Optimizing Wide-Area Collective Communication via Flow-Dependency-Aware Rate Allocation

Yining Wang, Toon Craps, Baochun Li
{*yning.wang, toon.craps*}*@mail.utoronto.ca, bli@ece.toronto.edu*
Department of Electrical and Computer Engineering, University of Toronto

***Abstract*—Large-scale distributed training increasingly spans multiple datacenters due to limited local GPU availability and growing regulatory and data-locality constraints. In these settings, collective communication over wide-area networks (WANs) often becomes a dominant performance bottleneck because WAN links are bandwidth-limited, heterogeneous, and heavily contended. A key but under-modeled property of widely used collectives such as Ring-AllReduce is that their underlying unicast transfers are not independently schedulable: they exhibit finish-to-start dependencies within and across stages, forming sequential chains of flows. Dependency-agnostic bandwidth allocation can therefore waste capacity by allocating bandwidth to flows that are not yet eligible to transmit.**

In this paper, we revisit WAN bandwidth allocation for distributed training through a dependency-aware lens. Assuming communication routes are pre-determined, we formalize the problem of minimizing the average completion time of multiple concurrent collectives subject to link-capacity constraints and flow-dependency constraints. Building on this formulation, we propose a flow-dependency-aware allocation framework that operates at the granularity of flows and dependency chains (flow groups). Within the framework, we design (i) a non-concurrent scheduling strategy that serializes conflicting flows to improve effective utilization, (ii) concurrent rate- and weight-based allocation schemes that are computationally efficient and robust to WAN variability, and (iii) an integrated MultiRing approach that combines inter-collective allocation with intra-collective dependency-aware scheduling. Our extensive array of experiments on Topology Zoo WAN topologies and emulation on the Nextmini testbed show convincing evidence that, explicitly accounting for flow dependencies consistently reduces average collective completion time compared to common baselines (equal-share, data-aware, and barrier-aware) and achieves comparable or better performance than TE-CCL in our evaluated settings, with modest optimization overhead.

I. Introduction

The rapid growth of deep learning models has made efficient distributed training a core requirement of modern machine learning systems [1], [2]. While many production training deployments historically assumed a single, well-provisioned datacenter, this assumption is increasingly fragile: GPU scarcity, cost, and operational constraints, together with regulatory and data-locality requirements, often force training workloads to span multiple geographically distributed sites [3]. In such multi-datacenter settings, communication performance becomes a primary scalability limiter. Compared to within-datacenter interconnects such as InfiniBand [4], inter-datacenter communication runs over wide-area networks (WANs) that are typically bandwidth-limited, heterogeneous, and subject to varying contention. As a result, collective communication primitives can dominate end-to-end iteration time and significantly reduce training throughput.

A large body of prior work improves collective communication by optimizing collective algorithms, execution schedules, and topology selection, often at the library level [5], [6]. Complementary work adopts a networking perspective and formulates scheduling and allocation problems using abstractions such as coflows or traffic engineering (TE), optimizing objectives related to job or stage completion time under capacity constraints [7]–[9]. These approaches have produced substantial gains, but they commonly rely on coarse-grained models of collective traffic: flows within a collective are often treated as interchangeable, concurrently active units, or are aggregated at the collective level. This abstraction can be overly permissive for real collective implementations, particularly in bandwidth-constrained WAN environments where small modeling mismatches translate directly into wasted capacity.

In practice, many widely deployed collectives exhibit explicit execution-order constraints among their constituent transfers. Ring-AllReduce, for example, decomposes gradient aggregation into *ReduceScatter* and *AllGather* stages, each realized as a set of unicast transfers arranged in a pipeline. The pipeline induces *finish-to-start* dependencies: certain transfers cannot begin until prerequisite transfers complete. These dependencies form sequential chains of dependent flows, which we refer to as *flow groups*. At any given moment, only the *ready* (dependency-satisfied) flows can make progress, while flows later in a dependency chain are effectively blocked. A dependency-agnostic allocator that divides link capacity across all flows on a path can therefore allocate bandwidth to blocked flows, reducing the effective rate of the ready flows and extending the end-to-end completion time of the collective. This effect becomes more pronounced in WANs, where bandwidth is scarce and sharing decisions have a first-order impact on completion time.

Motivated by this gap, we revisit bandwidth allocation for distributed training from a *flow-dependency-aware* perspective. We focus on a practical and deployment-friendly setting in which communication routes are already determined (e.g., by an overlay, a multipath routing mechanism, or a policy-driven routing layer), and the remaining problem is to allocate bandwidth and transmission priority across multiple concurrent collectives while respecting both link capacities and flow dependencies. Within this setting, we formulate an optimization objective that minimizes the *average completion*

time across concurrently running collectives under WAN capacity constraints and finish-to-start dependency constraints. The formulation highlights that bandwidth contention primarily occurs among flows that are simultaneously eligible to transmit, i.e., among flow groups whose ready segments overlap on shared links.

Building on this formulation, we develop a set of algorithms that explore the trade-off between completion-time performance and computational overhead. We first consider a non-concurrent strategy that serializes mutually conflicting transmissions to avoid inefficient fractional sharing on constrained links, serving both as a reference point and as a component in more scalable designs. We then design concurrent allocation schemes that compute either per-flow rates or per-group weights; the latter is particularly attractive in WAN settings where available capacity can vary and proportional sharing is easier to implement robustly. Finally, to handle multi-ring or multi-collective scenarios at scale, we introduce an integrated MultiRing approach that combines efficient inter-collective allocation with dependency-aware intra-collective scheduling.

We evaluate our methods using both Python-based simulations and real-world emulation on Nextmini [10], a new Rust-powered network emulation and experimentation testbed. Across these environments, our dependency-aware designs reduce average collective completion time relative to standard baselines such as equal-share and size-/barrier-aware heuristics, and achieve comparable or better performance than TE-CCL [6] in our evaluated configurations, while maintaining modest optimization overhead. These results suggest that explicitly modeling intra-collective flow dependencies is an important and largely orthogonal lever to routing-centric collective optimization in WAN-based distributed training.

The original contributions in this paper are as follows. *First,* we identify and model the finish-to-start dependency structure inherent in practical collectives (e.g., Ring-AllReduce) and introduce a *flow-group* abstraction that captures which flows are mutually constrained from contending for bandwidth at the same time. *Second*, under fixed communication routes, we formulate a dependency-aware bandwidth allocation problem that minimizes the average completion time of multiple concurrent collectives subject to link-capacity and dependency constraints, and we develop both non-concurrent and concurrent algorithms (rate- and weight-based) within a unified framework. *Finally*, through simulations on WAN-like topologies and emulation on the Nextmini testbed [10], we demonstrate that dependency-aware allocation improves completion time and utilization compared to common baselines, and is competitive with TE-CCL [6] in the studied settings.

II. Motivation

Collective communication is a central performance component in distributed machine learning systems: at each training iteration, workers must exchange and aggregate gradients (or model states) efficiently. Among commonly used collectives, *Ring-AllReduce* (RAR) is widely deployed in practice (e.g., via NCCL) because it is simple, bandwidth-efficient in homogeneous settings, and admits a highly structured execution. RAR proceeds in two sequential stages—*ReduceScatter* followed by *AllGather*—and each stage decomposes into a set of unicast transfers. Importantly, these unicast transfers are not freely reorderable: the semantics of the collective induce *finish-to-start* dependencies among flows both within a stage and across stages. Fig. 1 illustrates this structure.

