3b evaluates five Topology Zoo networks with a ring all-reduce collective, which contains only unicast flows. Due to varying topology structures, both shortest-path and maximum multi-commodity flow routing fluctuate across scenarios. *Square* maintains comparable performance across all cases even when multicast structure is absent, showing that the conceptual-flow formulation remains robust for unicast-only collectives.

Real-world overlay across Digital Ocean datacenters. Fig. 3c reports a 4-node variant of the first experiment with an increasing number of broadcast collectives on the real-world full-mesh topology. We repeated each experiment three times and included standard deviations to account for randomness in real-world overlays. *Square* outperformed both alternatives across the board, demonstrating efficient bandwidth utilization under real inter-datacenter conditions. As the number of collectives increases, both maximum multi-commodity flow routing and *Square* exhibit a linear increase in completion time, indicating relatively high network utilization. In contrast, shortest-path routing shows lower bandwidth utilization for multicast flows when the number of collectives is small. As the number of collectives grows, shortest-path routing improves because the larger full-mesh topology increases the chance of selecting better paths, though it still cannot directly exploit multicast structure.

The Digital Ocean deployment is the closest setting to an inter-datacenter training workload because it combines asymmetric WAN paths, host-side variability, and changing background traffic. The repeated runs preserve the same ordering across trials, showing that the improvement is not an artifact of a single favorable measurement and that the control plane remains effective even when the physical WAN is outside our direct control.

VII. Concluding Remarks

This paper presented *Square*, a software-defined overlay networking layer that improves collective communication for inter-datacenter ML training while balancing fairness and resource utilization. Its optimization framework accommodates multicast flows through conceptual flows from network coding theory, yielding a linear program that can be solved efficiently and deployed through the control plane. Our experiments show that *Square* outperforms shortest-path routing and maximum multi-commodity flow routing because these alternatives cannot directly exploit multicast structure.

References

[1] A. Douillard, Q. Feng, A. A. Rusu, R. Chhaparia, Y. Donchev, A. Kuncoro, M. Ranzato, A. Szlam, and J. Shen, "DiLoCo: Distributed Low-Communication Training of Language Models," 2024.

[2] L. Jose, S. Ibanez, M. Alizadeh, and N. McKeown, "A Distributed Algorithm to Calculate Max-Min Fair Rates Without Per-Flow State," in *Proc. ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, vol. 3, no. 2, 2019, pp. 1–42.

[3] P. Namyar, B. Arzani, S. Kandula, S. Segarra, D. Crankshaw, U. Krishnaswamy, R. Govindan, and H. Raj, "Solving Max-Min Fair Resource Allocations Quickly on Large Graphs," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1937–1958.

[4] Nvidia, "NCCL: Optimized Primitives for Collective Multi-GPU Communication." [Online]. Available: https://github.com/nvidia/nccl

[5] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches," in *Proc. 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2023, pp. 593–612.

[6] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, "Blink: Fast and Generic Collectives for Distributed ML," in *Proc. Machine Learning and Systems (MLSys)*, 2020.

[7] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. Kewitsch, "TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 739–767.

[8] Y. Perry, F. V. Frujeri, C. Hoch, S. Kandula, I. Menache, M. Schapira, and A. Tamar, "DOTE: Rethinking Predictive WAN Traffic Engineering," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 1557–1581.

[9] B. Wu, K. Qian, B. Li, Y. Ma, Q. Zhang, Z. Jiang, J. Zhao, D. Cai, E. Zhai, X. Liu *et al.*, "XRON: A Hybrid Elastic Cloud Overlay Network for Video Conferencing at Planetary Scale," in *Proc. ACM SIGCOMM*, 2023, pp. 696–709.

[10] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: A Low-Delay Multi-Party Conferencing Solution," in *Proc. ACM Multimedia*, 2011, pp. 493–502.

[11] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Info. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[12] Y. Feng, B. Li, and B. Li, "Airlift: Video Conferencing as a Cloud Service using Inter-Datacenter Networks," in *Proc. IEEE International Conference on Network Protocols (ICNP 2012)*, 2012.

[13] Z. Li, B. Li, and L. C. Lau, "On Achieving Maximum Multicast Throughput in Undirected Networks," *IEEE Trans. Info. Theory*, vol. 52, no. 6, pp. 2467–2485, 2006.