Harvest: Adaptive Photonic Switching Schedules for Collective Communication in Scale-up Domains

Mahir Rahman[♦] Samuel Joseph[♦] Nihar Kodkani[♦] Behnaz Arzani[♠] Vamsi Addanki[♦]
[♦]Purdue University [♠]Microsoft Research

Abstract

As chip-to-chip silicon photonics gain traction for their bandwidth and energy efficiency, their circuit-switched nature raises a fundamental question for collective communication: *when* and *how* should the interconnect be reconfigured to realize these benefits? Establishing direct optical paths can reduce congestion and propagation delay; however, each reconfiguration incurs non-negligible overhead, making naive per-step reconfiguration impractical.

We present Harvest, a systematic approach for synthesizing topology reconfiguration schedules that minimize collective completion time in photonic interconnects. Given a collective communication algorithm and its *fixed* communication schedule, Harvest determines how the interconnect should evolve over the course of the collective, explicitly balancing reconfiguration delay against congestion and propagation delay. We reduce the synthesis problem into a dynamic program with an underlying topology optimization subproblem and show that the approach applies to arbitrary collective communication algorithms. Furthermore, we exploit the algorithmic structure of a well-known AllReduce algorithm (Recursive Doubling) to synthesize *optimal* reconfiguration schedules without using any optimizers. By parameterizing the formulation using reconfiguration delay, Harvest naturally adapts to various photonic technologies. Using packet-level and flow-level evaluations, as well as hardware emulation on commercial GPUs, we show that the schedules synthesized by Harvest significantly reduce collective completion time across multiple collective algorithms compared to static interconnects and reconfigure-every-step baselines.

CCS Concepts

• **Networks → Network performance analysis**; **Network algorithms**; • **Theory of computation → Design and analysis of algorithms**.

Keywords

Photonic Interconnects, Collective Communication

ACM Reference Format:
Mahir Rahman[♦] Samuel Joseph[♦] Nihar Kodkani[♦] Behnaz Arzani[♠] Vamsi Addanki[♦] . 2026. Harvest: Adaptive Photonic Switching Schedules for Collective Communication in Scale-up Domains . In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/3789240.3829166

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/2026/08
https://doi.org/10.1145/3789240.3829166

1 Introduction

The explosive growth of AI/ML workloads [14, 24, 33, 53, 58, 68, 82], together with the increasing scale of distributed computing infrastructure [24, 25, 58], has led to rapidly rising demands on network bandwidth and energy efficiency. The performance of these workloads critically depends on collective communication among GPUs, such as AllReduce and All-to-All [16, 47, 65, 66]. Modern hyperscale systems consist of large numbers of multi-GPU servers interconnected by packet-switched networks that support GPU-to-GPU communication [54]. Despite their widespread adoption, these interconnects face fundamental limitations. Electrical links are power-hungry and generate significant heat [73], raising concerns for scalability and sustainability. At the same time, intra-server GPU interconnects rely on CMOS-based technologies whose bandwidth has not kept pace with GPU compute growth [9, 33, 49, 50]. As the slowdown of Moore's Law continues to widen the gap between computation and communication [9], these constraints become especially pronounced in scale-up systems, where limited interconnect bandwidth, such as PCIe, increasingly bottlenecks collective performance.

Silicon photonics promise substantially higher bandwidth and improved energy efficiency [21, 37, 40, 75], making them an attractive alternative. At the same time, their circuit-switched nature introduces new challenges for collective communication due to non-negligible reconfiguration delays. A static photonic topology avoids reconfiguration overhead but inevitably suffers from congestion due to multi-hop forwarding between GPUs. Conversely, a dynamically reconfigurable topology can, in principle, eliminate congestion by establishing direct optical paths between communicating GPUs, but only at the cost of reconfiguration delay. Balancing this fundamental trade-off is essential to realizing the practical benefits of silicon photonics.

Prior work on optical circuit-switched networks largely falls into three categories. One line of work advocates one-shot or infrequent reconfiguration when reconfiguration overhead is high [76], effectively treating the topology as static during execution. Another line assumes that reconfiguration overhead is negligible and relies on periodic or demand-aware reconfiguration [3, 5, 6, 26, 45, 56], often using Birkhoff–von Neumann (BvN) decompositions [12] of aggregate traffic matrices. A third category explicitly incorporates reconfiguration delay into the optimization objective [13, 44, 46], but still adopts a traffic-matrix abstraction and restricts routing to single-hop paths within each topology choice[1].

While appropriate for bulk or steady-state traffic, these abstractions fundamentally ignore the step-wise structure of collective communication. Unlike bulk traffic, collective communication is *staged*: progress unfolds over a sequence of steps, and the communication pattern at each step has *known* dependencies on prior ones. This structure creates an opportunity to plan reconfigurations across multiple

[1]These works permit multi-hop forwarding only across reconfiguration events [13], or rely on auxiliary electrical interconnects when reconfiguration delays are high [46].

steps, exploiting future knowledge to amortize reconfiguration delay against reductions in congestion and propagation delay. However, existing approaches, whether they ignore reconfiguration overhead, restrict reconfiguration during execution, or optimize over aggregate traffic matrices, fail to capture these step-level dependencies. As a result, they forgo a significant opportunity to reduce collective completion time. This gap in the literature raises a natural question:

To what extent can reconfigurability be exploited to reduce collective completion time?

Answering this question requires deciding *when* and *how* to reconfigure a photonic interconnect *during* a collective, while balancing reconfiguration delay against congestion and propagation delay. Our approach is grounded in two observations that together provide a principled way to reason about this trade-off.

First, unlike prior work that applies Birkhoff–von Neumann (BvN) decompositions to *aggregate* traffic matrices, many collective communication algorithms admit a natural BvN representation at the level of individual communication steps [12]. Each step corresponds to a matching, and the collective as a whole can be viewed as a weighted sequence of such matchings. This structure arises directly from the staged nature of collective primitives, which are traditionally designed around point-to-point communication patterns[2] [18]. Second, this representation connects naturally to performance analysis.

Based on these observations, we can express the completion time of each step through the lens of maximum concurrent flow [67], which permits multi-hop forwarding under a chosen topology. Interestingly, this representation uncovers the classic α–β cost model for collective communication, explicitly accounting for network congestion.

The model can then be extended for reconfigurable interconnects by explicitly accounting for reconfiguration overhead, where each topology change contributes an additional delay α_r to the overall completion time. This new formulation quantifies collective completion time in a way that explicitly captures reconfiguration overhead.

Finally, we cast this formulation as an optimization problem that synthesizes circuit switching schedules which adapt to the underlying reconfiguration delay and determine *when* and *how* the interconnect should reconfigure to minimize total collective completion time.

We present Harvest, a framework that synthesizes optimal photonic switching schedules for any *given* collective communication algorithm. The key insight underlying Harvest is that topology synthesis exhibits a natural recurrence over contiguous ranges of communication steps, which enables a dynamic programming formulation. Each subproblem selects an optimal topology for a sequence of steps executed without reconfiguration. The resulting subproblem resembles degree-bounded, demand-aware network design, with a crucial distinction: communication demand is not available upfront, but is revealed sequentially, as each step of the collective depends on the completion of prior ones. To capture this temporal structure, we formulate the subproblem as a Mixed-Integer Second-Order Conic Program (MISOCP) and integrate it with the dynamic program to construct a globally optimal reconfiguration schedule. We synthesize topologies offline, computing the schedule once and reusing it across all executions of the collective. Our framework applies to arbitrary collective communication algorithms and is suitable for scale-up domains with typical network sizes ranging from 8 to 64 GPUs.