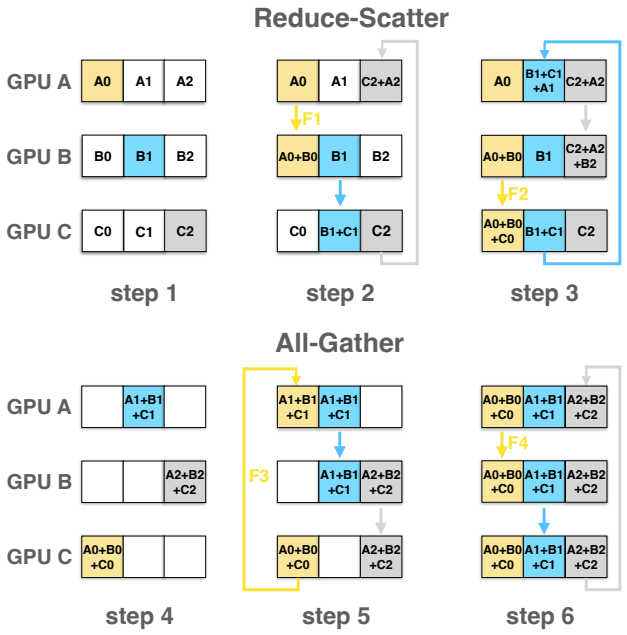

Fig. 1. *Ring-AllReduce* consists of two stages: *ReduceScatter* and *AllGather*. Each stage comprises multiple unicast flows with finish-to-start dependencies. Dependencies may occur within a stage (*e.g.,* $F_1 \rightarrow F_2$) or across stages (*e.g.,* $F_2 \rightarrow F_3$). Consecutive dependent flows form a *flow group*, highlighted by distinct colors.

Why dependencies matter for bandwidth allocation. A key observation is that dependency constraints partition the flows of a collective into sequential chains. We refer to each such chain as a *flow group*: within a flow group, at most one flow can be active at a time, since a successor is blocked until its predecessor completes. In contrast, different flow groups may be simultaneously eligible to transmit and therefore constitute the true sources of contention on shared links.

This structure is poorly captured by dependency-agnostic allocation strategies that treat all flows as concurrently competing whenever their paths overlap. Such strategies can *waste* scarce bandwidth by allocating capacity to flows that are not yet ready, effectively reducing the service rate of flows on the critical dependency frontiers. This inefficiency is especially harmful in bandwidth-constrained wide-area networks (WANs), where capacity is limited and contention is common.

Beyond routing: fixed paths, optimized transmission. Our work is motivated by the practical setting in which routing paths are determined by an existing mechanism (e.g., an overlay, a policy-driven routing layer, or a multipath protocol), and the remaining problem is to decide *how to share*

bandwidth among the resulting set of dependent flows. In other words, we consider a traffic-engineering (TE) problem *beyond routing*: given pre-defined routes and flow dependencies, compute a transmission plan that improves end-to-end collective completion time under link-capacity constraints. This focus is complementary to routing-centric optimizations and is attractive for deployment because it does not require redesigning underlying routing protocols.

Relevance and practicality in multi-datacenter training. As training increasingly spans multiple datacenters, inter-site communication runs over WAN links that are lower-bandwidth, more heterogeneous, and more contended than within-datacenter fabrics. In these environments, effective bandwidth allocation can materially affect iteration time. Moreover, distributed machine learning workloads typically involve thousands of iterations with highly repetitive communication patterns: the same sets of participants, routes, and dependency structures recur across iterations. Consequently, even if a controller requires computation to produce a dependency-aware plan, its output can be reused across many iterations, amortizing overhead while providing sustained improvements.

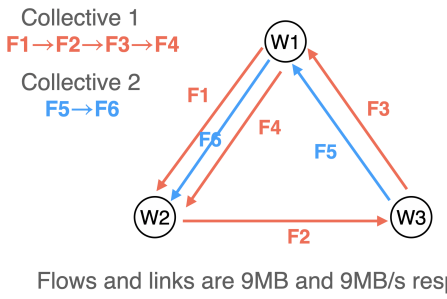

Fig. 2. A bandwidth allocation example with four 9 MB flows transmitted over 9 MB/s links. *Collective 1* comprises flow group $F_1 \to F_2 \to F_3 \to F_4$ and *Collective 2* comprises $F_5 \to F_6$.

A motivating example. Fig. 2 illustrates how ignoring dependencies can lead to inefficient sharing. Two collectives run concurrently. *Collective 1* consists of the chain $F_1 \to F_2 \to F_3 \to F_4$, and *Collective 2* consists of $F_5 \to F_6$. Each flow carries 9 MB of data, and each link has capacity 9 MB/s.

With equal (or size-proportional) *per-flow* allocation, each physical link is split among all flows that traverse it, regardless of whether they are currently eligible to transmit. For example, if a link is shared by one ready flow (say, F_1) and another flow whose predecessor has not yet completed (say, F_4), then the allocator still reserves bandwidth for F_4, even though F_4 cannot begin. Under an equal split, the ready flow receives only $9/3 = 3$ MB/s and thus takes $9/3 = 3$ s to complete. The same inefficiency repeats after the dependency frontier advances, yielding a completion time of 9 s for collective 1 and 5 s for collective 2 (and thus 7 s on average).

In contrast, a dependency-aware allocator reasons at the *flow-group* level and prioritizes ready transmissions. In the first phase, the ready flows at the heads of the two groups (here, F_1 and F_5) receive the full 9 MB/s on their respective bottlenecks and complete in $9/9 = 1$ s. Once they finish, their successors (F_2 and F_6) become ready and similarly complete in another 1 s. The average collective completion time therefore drops from 7 s to 3 s. This simple example highlights the central motivation of this paper: when dependencies exist, allocating bandwidth without regard to readiness can reduce effective utilization and increase end-to-end completion time, whereas dependency-aware allocation can translate scarce WAN bandwidth into faster collective completion.

III. Flow-Dependency-Aware Rate Allocation

This section formalizes dependency-aware bandwidth allocation for wide-area collectives under *fixed routing*. We begin with a continuous-time feasibility model (which clarifies what it means to "respect" dependencies and link capacities), then derive three tractable formulations aligned with our algorithms: a non-concurrent (scheduling) MILP, two concurrent (allocation) programs, and an integrated two-level MultiRing design. We end with a toy example that exposes why dependency-agnostic flow-level constraints can be fundamentally mismatched to collective semantics.

A. Network Model, Flows, and Dependencies

We model the WAN as a directed graph $G = (\mathcal{V}, \mathcal{E})$. Each link $e \in \mathcal{E}$ has a capacity $C_e > 0$ (bytes/s). A set of collectives run concurrently:

$$\mathcal{K} = \{1, 2, \ldots, K\}.$$

Collective $k \in \mathcal{K}$ decomposes into a set of *flow groups* $\mathcal{N}_k = \{1, 2, \ldots, N_k\}$, where each flow group is a sequential chain of finish-to-start dependent transfers, as motivated in Sec. II.

Flows. A flow in collective k and flow group n is denoted by $F^{k,n}_{i,j,o}$, representing a unicast transfer of size $F^{k,n}_{i,j,o} > 0$ (bytes) from source node i to destination node j. The index o indicates the flow's *dependency order* within its chain. Let $\mathcal{S}^{k,n}$ denote the index set of flows in group (k, n):

$$\mathcal{S}^{k,n} := \{(i, j, o) \mid F^{k,n}_{i,j,o} \text{ is a flow in group } (k, n)\}.$$

Fixed routing. We assume each flow has a pre-determined route (path) over links in $\mathcal{E}$. We use an indicator $\mathcal{I}^{k,n}_{i,j,o}(e) \in \{0, 1\}$ to denote whether link e lies on the path of flow $F^{k,n}_{i,j,o}$.