We further apply Harvest to the Recursive Doubling AllReduce algorithm [59], which exhibits additional structure that substantially simplifies schedule synthesis. Exploiting this structure, we show that optimal topology reconfigurations can be computed with polylogarithmic complexity, and empirically within tens of microseconds, even for interconnects with up to 1024 GPUs.

We evaluate Harvest using extensive packet-level simulations in Astra-Sim [77], numerical evaluations using Gurobi [29], and testbed emulation. Across a range of collective algorithms, including Swing [65], Recursive Doubling [38], Bine butterflies [20], binomial trees, and Bruck's algorithms [15], the topologies synthesized by Harvest reduce collective completion time by up to $\approx$ 2x even compared to the best strategy among static topologies and BvN schedules that reconfigure at every step. We further measure the synchronization overhead introduced by in-collective reconfigurations on a testbed with NVIDIA GPUs and find that these overheads are negligible, on the order of a few microseconds.

In summary, our main contributions are:

- Harvest, a systematic approach for navigating the trade-off between congestion, propagation delay, and reconfiguration delay in photonic interconnects for collective communication. Harvest synthesizes optimal photonic switching schedules by combining dynamic programming with topology optimization and applies to arbitrary collective algorithms.
- Structural insights into Recursive Doubling AllReduce that enable optimal schedule synthesis within polylogarithmic complexity, eliminating the need for mixed-integer optimization in this special case.
- An extensive evaluation using Astra-Sim, flow-level simulations, and hardware emulation on commercial GPUs, demonstrating significant performance improvements over static and reconfigure-every-step baselines.
- Public release of all artifacts at https://github.com/STyGIANet/Harvest, including the circuit switching implementation in Astra-Sim, to ensure reproducibility and support future research.

This work does not raise any ethical issues.

2 Background & Motivation

Unlike traditional datacenter applications, the collective operations among GPUs in AI/ML workloads result in staged and highly structured communication patterns. Among these collectives, AllReduce and All-to-All are especially prevalent [42, 43, 53, 58, 60, 61, 68, 81].

AllReduce, as the name suggests "reduces" (e.g., sum) the data each GPU holds and then distributes the results to all others. In All-to-All (Figure 1), each GPU j delivers a portion of the data it holds (block i) to another GPU (i). A wide range of algorithms exist for both of these collectives [15, 59, 64, 65]. Both primitives are bandwidth-intensive and latency-sensitive.

Limits of topology-aware collectives: Many prior works design collective algorithms for *specific* network topologies using the classic α–β cost model [16, 47, 66]. While these topology-aware algorithms can improve collective efficiency on fixed interconnects, they inherit the rigidity of the static networks on which they run. Multi-step collectives [59] repeatedly exchange data across different

[2]We generalize beyond the point-to-point communication model later in the paper (§3.3).

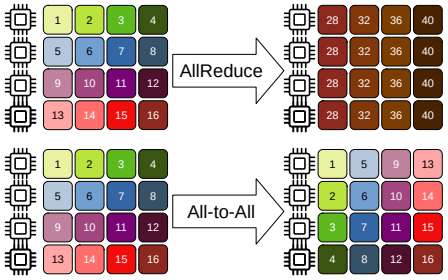


Figure 1: Collective communication primitives

pairs of communication partners. Under a fixed topology, some of these exchanges must traverse longer or congested paths, increasing both latency and bandwidth consumption [65]. As a result, static networks must provision for the worst-case demand over the entire lifetime of the collective, which often leads to underutilization when communication is sparse or staged. Techniques such as pipelining or mirroring can partially mitigate these effects [65] but they cannot fully overcome the limitations.

Throughput modeling and BvN decompositions: The maximum concurrent flow framework [67] is a standard tool for reasoning about network throughput and congestion [3, 6, 34, 51, 69]. It connects naturally to Birkhoff–von Neumann (BvN) decompositions, which express an aggregate traffic matrix as a convex combination of matchings [51]. This abstraction underlies many approaches for synthesizing circuit switching schedules in demand-aware networks [37, 56, 76]. However, BvN decompositions and traffic-matrix-based formulations inherently assume that all communication demand is available simultaneously. Collective communication violates this assumption: collectives generate and consume data in a strict sequence, and later communication steps cannot begin until earlier steps complete. As a result, static traffic-matrix decompositions fail to capture the temporal dependencies that are intrinsic to many collective algorithms.

Programmable but costly reconfiguration: Reconfigurable photonic interconnects enable the network topology to adapt to the communication pattern of each collective step and can potentially reduce congestion and improve throughput [7, 21, 40]. But this flexibility has a cost. The reconfiguration delay is high in photonic interconnects which can negate any gains if we are not careful [21]. Much of the existing literature either assumes that reconfiguration overheads are negligible or avoids reconfiguration altogether when they are not.

We advocate a more principled perspective that bridges the staged structure of collective algorithms, the limits imposed by network throughput, and the real cost of reconfiguration.

3 A Theory for Adaptive Scaleup Domains

We first describe our architecture (§3.1), and motivate the case for reconfiguration delay-aware circuit switching with an example (§3.2). We then revisit modeling the completion time of collectives (§3.3), revealing an optimization opportunity to account for interconnect reconfiguration delays (§3.4).

3.1 Architecture and Assumptions

Interconnect: We consider a scale-up domain with n GPUs, each equipped with an electrical-to-optical transceiver (e.g., TeraPhy [75]) of capacity c. All n transceivers connect to a photonic interconnect with n ports, which can establish direct optical paths between pairs of ports, thereby enabling GPU-to-GPU communication [72]. The interconnect is programmable and supports dynamic reconfiguration of optical paths on demand [40]. Either a central controller controls the interconnect or the interconnect is passive (transceivers can tune the wavelength of the emitted light). In the latter case, wavelength-selective switching within the photonic fabric establishes direct paths between ports without centralized control. In both designs, reconfiguring the interconnect incurs a non-negligible delay, denoted by α_r. Several photonic technologies incur reconfiguration delays that depend on the number of ports involved in the reconfiguration [21].

We assume the reconfiguration delay α_r is constant (e.g., based on the total port count). Our framework can be extended to capture port-dependent or topology-dependent reconfiguration delays. We assume that all GPUs reside within a single scale-up domain and have fast access to shared memory, as in modern systems such as DGX-class servers [32]. This enables GPUs to synchronize efficiently using a barrier before a collective step, perform reconfiguration synchronously if needed, and then proceed with communication. We assume collectives with n GPUs. We can also selectively apply our framework to subsets of GPUs (where we reconfigure a subset of ports when necessary).

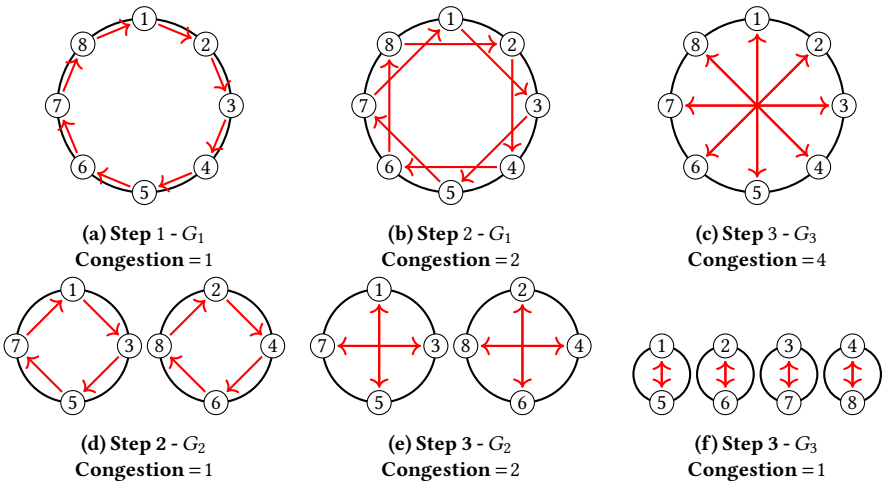
GPU forwarding: We assume that GPUs are equipped with an in-built router, similar to those used in Google's TPUs [33], and support cut-through forwarding at intermediate nodes. In particular, GPUs can begin forwarding data before the entire message has been received.