Dependencies. Dependencies are represented by precedence edges $\to$:

$$F^{k,n}_{h,l,o'} \to F^{k,n}_{i,j,o}$$

$$\Rightarrow \; F^{k,n}_{i,j,o} \text{ starts only after } F^{k,n}_{h,l,o'} \text{ completes.}$$

In Ring-AllReduce, these dependencies largely form disjoint chains. We exploit this by organizing flows into *flow groups*; within each group, at most one flow is eligible to transmit at any time, while different groups may be simultaneously eligible and thus contend on shared WAN links.

B. A Continuous-Time View: What is a Feasible Execution?

Although our solvers use simplified variables, it is helpful to state the underlying continuous-time feasibility conditions.

For each flow $F^{k,n}_{i,j,o}$, let $r^{k,n}_{i,j,o}(t) \geq 0$ be its instantaneous sending rate at time $t \geq 0$ (bytes/s). The *sent* volume by time t is $\int_0^t r^{k,n}_{i,j,o}(\tau)\, d\tau$. The *completion time* is

$$C^{k,n}_{i,j,o} := \inf\left\{ t \geq 0 \;\middle|\; \int_0^t r^{k,n}_{i,j,o}(\tau)\, d\tau \geq F^{k,n}_{i,j,o} \right\},$$

and the *start time* is

$$S^{k,n}_{i,j,o} := \inf\{t \geq 0 \mid r^{k,n}_{i,j,o}(t) > 0\}.$$

A continuous-time execution is feasible if it satisfies:

(i) Link capacity constraints (for all times). For every link $e \in \mathcal{E}$ and time t,

$$\sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}_k} \sum_{(i,j,o)\in\mathcal{S}^{k,n}} \mathcal{I}^{k,n}_{i,j,o}(e) \cdot r^{k,n}_{i,j,o}(t) \;\leq\; C_e. \tag{1}$$

(ii) Dependency constraints (finish-to-start). For every precedence edge $F^{k,n}_{h,l,o'} \to F^{k,n}_{i,j,o}$,

$$S^{k,n}_{i,j,o} \;\geq\; C^{k,n}_{h,l,o'}. \tag{2}$$

(iii) Non-negativity. $r^{k,n}_{i,j,o}(t) \geq 0$ for all flows and t.

Objective: average collective completion time. The completion time of collective k is the time when its last flow finishes:

$$T_k := \max_{n\in\mathcal{N}_k} \max_{(i,j,o)\in\mathcal{S}^{k,n}} C^{k,n}_{i,j,o}.$$

Our goal is to minimize the average completion time:

$$\min \; \frac{1}{|\mathcal{K}|} \sum_{k\in\mathcal{K}} T_k. \tag{3}$$

Problem (1)–(3) is a general coupled scheduling and rate-control problem with precedence constraints. Directly optimizing over functions $r(\cdot)$ is intractable at scale, so we next introduce restricted models that capture the dependency structure while remaining solvable.

C. Non-Concurrent Scheduling (Optimization-Based)

Our first approach is a *non-concurrent* restriction: two flows whose routes overlap are not allowed to transmit at the same time. This restriction is motivated by the observation that dependencies can prevent a system from fully utilizing bandwidth freed mid-execution; in such cases, aggressively time-sharing a bottleneck link among many flows can delay the completion of "frontier" flows that unlock successors. Non-concurrency instead dedicates bottleneck capacity to one flow at a time, which can reduce idle time induced by precedence constraints.

1) A single-link dominance property: We start with a basic structural fact on a *single* shared link. It formalizes why serialization can be attractive when flows compete on a bottleneck.

Lemma 1. *Let $F_1, \ldots, F_m > 0$ be flow sizes and $a_1, \ldots, a_m > 0$ be allocated rates. Then*

$$\frac{\sum_{i=1}^m F_i}{\sum_{i=1}^m a_i} \leq \max_{1\leq i\leq m} \frac{F_i}{a_i}.$$

Proof. Let $M = \max_i \frac{F_i}{a_i}$. Then $F_i \leq M a_i$ for all i. Summing gives $\sum_i F_i \leq M \sum_i a_i$, hence the claim. ∎

Theorem 2 (Serialization dominates time-sharing on a single link)**.** *Consider m flows sharing a link e of capacity C_e. Any feasible concurrent constant-rate allocation assigns rates $a_i > 0$ with $\sum_i a_i \leq C_e$, yielding completion times $t_i = F_i/a_i$. Order flows so that $t_1 \leq t_2 \leq \cdots \leq t_m$. If we instead transmit the flows sequentially at full rate C_e in this order, then the completion time of the i-th completed flow becomes $t'_i = (F_1 + \cdots + F_i)/C_e$, and $t'_i \leq t_i$ for all i.*

Proof. For any prefix $\{1, \ldots, i\}$, since $\sum_{j=1}^i a_j \leq C_e$,

$$t'_i = \frac{\sum_{j=1}^i F_j}{C_e} \leq \frac{\sum_{j=1}^i F_j}{\sum_{j=1}^i a_j} \leq \max_{1\leq j\leq i} \frac{F_j}{a_j} = t_i,$$

where the second inequality uses Lemma 1 and the last equality uses the ordering. ∎

Theorem 2 is a *local* statement (single-link, constant rates), but it captures a key mechanism: if a link is the dominant bottleneck and completion-time objectives reward finishing some flows early (e.g., because they unlock successors), then serializing contention can reduce dependency-induced slack.

2) Non-concurrent scheduling as a MILP: We next define a tractable non-concurrent scheduling model that replaces time-varying rates with flow start times.

Effective flow rate and duration under non-concurrency. If a flow transmits alone on its path, its end-to-end rate is limited by the minimum-capacity link on that path. Define

$$c^{k,n}_{i,j,o} := \min_{e\in\mathcal{E}:\; \mathcal{I}^{k,n}_{i,j,o}(e)=1} C_e, \qquad dur^{k,n}_{i,j,o} := \frac{F^{k,n}_{i,j,o}}{c^{k,n}_{i,j,o}}.$$

We introduce a start time variable $S^{k,n}_{i,j,o} \geq 0$ and set completion time $C^{k,n}_{i,j,o} = S^{k,n}_{i,j,o} + dur^{k,n}_{i,j,o}$.

Dependency constraints. For each edge $F^{k,n}_{h,l,o'} \to F^{k,n}_{i,j,o}$,

$$S^{k,n}_{i,j,o} \;\geq\; S^{k,n}_{h,l,o'} + dur^{k,n}_{h,l,o'}. \tag{4}$$

Conflict pairs. Two flows conflict if their routes share at least one link. Since dependencies already order flows within the same group, we only need non-concurrency across *different* flow groups. Define

$$\mathcal{P} := \left\{ (F^{k,n}_{i,j,o}, F^{k',n'}_{h,l,o'}) \;\middle|\; \begin{array}{l} (k,n) \neq (k',n'), \\ \exists\, e \in \mathcal{E} \text{ s.t. } \mathcal{I}^{k,n}_{i,j,o}(e) = \mathcal{I}^{k',n'}_{h,l,o'}(e) = 1 \end{array} \right\}.$$

Big-M non-overlap constraints. For each ordered pair $(f, g) \in \mathcal{P}$, introduce a binary variable $y_{f,g} \in \{0, 1\}$ indicating whether f is scheduled before g. The disjunction

$$C_f \leq S_g \quad \text{or} \quad C_g \leq S_f$$

is encoded by:

$$S_f + dur_f \leq S_g + M(1 - y_{f,g}), \tag{5}$$
$$S_g + dur_g \leq S_f + M y_{f,g}. \tag{6}$$

Here M is a sufficiently large upper bound on the schedule horizon (e.g., the sum of all dur's). If desired, a small guard time $\delta > 0$ can be incorporated by subtracting δ from the right-hand side.