Communication steps: Throughout this paper, we adopt the standard notion of *steps* used in prior work on collective algorithms [18, 20, 59, 64, 65, 71]. A step denotes a communication phase during which each GPU exchanges data with a predetermined set of peers according to the collective algorithm. Communication within a step may involve multi-hop forwarding through intermediate GPUs, consistent with prior works [33, 65].

3.2 Example Walkthrough

We consider the Recursive Doubling [59, 64, 65] algorithm for AllReduce, which completes in a logarithmic number of steps. For a network with n nodes, Recursive Doubling proceeds in $\log_2(n)$ steps. In step i (starting from $i=1$), node j communicates with node $(j+2^{i-1}) \bmod n$, so the communication distance doubles at each step under a static topology.[3] We first illustrate Recursive Doubling on a static one-dimensional interconnect, where a single ring topology supports communication across all steps (Figures 2a, 2b, and 2c). As the algorithm progresses, multiple communication pairs overlap on shared links in later steps. In particular, congestion increases to 2 in step 2 and to 4 in step 3. Higher congestion reduces the effective

[3]Throughout this paper, we follow the cyclic variant of Recursive Doubling [38], which retains the same asymptotic properties as the pairwise-exchange formulation.



(a) Step 1 - G_1
Congestion = 1

(b) Step 2 - G_1
Congestion = 2

(c) Step 3 - G_3
Congestion = 4

(d) Step 2 - G_2
Congestion = 1

(e) Step 3 - G_2
Congestion = 2

(f) Step 3 - G_3
Congestion = 1

Figure 2: Various configurations of topology and communication patterns during Recursive Doubling AllReduce algorithm. Black lines indicate the physical topology, and the red arrows indicate the communication between nodes.

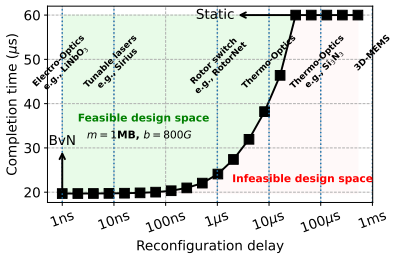


Figure 3: Reconfiguration delay-aware circuit switching schedules for Recursive Doubling reveal the full design spectrum between BvN and static topologies. The black curve denotes a lower bound on completion time beyond which no reconfiguration schedule can achieve further improvement.

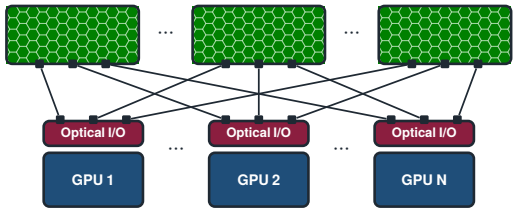


Figure 4: GPUs with on-chip optical I/O (with one or more transceivers) connect to a photonic interconnect that establishes direct optical paths between them.

bandwidth available to each flow, increasing per-step transfer time and, consequently, the overall collective completion time.

We can reconfigure the topology before step 2 to establish direct optical paths between the GPUs that communicate in this step (Figure 2d). At the final step, step 3, the interconnect faces a choice. It can either remain in the same topology, which results in congestion of 2 (Figure 2e), or reconfigure again to further reduce congestion (Figure 2f). The resulting reconfiguration options highlight a fundamental trade-off: reducing congestion comes at the cost of incurring reconfiguration delay.

Congestion is not the only factor that increases completion time. As communication distance grows, propagation delay also increases, further amplifying the impact of static or poorly chosen topologies (Figure 3). Birkhoff–von Neumann schedules (e.g., G_1, G_2, G_3) that reconfigure before every communication step eliminate congestion and achieve the lowest completion time when reconfiguration delays are small. As reconfiguration delays increase, static topologies become optimal, avoiding reconfiguration overhead at the cost of higher congestion and longer paths. Between these two extremes lies a broad design space in which carefully chosen, reconfiguration-aware schedules outperform both static interconnects and reconfigure-every-step baselines. Our goal is to expose this spectrum and provide a systematic framework for reasoning about *when* and *how* reconfiguration should be used to minimize collective completion time.

3.3 Modeling Collective Completion Time

To reason about the impact of topology reconfigurations on collective completion time, we revisit the classic α–β cost model and extend it to explicitly account for network-level effects, such as propagation delay, congestion, and reconfiguration delay. We model a collective communication algorithm that runs across n GPUs as a sequence of s communication steps. In each step i, the collective exchanges a fixed amount of data, m_i, between pairs of GPUs. We use a communication matrix $\mathcal{M}_i$ to represent this. An entry $\mathcal{M}_i(j,k) = 1$ indicates GPU j sends data to GPU k during step i (all other entries are zero). We describe a collective algorithm in this notation as a sequence $\langle \mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_s \rangle$ together with the data volumes $\langle m_1, m_2, ..., m_s \rangle$ associated with each step.

We use the *aggregate demand matrix* $\mathcal{M}$ to capture the total communication across all steps, where each entry $\mathcal{M}(j,k)$ denotes the total volume of data sent from GPU j to GPU k over the entire collective. We sum the stepwise communication matrices weighted by their corresponding data volumes to compute this matrix:

$$\mathcal{M} = m_1 \cdot \mathcal{M}_1 + m_2 \cdot \mathcal{M}_2 + ... + m_s \cdot \mathcal{M}_s. \quad (1)$$

Observation 1 (Relevance of BvN Decompositions)**.** *Collectives that proceed via a sequence of matchings naturally induce a BvN decomposition of their aggregate demand matrix.*

Point-to-point communication model: Most prior work designs collective communication algorithms under the point-to-point communication model [18, 59, 65, 71], where in each step every node sends to and receives from at most one other node. As a result, the communication matrix $\mathcal{M}_i$ in each step is a permutation matrix. Many well-known algorithms, including Ring, Recursive

Doubling [59], and swing [65][4], follow this abstraction. Under this model, Equation 1 corresponds to a *Birkhoff–von Neumann (BvN) decomposition* of $\mathcal{M}$ From this perspective, the steps of the collective algorithm correspond directly to the matchings in the decomposition, where each m_i represents the data volume the algorithm transferred during step i.

The converse does not hold. Not every BvN decomposition corresponds to a valid collective algorithm. BvN decompositions fail to capture the *temporal structure* inherent in collective communication. In practical algorithms, the ordering of communication steps matters, and we cannot arbitrarily rearrange steps. The data exchanged in step i is often generated as a consequence of the computation or communication performed in step $i-1$, which induces a strict sequence of dependencies.

One-to-many communication model: Recent work proposes multi-port AllReduce algorithms [64, 65] to better utilize network bandwidth when nodes are equipped with multiple links, such as in multi-dimensional Torus networks [33]. Under this model, the communication matrix $\mathcal{M}_i$ is not necessarily a permutation matrix and is often composed of multiple permutations, for example one per dimension. We do not further decompose these matrices, and instead focus on the dependencies between successive matrices $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$. We assume that each port participates in at most one point-to-point communication per step: the number of ports upper bounds the sum of each row and column of each $\mathcal{M}_i$. Thus, we can still use Equation 1 to express the communication model with this generalization

All-to-All communication model: In many All-to-All implementations, multiple send and receive operations are grouped into a single logical step. Here, we can represent the communication with a single bulk demand, through an aggregate demand matrix. We can use a BvN decomposition to decompose the aggregate matrix and represent the communication via Equation 1 as a sequence of s point-to-point steps. We sort the coefficients of the resulting decomposition such that m_i is the i^{th} largest coefficient, and $\mathcal{M}_i$ the corresponding point-to-point communication performed in the i^{th} step.

The matrix decompositions induced by collective algorithms, as we show next, reveal a useful connection to both network throughput and the classic α–β cost model.

Consider a graph $G_i=(V,E_i)$, where V is the set of n GPUs and E_i represents the photonic links between them during step i of the collective. We can express the total completion time $t_c(1,s)$ of the collective communication algorithm from step 1 through step s (inclusive) as:

$$t_c(1,s)=DCT(m_1\cdot\mathcal{M}_1,G_1)+DCT(m_2\cdot\mathcal{M}_2,G_2) +\ldots+DCT(m_s\cdot\mathcal{M}_s,G_s)+\sum_{i=1}^{s}\underbrace{\alpha_r\cdot\mathbb{I}(E_i\neq E_{i-1})}_{\text{reconfiguration delay}} \quad (2)$$

where $DCT(m_i\cdot\mathcal{M}_i,\ G)$ denotes the *demand completion time* of step i, corresponding to a data volume m_i and communication pattern $\mathcal{M}_i$, served by the underlying topology G_i. Each step incurs additional α_r delay if the topology differs from the previous step, representing a reconfiguration event.

$DCT(m_i\cdot\mathcal{M}_i,G_i)$ depends on the structure and capacity of the underlying graph G_i. We define the *maximum concurrent flow* $\theta(G_i,\mathcal{M}_i)$ as the largest fraction of the communication matrix $\mathcal{M}_i$[5] the network can route simultaneously without exceeding any link capacities. $\theta(G_i,\mathcal{M}_i)$ is the achievable throughput for that step's communication pattern. This implies we can write demand completion time as:

$$DCT(m_i\cdot\mathcal{M}_i,G_i)=\frac{m_i}{c}\cdot\frac{1}{\theta(G_i,\mathcal{M}_i)},$$

where c is the link capacity. Here, $\frac{m_i}{c}$ represents the ideal transmission time assuming full throughput, while the factor $\frac{1}{\theta(G_i,\mathcal{M}_i)}$ accounts for congestion. By definition of the maximum concurrent flow, the effective capacity available for this communication is $c\cdot\theta(G_i,\mathcal{M}_i)$. Therefore, the actual transmission time scales inversely with the achievable throughput.

Each communication step i incurs a fixed overhead α, which captures startup latencies such as data preparation. In each step, we also have to account for the latency of the longest route that went through the most congested links in each step. This latency is given by $\delta\cdot\ell_i(G_i)$, where δ is the per-link propagation delay and $\ell_i(G_i)$ is the length of that longest path. The latency term is often neglected and absorbed into the constant α. If the network offers capacity c per node, we define $\beta=\frac{1}{c}$. We can write demand completion time for step-i as:

$$DCT(m_i\mathcal{M}_i,G_i)=\underbrace{\alpha+\delta\cdot\ell_i(G_i)}_{\text{latency factor}}+\overbrace{\beta}^{bandwidth\ factor}\ m_i\underbrace{\frac{1}{\theta(G_i,\mathcal{M}_i)}}_{congestion\ factor} \quad (3)$$

We can now express the total completion time of the collective for all steps s as:

$$t_c(1,s)=\sum_{i=1}^{s}\left(\alpha+\delta\cdot\ell_i(G_i)+\frac{\beta\cdot m_i}{\theta(G_i,\mathcal{M}_i)}+\alpha_r\cdot\mathbb{I}(E_i\neq E_{i-1})\right) \quad (4)$$

Observation 2 (Collective Completion Time as α–β Cost)**.** *The classic α–β cost model emerges naturally when collective completion time is expressed in terms of network latency, bandwidth, propagation delay, and congestion, where congestion is captured by the inverse of maximum concurrent flow. This perspective grounds the model in network throughput and exposes its dependence on both the interconnect topology and the staged structure of the collective.*

While the α–β model is widely used in practice, network throughput, propagation delay, and congestion are rarely made explicit in its formulation. A few exceptions relate congestion to communication distance or to the number of messages traversing a link in structured topologies [18, 64, 65], but these approaches are typically limited to specific communication patterns or architectures and often assume unsplittable flows. TE-CCL [47] explored the relationship between the α–β model and multi-commodity flow in the context of collective algorithm synthesis by mapping collective communication patterns to demand matrices and interpreting the cost model through that formulation. In a similar vein, our approach links the α–β model to network throughput via maximum concurrent flow in optical interconnect configurations. As detailed in Appendix C.3, this perspective generalizes prior demand-aware

[4]The Swing algorithm also admits a multi-port variant, which departs from the strict point-to-point model and is captured by our one-to-many model.

[5]Here, we consider that $\mathcal{M}_i$ is scaled proportionally to the node capacity.

network formulations to yield a comprehensive performance model accounting for communication structure, congestion, propagation delay, and network topology. Crucially, our formulation applies to arbitrary topologies, making these insights broadly applicable beyond structured or hierarchical networks.

3.4 An Optimization Opportunity

For circuit-switched photonic interconnects, we observe that collective communication time can be reduced by establishing direct, high-bandwidth optical paths that eliminate congestion and exactly match the communication pattern $\mathcal{M}_i$ for each step i. However, realizing these paths requires reconfiguring the interconnect, which introduces a reconfiguration delay α_r. This creates a clear trade-off: dynamic reconfiguration eliminates congestion and maximizes throughput at the expense of added latency, whereas a static topology avoids the reconfiguration penalty but potentially suffers from more network congestion.

This tension opens up an opportunity for optimization: given a collective communication schedule, how should we schedule interconnect reconfigurations to minimize the total completion time for any given collective? An effective circuit switching schedule must strike a balance, where we reconfigure only in steps when the throughput gain outweighs the cost.

This paper focuses on the *topology synthesis* problem:

Input: We are given a predefined collective communication algorithm i.e., a communication schedule. The schedule specifies, the amount of data each source-destination pair exchanges in each step of the collective. The input also includes topology constraints, such as link bandwidth, node degree (number of links), and fixed propagation latencies.

Output: The output is a topology reconfiguration schedule that specifies a network topology for each communication step of the collective. When the topology differs between consecutive steps we incur a reconfiguration delay.

A related but orthogonal line of work studies the *collective synthesis* problem, in which the input is a fixed topology and the output is a communication schedule [16, 47, 66]. This setting is the converse of the problem we solve.

4 Harvest

At its core, topology synthesis consists of two tightly coupled components. First, given a contiguous range of collective steps a through b, we must decide *how* to reconfigure the interconnect i.e., find a static topology that minimizes the completion time of those steps when no reconfiguration is allowed within the range. Second, we must partition the full sequence of collective steps into such ranges to determine *when* to reconfigure. This partitioning induces a recurrence that jointly determines both the reconfiguration events and the corresponding topologies.

4.1 Subproblem

Given a collective communication algorithm with s steps, where each step i is characterized by a communication pattern $\mathcal{M}_i$ and data volume m_i, our subproblem is to find an optimal topology $G_{a,b}$ that minimizes the completion time of steps a through b (inclusive), without any reconfigurations during this interval.

$$G_{a,b} = \arg\min_{G \in \mathcal{G}} \sum_{i=a}^{b} DCT(m_i \cdot \mathcal{M}_i, G) \tag{5}$$

$$t_c(a,b) = \sum_{i=a}^{b} DCT(m_i \cdot \mathcal{M}_i, G_{a,b}) \tag{6}$$

Here, $\mathcal{G}$ denotes the set of all feasible topologies that satisfy the per-node degree constraints, and $t_c(a,b)$ denotes the completion time for steps a through b. We use Equation 3 to compute the demand completion time $DCT(m_i \cdot \mathcal{M}_i, G_{a,b})$.