Objective linearization. We introduce T_k for each collective k and enforce:

$$T_k \geq S^{k,n}_{i,j,o} + dur^{k,n}_{i,j,o}, \quad \forall n \in \mathcal{N}_k,\ \forall (i, j, o) \in \mathcal{S}^{k,n}.$$

The MILP minimizes the average objective:

$$\min \ \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} T_k \quad \text{s.t. (4), (5)–(6), and } S \geq 0. \tag{7}$$

We solve this MILP using MOSEK. While it can deliver strong completion-time performance, its binary variables scale with $|\mathcal{P}|$, motivating the concurrent alternatives below.

D. Concurrent Methods: Allocation Without Explicit Scheduling

The remaining methods avoid start-time variables and integer constraints by computing *static* allocations that can be applied repeatedly across iterations of training. The key modeling device is the flow-group abstraction: since at most one flow is active per group, a group behaves like a *single* consumer of bandwidth on each link at any instant.

*1) **Rate Allocation**:* Rate Allocation assigns each flow a constant rate parameter $b^{k,n}_{i,j,o} > 0$ that will be used when the flow is eligible. The challenge is to encode link feasibility without tracking which flow in each group is active over time.

A sufficient (group-based) link feasibility condition. For each link e and each flow group (k, n) that has at least one flow traversing e, introduce an auxiliary variable $E^{k,n}_e \geq 0$ and enforce

$$E^{k,n}_e \ \geq \ b^{k,n}_{i,j,o}, \qquad \forall (i, j, o) \in \mathcal{S}^{k,n} \text{ s.t. } \mathcal{I}^{k,n}_{i,j,o}(e) = 1. \tag{8}$$

Let $\mathcal{G}_e$ be the set of groups that ever use link e:

$$\mathcal{G}_e := \left\{ (k, n) \ \middle|\ \exists\, (i, j, o) \in \mathcal{S}^{k,n} \text{ with } \mathcal{I}^{k,n}_{i,j,o}(e) = 1 \right\}.$$

We then impose:

$$\sum_{(k,n) \in \mathcal{G}_e} E^{k,n}_e \leq C_e, \qquad \forall e \in \mathcal{E}. \tag{9}$$

Lemma 3 (Safety of group-max link constraints)**.** *Consider any execution that respects dependencies (at most one active flow per group at any time). If rates satisfy* (8)–(9)*, then the instantaneous link loads always satisfy* (1)*.*

Proof. Fix any link e and time t. For each group $(k, n) \in \mathcal{G}_e$, either no currently-active flow in that group uses e (contributing 0), or exactly one active flow in that group uses e (because the group is sequential), contributing at most its rate $b^{k,n}_{i,j,o} \leq E^{k,n}_e$ by (8). Therefore, the total instantaneous load on e is at most $\sum_{(k,n) \in \mathcal{G}_e} E^{k,n}_e \leq C_e$ by (9). ∎

Lemma 3 justifies viewing $E^{k,n}_e$ as the group's *per-link reservation* that upper-bounds whichever flow from that group may traverse e when active.

Completion time surrogate via flow groups. Because flows in a group execute sequentially, the completion time of group (k, n) under fixed per-flow rates is

$$\tau_{k,n} := \sum_{(i,j,o) \in \mathcal{S}^{k,n}} \frac{F^{k,n}_{i,j,o}}{b^{k,n}_{i,j,o}}.$$

A collective completes when its slowest group completes, so we introduce T_k and enforce

$$T_k \geq \tau_{k,n}, \qquad \forall n \in \mathcal{N}_k.$$

The Rate Allocation program is:

$$\begin{aligned} \min_{\{b,E,T\}} \quad & \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} T_k \\ \text{s.t.} \quad & T_k \geq \sum_{(i,j,o) \in \mathcal{S}^{k,n}} \frac{F^{k,n}_{i,j,o}}{b^{k,n}_{i,j,o}}, \qquad \forall k,\ \forall n \in \mathcal{N}_k, \\ & \sum_{(k,n) \in \mathcal{G}_e} E^{k,n}_e \leq C_e, \qquad \forall e \in \mathcal{E}, \\ & E^{k,n}_e \geq b^{k,n}_{i,j,o} \quad \text{whenever } \mathcal{I}^{k,n}_{i,j,o}(e) = 1, \\ & b^{k,n}_{i,j,o} > 0 \quad \forall k, n, (i, j, o). \end{aligned} \tag{10}$$

Lemma 4 (Convexity of Rate Allocation)**.** *Problem* (10) *is a convex optimization problem.*

Proof sketch. For $b > 0$, the function F/b is convex in b. Hence each $\tau_{k,n}$ is convex as a sum of convex terms. The constraints $T_k \geq \tau_{k,n}$ are epigraph constraints of convex functions and are convex. All remaining constraints are linear. Therefore the feasible set is convex and the objective is linear, implying convexity. ∎

In our implementation, we solve (10) using CVXPY with MOSEK, using standard reciprocal primitives (e.g., `inv_pos`) over positive variables.

*2) **Weight Allocation** (Robust Proportional Sharing):* Exact WAN capacities can vary over time due to cross traffic and measurement noise. Weight Allocation therefore computes *relative* group priorities rather than absolute rates.

Weights and per-link proportional shares. Assign each group (k, n) a weight $w^{k,n} > 0$ normalized by

$$\sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}_k} w^{k,n} = 1.$$

On link e, all groups in $\mathcal{G}_e$ share C_e proportionally:

$$s^{k,n}_e := C_e \cdot \frac{w^{k,n}}{\sum_{(k',n') \in \mathcal{G}_e} w^{k',n'}}, \qquad \forall (k, n) \in \mathcal{G}_e. \tag{11}$$

A flow's effective rate is the minimum share along its path:

$$b_{i,j,o}^{k,n} := \min_{e:\ \mathcal{I}_{i,j,o}^{k,n}(e)=1} s_e^{k,n}.$$

The same group completion surrogate applies: $\tau_{k,n} = \sum \frac{F}{b}$ and $T_k \geq \tau_{k,n}$, with objective (3).

Non-convexity and successive convex approximation. The expression (11) contains weights in both numerator and denominator, which makes the optimization over $\{w^{k,n}\}$ non-convex. We therefore apply an iterative scheme: at iteration t, we fix the denominator

$$D_e^{(t-1)} := \sum_{(k',n')\in\mathcal{G}_e} w^{k',n',(t-1)}$$

and optimize only the numerators, yielding the convex approximation

$$\tilde{s}_e^{k,n,(t)} := C_e \cdot \frac{w^{k,n,(t)}}{D_e^{(t-1)}}.$$

We solve the resulting convex subproblem, update weights, evaluate the true objective under (11), and iterate until convergence (Algorithm 1).

Algorithm 1: Iterative Weight Allocation (Successive Convex Approximation)

Input: Flow groups $\{\mathcal{S}^{k,n}\}$, flow sizes $F_{i,j,o}^{k,n}$, capacities C_e, routes $\mathcal{I}_{i,j,o}^{k,n}(e)$, threshold ϵ, max iterations T.

Output: Weights $\mathbf{w}$ and average completion time $\bar{\tau}$.