We solve this subproblem using a Mixed-Integer Second-Order Conic Program (MISOCP). We introduce integer decision variables $x_{u,v}$ that denote the number of directed edges from node u to node v. These variables are subject to degree constraints $\sum_u x_{u,v} \le d$ and $\sum_v x_{u,v} \le d$, and ensure each node has degree at most d. The objective is to minimize the total demand completion time for the sequence of communication matrices $\langle \mathcal{M}_a, \ldots, \mathcal{M}_b \rangle$.

The constraints follow a standard maximum concurrent flow formulation, with the key distinction that edge capacities are decision variables rather than fixed inputs. The objective minimizes a weighted sum of the reciprocals of the concurrent flow values across steps (Equation 3). This yields a second-order conic optimization problem with integer variables. Unlike a classical concurrent flow formulation, the presence of sequential dependencies across steps induces the conic structure. We present the complete formulation in Appendix A.

4.2 Recurrence and Dynamic Programming

At a high level, our dynamic programming approach partitions the collective communication algorithm's steps into contiguous intervals, separated by k reconfiguration events. Central to our approach are three variables:

- $DP[a][t]$ denotes the optimal completion time with t reconfigurations for the sequence of steps starting at a, until the end. We do not include the reconfiguration delays at this point. Our dynamic program takes a fixed number of reconfigurations as input, and finds a corresponding optimal schedule. We later find the best number of reconfigurations (§4.3).
- $next[a][t]$ stores the next step after a at which a reconfiguration occurs in the optimal schedule.
- $\text{topo}[a][t]$ stores the optimal topology for the steps a through $next[a][t]$.

Lemma 1 (Recurrence)**.** *For any starting step $a \in \{1,\ldots,s\}$ and number of reconfigurations $k \ge 1$, the optimal completion time is*

$$DP[a][k] = \min_{a < b \le s+1} \left(t_c(a, b-1) + DP[b][k-1] \right) \tag{7}$$

where $t_c(\cdot,\cdot)$ is the completion time for steps $a-b$, given by Eq 6.

Proof. We prove the claim by induction on the number of reconfigurations $k \ge 0$. For the base case $k=0$, no reconfigurations are allowed and the schedule consists of a single contiguous interval covering steps a through s. The collective completion time is therefore $t_c(a,s)$ by definition, and hence $DP[a][0] = t_c(a,s)$. Now assume the claim holds for all numbers of reconfigurations $t < k$ and for all starting steps. Fix $k \ge 1$ and a starting step a. Consider any feasible schedule with exactly k reconfigurations over steps a through s. Let b denote the step at which the first reconfiguration

occurs, where $a < b \le s$. This choice partitions the schedule into two segments: steps a through $b-1$, executed without reconfiguration, and steps b through s, executed with the remaining $k-1$ reconfigurations. The completion time of the first segment is $t_c(a,b-1)$. By the inductive hypothesis, the minimum completion time achievable for steps b through s with $k-1$ reconfigurations is $DP[b][k-1]$. Therefore, the total completion time of any such schedule is at least $t_c(a,b-1) + DP[b][k-1]$. Minimizing over all valid choices of b yields the recurrence defining $DP[a][k]$, which completes the proof. Thus any b yields total cost $t_c(a,b-1) + DP[b][k-1]$, and minimizing over b gives Equation 7. □

The optimal schedule with exactly k reconfigurations is $DP[1][k]$. Algorithm 1 unrolls Equation 7 for $DP[1][k]$ within a for loop. At each partition, we record $next[a][t]$, which then allows us to reconstruct the optimal schedule.

4.3 Synthesizing Switching Schedules

Lemma 1 gives the optimal topology sequence for a fixed number of reconfigurations k, excluding reconfiguration delays. It remains to find the number of reconfigurations for which the reconfiguration schedule minimizes the overall completion time, including reconfiguration delays. Taking the minimum over $k = 0,\ldots, s$ and adding the reconfiguration delay $k \cdot \alpha_r$ corresponding to k reconfigurations, yields the delay-aware schedule that minimizes overall completion time.

Theorem 1 (Optimality of the schedule). *Fix the number of steps s. For any $k \in \{0,\ldots,s\}$, the schedule reconstructed from $next[\cdot][\cdot]$ that attains $DP[1][k]$ is optimal among all schedules with exactly k reconfigurations. Moreover, $\operatorname{argmin}_{k\in\{0,\ldots,s\}} \left(DP[1][k] + k\,\alpha_r\right)$ is optimal among all schedules that account for reconfiguration delay.*

Proof. By Lemma 1 and induction on k, $DP[a][k]$ equals the optimal completion time for steps a through s with exactly k reconfigurations; in particular $DP[1][k]$ is optimal for the full instance, and $next$ reconstructs an optimal switching schedule. Since reconfiguration delay is additive and depends only on k, minimizing $DP[1][k] + k\alpha_r$ over $k \in \{0,\ldots,s\}$ yields the delay-aware optimum. □

Overall, our framework captures the fundamental trade-off between reconfiguration delay and congestion in adaptive photonic interconnects. It provides a systematic way to synthesize circuit switching schedules for collective communication, balancing the benefits of reconfiguration against its costs. Notably, the synthesis is aware of data volume in each step, reconfiguration delay, propagation delay, and the underlying network throughput.

4.4 Discussion

Our synthesis framework combines a dynamic program over step intervals with a topology optimization subproblem to compute the optimal reconfiguration. The dynamic program has polynomial complexity in the number of collective steps, while the dominant computational cost arises from solving the topology subproblem via a MISOCP. The overall complexity is $O(s^4 \cdot g)$, where s denotes the number of communication steps and g captures the cost of solving a single MISOCP instance.

In practice, the number of steps s is modest for many widely used collective algorithms. For example, recursive doubling and Swing have $s = \Theta(\log_2 n)$ steps. We can synthesize the schedule offline and cache it to avoid computations at runtime when messages arrive. As a result, the synthesis cost does not lie on the critical

Algorithm 1: Harvest

Input: Setup latency α, bandwidth b, $\beta = \frac{1}{b}$, propagation delay δ, reconfiguration delay α_r, sequence of steps and corresponding data volumes $\langle m_1 \cdot \mathcal{M}_1, m_2 \cdot \mathcal{M}_2, \ldots, m_s \cdot \mathcal{M}_s \rangle$.
Output: Reconfiguration schedule