Initialize $\mathbf{w}^{(0)}$ using an inverse-bottleneck heuristic; normalize so $\sum w^{(0)} = 1$;

Compute $\bar{\tau}^{(0)}$ under exact proportional sharing (11);

for $t \leftarrow 1$ **to** T **do**

  Compute $D_e^{(t-1)} = \sum_{(k,n)\in\mathcal{G}_e} w^{k,n,(t-1)}$ for all e;

  Solve convex subproblem with shares $\tilde{s}_e^{k,n,(t)} = C_e \cdot w^{k,n,(t)} / D_e^{(t-1)}$;

  Normalize $\mathbf{w}^{(t)}$ so $\sum w^{(t)} = 1$;

  Evaluate true $\bar{\tau}^{(t)}$ using exact proportional sharing (11);

  if $|\bar{\tau}^{(t)} - \bar{\tau}^{(t-1)}| < \epsilon$ **then**

    break;

return $\mathbf{w}^{(t)}$, $\bar{\tau}^{(t)}$;

E. Integrated MultiRing: Coarse Allocation + Fine Scheduling

The non-concurrent MILP can produce strong completion times but scales poorly with many flow conflicts, while concurrent methods scale but may be conservative. We therefore propose MULTIRING, which combines a *fast inter-collective* allocation with an *intra-collective* dependency-aware schedule.

Inter-collective allocation: At the inter-collective layer, each collective k receives a bandwidth budget $b^k > 0$. Let F^k be the total data volume of collective k. Define the set of collectives that traverse link e:

$$\mathcal{K}_e := \left\{ k \;\middle|\; \exists\, n, (i,j,o) \in \mathcal{S}^{k,n} \text{ s.t. } \mathcal{I}_{i,j,o}^{k,n}(e) = 1 \right\}.$$

We compute $\{b^k\}$ by solving:

$$\begin{aligned} \min_{\{b^k\}} \quad & \frac{1}{|\mathcal{K}|} \sum_{k\in\mathcal{K}} \frac{F^k}{b^k} \\ \text{s.t.} \quad & \sum_{k\in\mathcal{K}_e} b^k \leq C_e, \qquad \forall e \in \mathcal{E}, \\ & b^k > 0, \qquad \forall k \in \mathcal{K}. \end{aligned} \tag{12}$$

This program is convex for the same reason as Lemma 4.

Intra-collective scheduling: Given b^k, we schedule flows *within* each collective to respect dependencies and mitigate intra-collective conflicts. For flow $F_{i,j,o}^{k,n}$, define duration $dur_{i,j,o}^{k,n} = F_{i,j,o}^{k,n}/b^k$. We then solve, for each k, a per-collective version of the non-concurrent MILP using: (i) dependency constraints (4); and (ii) conflict pairs restricted to flows in the same collective but different groups:

$$\mathcal{P}_k := \left\{ \begin{matrix} (F_{i,j,o}^{k,n}, F_{h,l,o'}^{k,n'}) \mid n \neq n', \\ \exists\, e \in \mathcal{E} \text{ s.t. } \mathcal{I}_{i,j,o}^{k,n}(e) = \mathcal{I}_{h,l,o'}^{k,n'}(e) = 1 \end{matrix} \right\}.$$

This decomposition controls MILP size and concentrates integer decisions where they matter most: within-collective dependency frontiers.

Greedy alternatives. To obtain millisecond-level runtimes, we also implement greedy schedulers that repeatedly select among *ready* flows (all predecessors completed) while respecting link conflicts, prioritizing either (i) shortest duration or (ii) largest downstream dependent volume. These heuristics approximate the intra-collective MILP while preserving dependency awareness.

F. Toy Example: Why Flow Groups Change the Constraints

We conclude with a minimal example that illustrates two distinct effects: (1) why flow-group modeling avoids allocating bandwidth to non-eligible dependent flows, and (2) why serialization can further reduce average completion time by accelerating dependency frontiers.

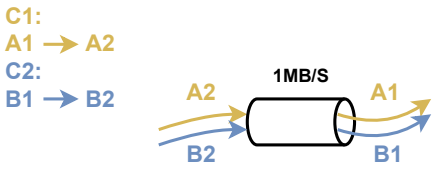

Fig. 3. Toy example with two collectives ($A_1 \to A_2$ and $B_1 \to B_2$) sharing a bottleneck link e ($C_e = 1$ MB/s). Only the first flows (A_1, B_1) are eligible initially; successors become eligible after their predecessors complete.

Setup. See Fig. 3. Consider a single bottleneck link e with capacity $C_e = 1$ MB/s. Two collectives $k \in \{1, 2\}$ run concurrently. Each collective contains one flow group (a chain of two flows):

$$\text{Collective 1: } A_1 \to A_2, \qquad \text{Collective 2: } B_1 \to B_2,$$

where all flows traverse the same link e, and each flow has size 1 MB. Thus, at time 0, only A_1 and B_1 are eligible; A_2 and B_2 are blocked.

Dependency-agnostic per-flow equal sharing (wasted bandwidth). A naive per-flow static split among all four flows gives each flow rate $1/4$ MB/s, so eligible flows complete in $1/(1/4) = 4$ s. Only then do A_2 and B_2 become eligible, and they also take 4 s. Hence each collective completes in 8 s, and the average completion time is 8 s.

Group-aware concurrent sharing (frontier-aware). A flow-group-aware allocator recognizes that each group contributes *at most one* active flow at any time, so it splits the link across the two groups, e.g., $E_e^{1,1} = E_e^{2,1} = 1/2$ MB/s. Now A_1 and B_1 complete in $1/(1/2) = 2$ s; then A_2 and B_2 complete in another 2 s. Each collective completes in 4 s on average—a $2\times$ improvement over 8 s.

Non-concurrent scheduling (accelerating one dependency chain). Finally, consider a non-concurrent schedule that serializes eligible flows: run A_1 at full rate for 1 s, then B_1 for the next 1 s. At $t = 1$, A_2 becomes eligible, but waits while B_1 finishes at $t = 2$. Then run A_2 from $t = 2$ to $t = 3$, and B_2 from $t = 3$ to $t = 4$. Thus $T_1 = 3$ s and $T_2 = 4$ s, and the average completion time is 3.5 s, which is strictly better than the concurrent 4 s in this example. The gain comes from finishing one chain early and immediately unlocking its successor with full bandwidth.

This toy example mirrors the core motivation of our formulations: with finish-to-start dependencies, the true contention unit is the *flow group* (the dependency chain), and accelerating the dependency frontier (sometimes via serialization) can reduce end-to-end collective completion time beyond what dependency-agnostic sharing can achieve.

IV. Performance Evaluation

We are now ready to evaluate the effectiveness and practicality of our *flow-dependency-aware* allocation framework (Sec. III) for wide-area collectives under *fixed routing*. Our experiments are designed to answer four questions:

(Q1) Effectiveness: Does explicitly modeling finish-to-start dependencies (via flow groups) reduce average collective completion time compared to dependency-agnostic heuristics?

(Q2) Scalability and robustness: How do the proposed methods behave as we increase (i) the number of concurrent collectives and (ii) WAN heterogeneity/topology size?

(Q3) Competitiveness: How does dependency-aware fixed-path allocation compare against TE-CCL [6], a routing-centric traffic-engineering scheduler?

(Q4) Practicality: Do the simulation trends persist under emulation with real protocol stacks and runtime variability (Nextmini [10])?

A. Metric and Evaluation Methodology

Primary metric (objective-consistent). We report the *average collective completion time*, which matches the objective in Eq. (3). Let T_k denote the completion time of collective k (the finish time of its last flow; see Sec. III). For an experiment instance with K concurrent collectives, we report

$$\bar{T} \;:=\; \frac{1}{K}\sum_{k=1}^{K} T_k. \tag{13}$$

When a plot contains multiple operating points (e.g., varying K), each point is computed by instantiating the corresponding traffic set and solving/allocating using the specified method.

Execution model in simulation. All methods are evaluated under the same *dependency-respecting* execution semantics: within each flow group (Sec. II), at most one flow is eligible (ready) at a time. Given an allocation outcome (static rates, weights, or a schedule), we compute T_k by simulating the induced transmission process with finish-to-start precedence constraints. Concretely, we maintain the set of *ready* flows (those whose predecessors are complete), advance time to the next flow completion event under the method's rate rule, then unlock successors. This event-driven procedure implements the continuous-time feasibility model in Sec. III while remaining efficient.

Why fixed routing matters. A central goal of this paper is to isolate the effect of *dependency-aware rate allocation beyond routing* (Sec. II). Unless explicitly stated (TE-CCL comparison), all schemes operate on the same pre-determined paths, so improvements are attributable to better allocation/scheduling under dependencies rather than to route changes.

B. Workloads, Topologies, and Instance Generation

WAN topologies (Topology Zoo). Our Python simulation uses six WAN-like topologies from Topology Zoo [11]: Abilene (11 nodes), Cesnet (10), Gblnet (8), HiberniaIreland (8), Napnet (6), and Arpanet (4). We treat each undirected WAN link as two directed links of identical capacity for routing/allocation.

Ring-AllReduce traffic with dependencies. Each collective instance models Ring-AllReduce (RAR) communication, as motivated in Sec. II. We instantiate the set of unicast transfers implied by RAR and attach the corresponding finish-to-start dependencies, then *coarsen* these precedence relations into *flow groups* (dependency chains). This flow-group abstraction is exactly the unit used by our formulations in Sec. III.

Flow sizes, capacities, and fixed routing. To reflect WAN heterogeneity while retaining controlled comparisons, we generate: (i) flow sizes i.i.d. from a normal distribution with mean 5 MB and standard deviation 2.5 MB (truncated to remain positive), and (ii) link capacities i.i.d. from a normal distribution with mean 22.5 MB/s and standard deviation 2.5 MB/s (truncated to remain positive). Each unicast flow is routed on a shortest path in the WAN topology, yielding a fixed indicator $\mathcal{I}_{i,j,o}^{k,n}(e)$ (Sec. III). Unless stated otherwise, all compared methods operate on these same fixed routes.

Concurrency and stress testing. To study the effect of contention, we vary the number of concurrent collectives K. On Abilene, we sweep K from 2 to 8 (Fig. 4) to create progressively heavier overlap on bottleneck links. We also evaluate across multiple topologies of different sizes (Fig. 5) to test robustness to structural differences.

C. Compared Schemes

We compare three classes of schemes.

Proposed dependency-aware schemes.

- **RATEALLOC** (Sec. III-D1): solves the convex program in Eq. (10) to assign per-flow rates $b_{i,j,o}^{k,n}$ while enforcing *group-max* link reservations that are provably safe under dependencies (Lemma 3).
- **WEIGHTALLOC** (Sec. III-D2): computes per-group weights $w^{k,n}$ for proportional sharing, targeting robustness to WAN capacity variability.
- **MULTIRING** (Sec. III): integrates a coarse inter-collective allocation with a dependency-aware intra-collective scheduler, bridging scalability and fidelity.

Dependency-agnostic and lightweight heuristics. We include widely used baselines that either ignore dependencies or incorporate them only indirectly:

- ***OutofOrder***: equal sharing at the *flow* level without dependency awareness. Rates are computed as if all flows can compete simultaneously; blocked flows therefore induce *implicit underutilization* because reserved bandwidth cannot be used by ready flows (Sec. II).
- ***EqualAlloc***: equal sharing at the *flow-group* level, serving as a group-aware baseline that does not optimize for $\bar{T}$.
- ***DataAware***: allocates at the flow-group level proportional to group data volume, capturing the intuition that larger groups should receive more bandwidth.
- ***Barrier-Aware*** [9]: a synchronization-aware baseline that accounts for barrier-style completion semantics but does not model intra-collective finish-to-start dependencies.

Routing-centric TE baseline. *TE-CCL* [6] jointly optimizes routing and bandwidth via a multi-commodity flow formulation over discretized time epochs. We compare against both TECCL-fast and TECCL-slow variants as described in the TE-CCL paper (Sec. IV-E).

D. Simulation Results on WAN Topologies

1) Impact of dependency awareness under increasing concurrency (Abilene): Fig. 4 reports $\bar{T}$ on Abilene while increasing the number of concurrent RAR collectives K. Two consistent trends emerge.

(1) Dependency-agnostic flow-level allocation degrades quickly with K. As K grows, the number of flows that *exist* (and thus are counted by flow-level equal sharing) increases, but the number of flows that are *ready* at a given time increases far more slowly due to precedence constraints. Consequently, methods like *OutofOrder* tend to reserve bandwidth for flows that cannot transmit yet, reducing the instantaneous service rate of the dependency frontier. This creates two compounding effects: (a) slower completion of the frontier flows, and (b) delayed unlocking of successors, both of which directly increase $\bar{T}$.

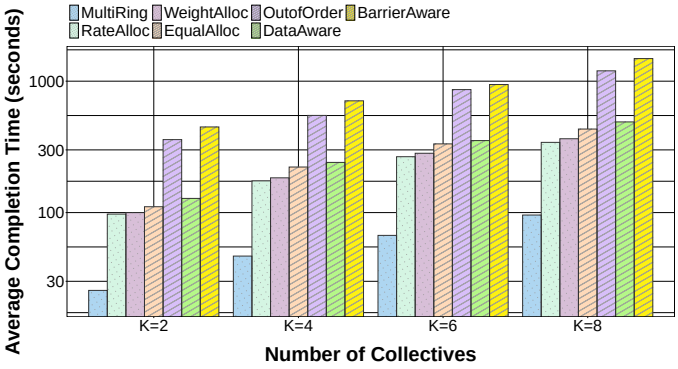

Fig. 4. Abilene (11-node) topology: average collective completion time $\bar{T}$ for multiple approaches under increasing numbers of concurrent collectives K.

(2) Modeling flow groups converts wasted reservations into useful throughput. In contrast, RATEALLOC and WEIGHTALLOC explicitly treat each flow group as contributing at most one active flow at a time. By enforcing feasibility at the level of group-max reservations (Eq. (9)) or proportional group shares (Eq. (11)), these methods allocate bandwidth to units that can actually contend. The improvement is most visible at higher K, where dependency-induced blocking is pervasive, and where any suboptimal allocation of scarce WAN capacity has amplified impact on completion time.

Where simple group-aware baselines fall short. Group-aware heuristics such as *EqualAlloc* and *DataAware* already avoid the most severe form of "bandwidth to blocked flows" waste, but remain noticeably behind the optimization-based schemes. Intuitively, equal or size-proportional splitting is not aligned with minimizing $\bar{T}$ when contention is multi-link and heterogeneous: the globally critical groups are those whose end-to-end rate is constrained by shared bottlenecks and whose completion gates the collective makespan. RATEALLOC optimizes precisely for this objective by directly minimizing an epigraph form of the sum of per-group reciprocals (Eq. (10)).

2) Robustness across topologies (Topology Zoo): Fig. 5 evaluates the same methods across five additional WAN topologies. The key observation is that the gains from dependency-aware allocation persist across: (i) different node counts (from 4 to 10), (ii) different path overlap patterns (which affect the conflict structure $\mathcal{G}_e$), and (iii) heterogeneous bottleneck placements.