Function `CompletionTime`(a,b):
  Find $G_{a,b}$ ▷ Equation 5
  Find $t_c(a,b)$ ▷ Equation 6
  return $(G_{a,b}, t_c(a,b))$
To find the optimal reconfigurations for a given number of reconfigurations k.
Function `SynthesizeSchedule`(s,k):
  Initialize DP$[a][t] \leftarrow \infty$, next$[a][t] \leftarrow \varnothing$, topo$[a][t] \leftarrow \varnothing$ for $a \in \{1,\ldots,s+2\}, t \in \{0,\ldots,k\}$
  for $a \leftarrow 1$ **to** s **do**
    DP$[a][0] \leftarrow$ `CompletionTime`(a,s)
  for $t \leftarrow 0$ **to** k **do**
    DP$[s+1][k] \leftarrow 0$
  for $t \leftarrow 1$ **to** k **do**
    for $a \leftarrow 1$ **to** s **do**
      best $\leftarrow \infty$; arg $\leftarrow \varnothing$; graph $\leftarrow \varnothing$
      for $b \leftarrow a+1$ **to** $s+1$ **do**
        $(G,v) \leftarrow$ `CompletionTime`$(a,b-1)$+DP$[b][t-1]$ ▷ Recurrence (Lemma 1)
        if $v <$ best **then**
          best $\leftarrow v$; arg $\leftarrow b$; graph $\leftarrow G$
      DP$[a][t] \leftarrow$ best; next$[a][t] \leftarrow$ arg;
      topo$[a][t] \leftarrow graph$
  $r \leftarrow []$; $a \leftarrow 1$; $t \leftarrow k$
  while $t > 0$ **do**
    $b \leftarrow$ next$[a][t]$; **if** $b = \varnothing$ **then break**
    $G \leftarrow$ topo[a][t]
    if $b \le s$ **then**
      append (G,b) to r
    $a \leftarrow b$; $t \leftarrow t-1$
  return (DP$[1][k]$,r)
    ▷ Completion time w/o reconfiguration delays, schedule
To find the optimal reconfigurations across all number of reconfigurations k.
bestCost $\leftarrow \infty$; bestR $\leftarrow []$
for $k \leftarrow 0$ **to** s **do**
  $(C,R) \leftarrow$ `SynthesizeSchedule`(s,k)
    ▷ Schedule with exactly k reconfigurations
  $C \leftarrow C + k \cdot \alpha_r$
  if $C <$ bestCost **then**
    bestCost $\leftarrow C$; bestR $\leftarrow R$
return bestR ▷ Optimal schedule (Theorem 1)

path of collective execution. We discuss the practical aspects of the computation cost in §6.

The primary scalability challenge lies in the topology optimization subproblem. We can solve our MISOCP for interconnect sizes of up to 64 GPUS, but we need to reduce the effective topology search space further to scale to larger interconnects. This observation motivates us to exploit the structure in collective communication patterns and to restrict our attention to a small set of candidate topologies that are likely to be optimal over contiguous intervals of steps.

It is interesting to understand the precise conditions under which restricted topology classes suffice for arbitrary collectives in future work. In the next section, we address the following questions, which guide the design of practical synthesis algorithms that balance optimality and efficiency:

(Q1) *To what extent can we reduce the topology search space without compromising optimality?*

(Q2) *Can we synthesize optimal schedules within polynomial-time complexity in the number of nodes?*

5 Optimal Photonic Switching Schedules for Recursive Doubling AllReduce

Building on our observations in §3 and the synthesis technique in §4, our goal is to efficiently synthesize an *optimal* schedule. Our design centers on the Recursive Doubling algorithm for AllReduce. We make two new observations about Recursive Doubling that, as we later show in this section, enable synthesis of schedules within polylogarithmic-time complexity.

5.1 Observations on Recursive Doubling

We make two simple yet powerful observations about the Recursive Doubling algorithm, which directly guide the synthesis. We consider the cyclic version of Recursive Doubling [38] that retains the same properties as the pairwise exchange version. A node u communicates with node $u+2^{i-1}$, and transmits $\frac{m}{2^i}$ chunk size in step i, during the Reduce-Scatter phase. The communication pattern reverses in the AllGather phase.

We first check if we need additional reconfigurations to preserve reachability when we establish direct links based on the communication step i. This, in turn, helps find topologies that can serve multiple steps without frequent reconfiguration.

Observation 3 (Connectivity)**.** *The topology that establishes direct links between GPUs according to the communication pattern of step i also preserves connectivity for all subsequent steps $j \geq i$ in Recursive Doubling.*

In Recursive Doubling, step i requires each node a to communicate with $a+2^{i-1}$ (with steps indexed from 1). Establishing these direct links does not break connectivity for any later step $j \geq i$. In step j, node a must communicate with $a+2^{j-1}$, which lies at distance 2^{j-i} in this topology. This is because a connects to $a+1$, which in turn connects through a chain of nodes $a+2, a+4, \ldots, a+2^{j-i}$, which ensures connectivity. The proof follows.

Next, we aim to find a single topology that minimizes the completion time for any sequence of steps a through b in Recursive Doubling. This enables us to restrict the search space to a specific class of topologies.

Observation 4 (Optimal Topology)**.** *For any interval of steps a through b in Recursive Doubling, the topology that minimizes completion time is the one that establishes direct links between GPUs according to the communication pattern of step a.*

We can characterize the interval of steps a through b as: in step a, each node u communicates with $u+2^{a-1}$ (with steps indexed from 1). The sequence of minimum path lengths for steps a through b is $\langle 1,\ldots,2^{b-a}\rangle$. Likewise, the minimum congestion we incur is at least $\langle 1,\ldots,2^{b-a}\rangle$, which corresponds to each step from a through b. The topology that establishes direct connections according to step a's communication pattern, i.e., a direct link between u and $u+2^{a-1}$, achieves exactly this minimum sum of path lengths and congestion. Specifically, for step a, the communication distance is reduced to 1. For step $a+1$, node u communicates with $u+2^a$, which is at distance 2: in our chosen topology, u connects to $u+2^{a-1}$, which in turn connects to $u+2^{a-1}+2^{a-1}=u+2^a$. The proof follows for both path lengths and congestion.

We can now express the completion time of steps $a-b$ where the communication pattern matches that of step a as:

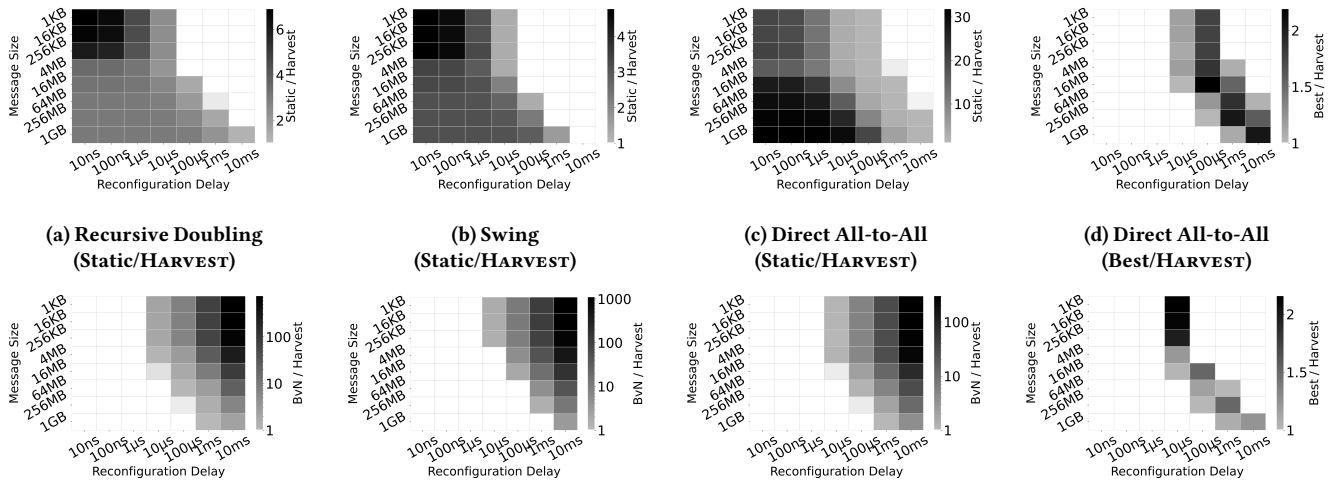
$$\begin{aligned} t_c(a,b) &= \sum_{i=a}^{b} DCT(m_i \cdot \mathcal{M}_i, G_{a,b}) \\ &= \alpha\cdot(b-a+1)+\delta\cdot\sum_{i=a}^{b}2^{i-a}+\beta\cdot m\cdot\sum_{i=a}^{b}\frac{1}{2^i}\cdot 2^{i-a} \\ &= \alpha\cdot(b-a+1)+\delta\cdot\left(2^{(b-a+1)}-1\right)+\beta\cdot m\cdot\frac{b-a+1}{2^a} \end{aligned} \tag{8}$$

Observation 3 shows certain topologies can preserve connectivity without further reconfiguration. This removes forward dependencies in reconfiguration decisions. Observation 4 finds the optimal topology for any range of steps a through b in $O(1)$ time, without additional work. Together, they hugely simplify the topology search space to just $O(1)$, leaving the key question: when should we reconfigure?