This consistency supports the main thesis of the paper: the benefit does not rely on a particular "nice" topology; rather, it derives from correcting a modeling mismatch that appears whenever (a) flows are precedence-constrained and (b) capacity is scarce enough that allocating to non-ready flows is costly. Notably, WEIGHTALLOC remains competitive across topologies, suggesting that proportional weight-based control can deliver robust performance even when absolute

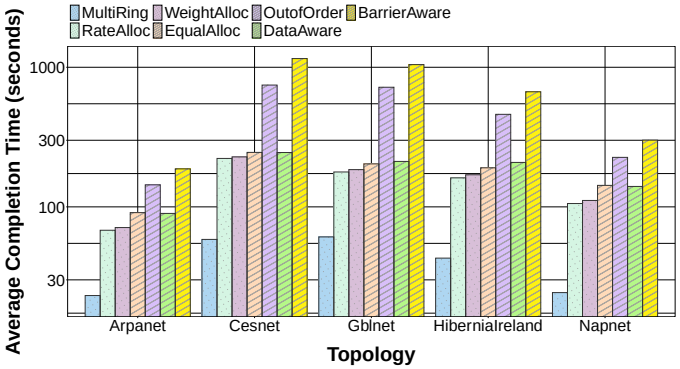

Fig. 5. Average completion time $\bar{T}$ across five additional Topology Zoo WAN topologies of different sizes and structures (Arpanet, Cesnet, Gblnet, HiberniaIreland, Napnet).

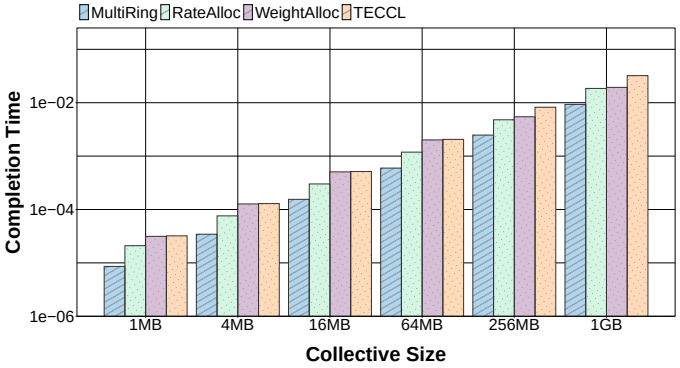

Fig. 6. Comparison against *TECCL-fast* on Abilene for different collective sizes. Bars show MULTIRING, RATEALLOC, WEIGHTALLOC, and TECCL-fast. The y-axis is log-scaled average completion time.

bandwidth varies—a central design goal for WAN deployment (Sec. III-D2).

E. Comparison with TE-CCL

TE-CCL [6] represents a complementary point in the design space: it uses a traffic-engineering formulation that can *jointly* choose routes and schedule transmission over time epochs. This section evaluates whether explicitly modeling flow dependencies under fixed paths can match or exceed the completion-time benefits of routing-centric optimization, and clarifies when each approach is advantageous.

Experimental setup. We evaluate on the Abilene topology while varying the *collective size* (i.e., scaling the amount of traffic carried by the collective), keeping the topology and contention structure fixed. We compare against both TECCL-fast and TECCL-slow, which differ in their epoch durations: TECCL-fast uses a finer epoch (fastest-link chunk time) enabling tighter pipelining but introducing more constraints, whereas TECCL-slow uses a coarser epoch (slowest-link chunk time) that is cheaper to solve but less expressive.

Results and interpretation. Across the tested size range, Figs. 6 and 7 show that our dependency-aware allocations remain competitive with, and often outperform, both TECCL

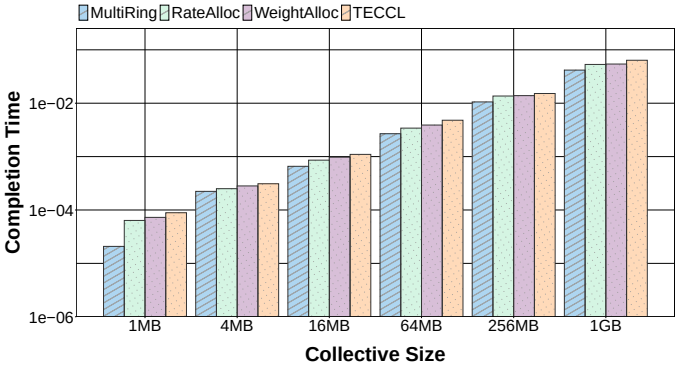

Fig. 7. Comparison against *TECCL-slow* on Abilene for different collective sizes. Bars show MULTIRING, RATEALLOC, WEIGHTALLOC, and TECCL-slow. The y-axis is log-scaled average completion time.

variants. Two mechanisms explain why this can occur even though TE-CCL has greater freedom through routing.

(1) Dependency modeling is orthogonal to routing. TE-CCL's formulation focuses on bandwidth feasibility across time and paths but does not, by itself, encode the fine-grained finish-to-start semantics of RAR as flow-group precedence constraints. When dependency frontiers dominate the makespan, allocating bandwidth to ineligible transfers (or failing to prioritize unlock-critical transfers) can negate the advantage of improved routing. Our methods directly target this frontier effect by constraining contention at the flow-group level (Lemma 3) and by minimizing an objective consistent with Eq. (3).

(2) Epoch discretization can introduce scheduling slack. TECCL-slow, in particular, operates on coarse epochs; such discretization can force allocations to remain constant over an epoch even when dependency frontiers advance within that time window, which effectively delays reallocation to newly ready transfers. This phenomenon becomes more pronounced as collective size grows and the number of dependency steps increases. Our convex programs operate in continuous time at the level of static parameters (rates or weights) and therefore avoid epoch-induced rigidity.

Overall, these results suggest that explicitly modeling dependencies is a powerful and largely independent lever from routing-centric TE optimization: even with fixed paths, dependency-aware allocation can translate WAN bandwidth into lower end-to-end collective completion time.

F. Emulation Results on Nextmini

Simulation results isolate the algorithmic effect under controlled models; we next validate that the same trends hold under a realistic execution environment with OS scheduling and protocol overhead.

Testbed and workload. We use Nextmini [10], a Rust-based network emulation platform that enforces bandwidth constraints while measuring flow-level behavior. We employ Nextmini's *namespace mode*, where a single host process spawns all nodes as lightweight Linux network namespaces connected via virtual Ethernet (veth) pairs. This eliminates

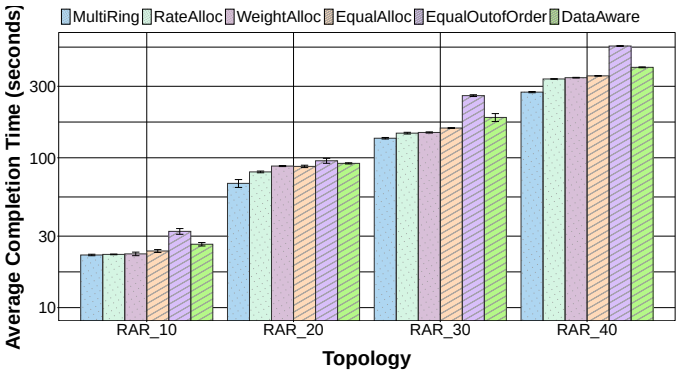

Fig. 8. Nextmini emulation: mean average collective completion time for seven approaches on four synthetic ring topologies (10–40 nodes). Each configuration is repeated 5 times; error bars show the standard deviation across runs.

per-node container overhead and image management, significantly reducing startup latency and resource consumption while preserving the same flow-level semantics and bandwidth enforcement. The controller and database continue to run in Docker, while the dataplane nodes operate directly on the host. We synthesize ring topologies with 10, 20, 30, and 40 nodes and run RAR-like traffic with dependencies matching Sec. II. Because emulation outcomes can vary due to host and namespace effects, we repeat each configuration five times and report the mean and standard deviation.

Results. Fig. 8 shows that dependency-aware allocation reduces completion time not only in the simulator but also when executing real traffic over an emulated WAN. Importantly, the gap relative to dependency-agnostic baselines persists despite run-to-run noise, indicating that the improvement is not a brittle artifact of idealized assumptions. This aligns with the paper's deployment motivation: in WAN settings where available bandwidth and contention can fluctuate, the dominant inefficiency is often *structural* (allocating to non-ready dependent flows), and correcting that structure yields robust gains.