Building on Observations 3 and 4, we synthesize optimal schedules for Recursive Doubling using the dynamic programming approach we described in §4 (Algorithm 1). In particular, the function `CompletionTime(a,b)` simplifies i.e., finding $G_{a,b}$ is $O(1)$. The rest of the procedure remains the same as earlier: *(i)* the function `SynthesizeSchedule(s,k)` synthesizes the optimal schedule corresponding to a given k number of reconfigurations; *(ii)* we iterate from 0 to $s = \log_2(n)$ number of reconfigurations, synthesize the schedule for each, and return the best global schedule. Given the logarithmic number of steps in Recursive Doubling AllReduce, synthesizing optimal switching schedules reduces to polylogarithmic complexity of $O((\log n)^4)$.

Our observations are not limited to Recursive Doubling and extend to other collectives:

- Hypercube All-to-All
- Bruck's AllGather, Reduce-Scatter, and AllReduce
- Bruck's All-to-All
- Binomial tree Broadcast



(a) Recursive Doubling (Static/Harvest) **(b) Swing (Static/Harvest)** **(c) Direct All-to-All (Static/Harvest)** **(d) Direct All-to-All (Best/Harvest)**

(e) Recursive Doubling (BvN/Harvest) **(f) Swing (BvN/Harvest)** **(g) Direct All-to-All (BvN/Harvest)** **(h) Recursive Doubling (Best/Harvest)**

Figure 5: [Simulations] We show Harvest speeds up collective completion time compared to BvN-based schedules and static topologies with link bandwidth=800Gbps, α=500ns, and δ=500ns. Best denotes the best out of static and BvN

- Trivance [35]
- A class of algorithms with a constant distance multiplier across successive steps (Appendix B)

We present formal proofs for each of these algorithms in Appendix B. We believe similar reductions are possible for other collectives and leave this as an interesting direction for future work.

6 Evaluation

We evaluate the schedules Harvest synthesizes and their completion time across several collective communication algorithms. We compare their performance against BvN schedules and static topologies. Our evaluation spans simulation, hardware emulation, and numerical optimization, capturing both performance and system-level effects.

6.1 Setup

Network: We consider networks with 8 to 64 GPUs, representative of typical scale-up domains. Each GPU has d ports, where d ranges from 2 (e.g., a 1-D ring) to 6 (e.g., a 3-D torus). Unless stated otherwise, we set the per-port bandwidth to 800Gbps for simulations and 100Gbps for our hardware emulation experiments. We vary the setup latency α, link propagation delay δ, and the interconnect reconfiguration delay α_r over a wide range, from 10ns (e.g., tunable lasers [9]) to 10ms (e.g., 3-D MEMS [55]). This range captures diverse photonic switching technologies and allows us to identify regimes in which reconfiguration is beneficial.

Baselines: We compare Harvest against two representative baselines: *(i)* a *static* topology that remains fixed throughout the collective and *(ii) BvN schedules* that reconfigure the topology at every communication step to directly connect the communicating GPU pairs [7, 63]. For static topologies, we consider rings, 2D and 3D tori, and generalized Kautz graphs [80]. Together, these baselines capture the two extremes of the design space: interconnects that never reconfigure, and those that reconfigure at every step.

Collective algorithms: The input to Harvest and other baselines is the workload which consists of standard collective algorithms. For AllReduce, we evaluate Recursive Doubling [59], Swing [65] (which is equivalent to Bine Butterfly [20] in terms of communication pattern), and Bruck's concatenation algorithm [15]. For All-to-All, we evaluate the total exchange or transpose operation (direct All-to-All) as well as Bruck's All-to-All (index) algorithm [15]. For broadcast, we evaluate the binomial tree and binary tree algorithms.

Simulations: We perform packet-level simulations using Astra-Sim [62, 77], which we extend to support circuit-switched interconnects. The simulator now accepts a topology reconfiguration schedule as input and dynamically reconfigures the topology according to it during collective operations.

Hardware emulation: We would need specialized hardware to directly validate Harvest across a broad range of photonic switching technologies that have varying reconfiguration delays — this is costly. Instead, we emulate a reconfigurable photonic interconnect on an 8-GPU testbed. The GPUs are connected in a ring topology via 8 BlueField-3 NICs equipped with 100Gbps optical transceivers. GPU–NIC communication uses GPUDirect RDMA over PCIe, and we ensure that the available PCIe bandwidth exceeds the NIC I/O bandwidth to eliminate PCIe bottlenecks. We implement NIC–NIC communication and routing via the BlueField-3 eSwitch, with flow steering offloaded to hardware.

We use NCCL to execute collective operations step by step, and measure the runtime of each step until a reconfiguration is required. At that point, we pause execution, update the interconnect configuration, and resume the remaining steps. We compute the total completion time as the sum of the step-wise runtimes plus a fixed physical reconfiguration penalty. This methodology is practical, cost-effective, and yields representative results.

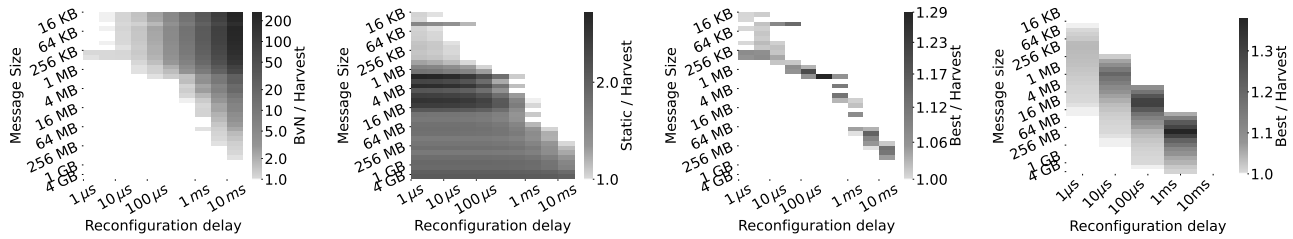


Figure 6: [Hardware emulation] Heatmaps showing the speedup in collective completion time achieved by Harvest relative to (a) BvN-based schedules, (b) a static ring, (c) the best of BvN and static, and (d) the speedup estimated by synthesis, which closely matches the measured speedups observed in hardware emulation.

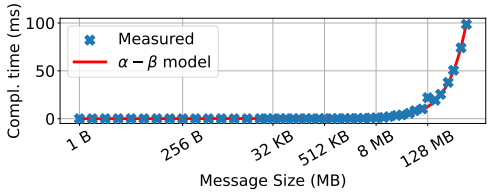


Figure 7: [Hardware emulation] Correlation between the α–β cost and measured completion times across different message sizes for one-hop communication.

Numerical evaluation: We implement the synthesis component of Harvest in C++ and use Gurobi to solve the subproblems in each dynamic program. We describe the optimization formulation in more detail in § A. For a range of network sizes and topologies, primarily multi-dimensional topologies, we synthesize reconfiguration schedules and report the corresponding collective completion times produced by the optimization. Moreover, we compare the trends in the synthesized schedules with the results obtained from our test-bed experiments.

Due to space constraints, additional results appear in § D.

6.2 Results

When does Harvest outperform static topologies?