Summary. Across multiple Topology Zoo WAN graphs, increasing concurrency levels, comparisons to TE-CCL, and Nextmini emulation, the results consistently support the same conclusion: *explicitly accounting for finish-to-start flow dependencies is critical for converting scarce WAN bandwidth into lower end-to-end collective completion time*. Our optimization-based dependency-aware methods reduce wasted capacity allocated to blocked flows, prioritize dependency frontiers, and remain competitive against state-of-the-art routing-centric schedulers in the evaluated settings.

V. Related Work

Our work lies at the intersection of ① collective communication optimization for distributed machine learning and ② network-side resource allocation and scheduling. A central theme that distinguishes our approach from related work is the explicit modeling of *finish-to-start* dependencies among unicast transfers inside a collective, captured via the *flow-group* abstraction and enforced in our optimization formulation.

Collective communication libraries and algorithm synthesis. Widely used libraries such as NCCL [12] implement Ring-AllReduce and related primitives with carefully engineered pipelines, but are typically optimized for homogeneous and high-bandwidth intra-datacenter environments. A large body of work improves performance by changing the *collective algorithm* or its execution topology. Blink [13], for example, is reported to improve synchronization by probing available bandwidth and constructing communication structures that better exploit heterogeneous links, addressing scenarios where a single slow link throttles throughput. TACCL [5] synthesizes collectives tailored to a given topology and is reported to outperform standard implementations by generating communication schedules that better match hardware constraints. These approaches primarily operate at the library level by changing *which transfers occur* (or their logical ordering/topology) to improve utilization. Our work is complementary: we do not redesign the collective algorithm or require replacing NCCL-like implementations. Instead, we assume that the set of transfers and their routes are determined (e.g., by a chosen collective algorithm and a routing layer) and optimize *how bandwidth is shared* among the resulting dependent transfers under WAN capacity constraints. This design is motivated by multi-datacenter training settings, where routing and policy constraints may be fixed, and where respecting dependency readiness is critical to avoid allocating bandwidth to non-eligible flows (Sec. II).

Routing-centric traffic engineering for collectives. TE-CCL [6] takes a traffic-engineering perspective and formulates collective communication as a (time-discretized) optimization problem that jointly selects paths and bandwidth shares, treating collective traffic as commodities in a multi-commodity flow framework. This approach is powerful when routing flexibility is available and when time-epoch constraints capture the execution semantics. Our work differs along two axes. First, we focus on the *fixed-route* regime, isolating a deployable control knob: rate allocation/scheduling *beyond routing*. Second, we incorporate *intra-collective finish-to-start dependencies* induced by Ring-AllReduce by modeling sequential chains as flow groups and enforcing dependency-respecting feasibility.

End-host congestion control and decentralized prioritization. Another line of work improves collective performance without centralized optimization by modifying congestion control or end-host behavior. MLTCP [14], for instance, augments TCP to reduce the impact of bursty collective traffic by adjusting sending behavior based on progress toward completion, inducing an implicit scheduling effect among flows. Our approach targets a different deployment point. Rather than modifying congestion-control dynamics, we assume a centralized controller computes dependency-aware allocations (rates or weights) and reuses them across iterations for recurring communication patterns.

Coflow scheduling, barrier-aware allocation, and job-

level abstractions. From a networking viewpoint, a collective can be interpreted as a structured set of related flows with a shared completion objective, which is closely connected to the coflow abstraction [8]. Prior work on coflows studies scheduling and bandwidth allocation to minimize coflow completion time (CCT) or job completion time (JCT), typically in datacenter fabrics. Varys [15] proposes an offline coflow scheduler that exploits knowledge of coflow sizes, while Aalo [16] develops an online approach that approximates shortest-job-first behavior under uncertainty. Utility-based formulations have also been explored; for example, Chen *et al.* [7] study objectives based on fairness/utility (e.g., max-min style utilities) to balance competing coflows. Barrier-aware allocation [9] is particularly relevant because it recognizes synchronization effects: a stage (or job phase) cannot complete until all its constituent transfers complete.

WAN traffic engineering systems. Classical WAN TE systems such as B4 [17] and SWAN [18] use centralized controllers to compute bandwidth allocations (and sometimes routes) under link-capacity constraints, achieving high utilization and predictable performance, and typically treat each traffic class as continuously backlogged demand. Our formulation is inspired by this TE viewpoint but differs in workload semantics and objective: collective communication traffic is *structured* by precedence constraints and is evaluated by completion-time metrics rather than steady-state throughput alone. Allocating capacity to non-ready dependent transfers does not accelerate completion and can delay the dependency frontier; by explicitly incorporating these constraints, our work bridges TE-style capacity modeling with collective-specific semantics. In contrast to prior work that ① redesigns collectives, ② jointly optimizes routing and bandwidth without modeling fine-grained precedence, or ③ schedules at coflow/stage granularity, we focus on *dependency-aware allocation under fixed routes*, treating the dependency chain as the fundamental contention unit (flow groups) to enable completion-time-driven optimization for WAN-based multi-datacenter training.

VI. Concluding Remarks

Wide-area distributed training is common, yet WAN collective communication remains difficult to optimize because it combines stringent capacity constraints with *algorithm-imposed execution semantics*. Widely deployed collectives such as Ring-AllReduce are not well modeled as schedulable flows: their transfers exhibit *finish-to-start* precedence constraints, inducing sequential dependency chains. Ignoring these constraints can allocate capacity to flows that are not yet eligible, slowing the dependency frontier that governs completion time. To address this mismatch, we introduced a flow-group abstraction and derived optimization programs and algorithms that respect per-link capacity and flow readiness.

We showed that, on a shared bottleneck, serialization can dominate concurrent time-sharing in terms of prefix completion times, and proposed four practical algorithms (RateAlloc, WeightAlloc, MultiRing, and a non-concurrent MILP scheduler). Simulations on Topology Zoo WAN topologies and Nextmini-based emulation confirm that modeling dependencies consistently reduces average collective completion time relative to common baselines, with performance comparable to or better than TE-CCL.

References

[1] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[2] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[3] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: geo-distributed machine learning approaching lan speeds," in *Proc. 14st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 629–647.

[4] G. F. Pfister, "An introduction to the infiniband architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, p. 102, 2001.

[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)*, 2023, pp. 593–612.

[6] X. Liu, B. Arzani, S. K. R. Kakarla, L. Zhao, V. Liu, M. Castro, S. Kandula, and L. Marshall, "Rethinking machine learning collective communication as a multi-commodity flow problem," in *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, 2024, pp. 16–37.

[7] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2016, pp. 1–9.

[8] M. Chowdhury and I. Stoica, "Coflow: a networking abstraction for cluster applications," in *Proc. 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[9] L. Chen, B. Li, and B. Li, "Barrier-aware max-min fair bandwidth sharing and path selection in datacenter networks," in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 151–160.

[10] X. Zhang, S. Chang, and B. Li, "Nextmini: A New Research Testbed for Network Emulation and Experimentation," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2026.

[11] University of Adelaide, "The internet topology zoo," http://www.topology-zoo.org/.

[12] NVIDIA Corporation, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl, 2016.

[13] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ml," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 172–186, 2020.

[14] S. Rajasekaran, S. Narang, A. A. Zabreyko, and M. Ghobadi, "Mltcp: A distributed technique to approximate centralized flow scheduling for machine learning," in *Proc. 23rd ACM Workshop on Hot Topics in Networks*. ACM, 2024, pp. 167—-176.

[15] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, 2014, pp. 443–454.

[16] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 393–406, 2015.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proc. ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013, pp. 15–26.