We observe Harvest consistently outperforms static topologies when reconfiguration delay is low (Figures 5a, 5b, and 5c). We use packet-level simulations and a one-dimensional topology. Harvest speeds up Recursive Doubling by 6.4×, Swing by 4.7×, and All-to-All by 20× for small message (1 – 256KB) and reconfiguration delays $< 1\mu$s. This is because Harvest reconfigures the topology to shorten the long-distance steps which in turn reduces both congestion and propagation delays (these dominate small transfers).

As reconfiguration delays increase, the benefits of reconfiguration diminish for smaller message sizes, where Harvest naturally falls back to static schedules. In contrast, for larger messages the gains from reconfiguration persist even at higher delays. For example, with a 1GB message size, Harvest achieves up to 3.0× speedup over Recursive Doubling, 3.1× over Swing, and up to 30× over direct All-to-All, even with a 10μs reconfiguration delay. These results show that Harvest effectively balances reconfiguration overheads against congestion costs, selectively reconfiguring only when the performance benefits outweigh the overhead.

When does Harvest outperform BvN-based schedules?

Even though it is useful to reconfigure the topology, if we do so in each step we may inflate the completion time (when reconfiguration delays are non-negligible). Harvest consistently outperforms BvN-based schedules at higher reconfiguration delays (Figures 5e, 5f, and 5g).

The performance gap is most pronounced for small message sizes, where Harvest strategically limits the number of reconfigurations to reduce overhead. For instance, with a 100μs reconfiguration delay and message sizes between 1KB and 256KB, Harvest achieves on average around 7.3× speedup over BvN schedules for Recursive Doubling, 10× for Swing, and 5.3× for personalized All-to-All. As message sizes increase, the benefits of reconfiguration become more pronounced, and Harvest adapts toward BvN-like schedules. Notably, even at moderate reconfiguration delays between 10μs and 100μs, Harvest outperforms BvN schedules by up to 3.0× for 4MB messages with Recursive Doubling, and up to 4.8× with personalized All-to-All.

Overall, these results highlight the importance of carefully balancing the benefits of reconfiguration against its costs, rather than reconfiguring indiscriminately at every step.

When does Harvest outperform other schedules?

The natural trade-off between reconfiguration delay and congestion means there is an intermediate regime in which Harvest outperforms both static and BvN schedules. Harvest achieves the best performance in a transitional regime where it selectively reconfigures only a subset of communication steps (Figures 5d and 5h). For example, in personalized All-to-All (Figure 5d), Harvest outperforms both static and BvN schedules for message sizes up to 16MB when reconfiguration delays is in [10,100]μs. For larger reconfiguration delays, this regime moves towards larger messages. We observe a similar trend for Recursive Doubling. These results highlight the optimal strategy is neither to avoid reconfiguration entirely nor to reconfigure at every step, but to carefully choose a subset of reconfigurations that balances their benefits against their costs.

Does hardware reflect the performance trends revealed by our synthesis framework and cost model?

We validate the performance trends our synthesis framework predicts with hardware emulations. The results closely match those we saw in simulations (Figures 6a, 6b, and 6c). Figure 6a reports the completion-time ratio between BvN schedules and Harvest for Recursive Doubling — it confirms Harvest outperforms BvN schedules when reconfiguration delays are high. Harvest also outperforms a static ring (Figure 6b) topology across a wide range of the space (3× speedup). We also show there exists a transitional regime in which Harvest outperforms both (Figure 6c).

We parameterize our cost model based on measurements from the testbed to further validate our results. We first estimate the α–β parameters: we measure the single-hop GPU-to-GPU communication as a function of message size (Figure 7). We find $\alpha = 30.32\mu$s and $\beta = 85.11$Gbps. We then run our synthesizer to compute the completion time it estimates across message sizes and reconfiguration delays. We find the transitional regime and performance trends; as well as the speedup values closely match hardware (Figure 6c).

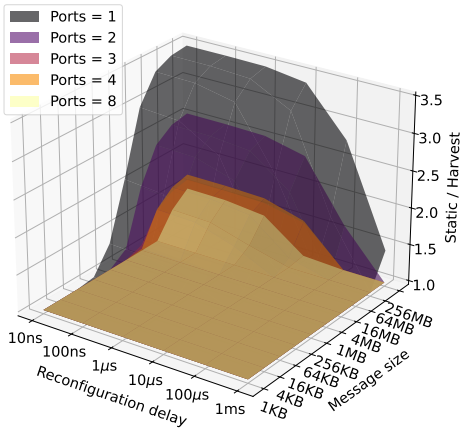
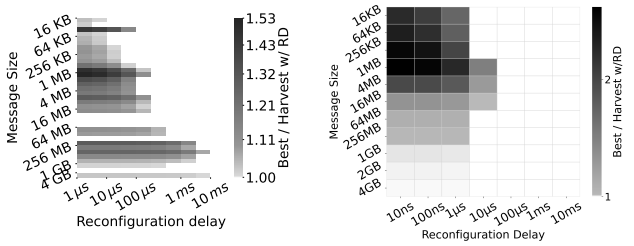


Figure 8: [Numerical optimization] All-to-All with Harvest compared to static low-diameter topology with optimal flow schedule [15, 80].

Is Harvest multi-port, multi-dimension compatible?

We compare Swing (2-D and 3-D torus topologies) with static, BvN and Harvest schedules (Figure 14). We consider a multidimensional extension of Swing with mirroring, which exploits all available ports simultaneously to maximize link utilization. Numerical evaluations using our synthesizer reveal trends that are consistent with those observed in one-dimensional simulations and hardware emulation. Across a range of 2-D and 3-D torus configurations with 64 GPUs, Harvest consistently outperforms all schedules. The regimes in which Harvest provides the largest gains are dictated by the trade-off between reconfiguration delay and congestion overhead as a function of message size. We observe similar trends for other multi-port collectives, including Bruck's index and concatenation algorithms, as well as Binomial Tree and Binary Tree broadcast (Figure 15). All-to-All communication in multi-port settings is different



(a) Emulation for AllGather **(b) Simulation for AllReduce**

Figure 9: When Recursive Doubling (RD) with Harvest outperforms best of Ring and RD on a static ring topology. White space indicates a speedup[5] less than or equal to 1×.

(Figure 8): the performance advantage over static topologies diminishes as the number of ports increases. This is expected, since All-to-All communication benefits from low average shortest-path lengths, which decrease significantly as the topology degree increases and the network diameter shrinks, reducing the need for reconfiguration.

When does Harvest outperform Ring collectives?

Naturally, for ring-based collectives, Observation 3 implies that the topology matching the communication pattern of any step i, which is a ring, preserves connectivity for all subsequent steps. Likewise, Observation 4 implies that the optimal topology for any range of steps a through b remains a ring. Consequently, the optimal schedule for ring algorithms is a static ring topology with no further reconfigurations. The tradeoff is that ring algorithms require a larger number of communication steps, resulting in a higher latency cost (α), whereas algorithms such as Recursive Doubling and Bruck's communication patterns complete in fewer steps at the cost of higher congestion ($\frac{1}{\theta}$) and potentially additional reconfiguration overhead (α_r).

Figure 9a (hardware emulation) compares the performance of Ring and Recursive Doubling (RD) AllGather algorithm executed on a static ring topology against RD AllGather executed using the optimal schedule synthesized by Harvest. The physical topology is a ring (unless reconfigured) and running a ring algorithm incurs no bandwidth sharing. Our observations reveal that ring algorithm performs favorably against RD with Harvest when reconfiguration costs dominate and when the message size is small, whereas RD with Harvest benefits from substantially fewer communication steps when reconfiguration overhead is sufficiently small. While results are discussed in more detail in Appendix C.2, the takeway is that the emulation trend shows the fundamental tradeoff between communication steps, congestion, and topology reconfiguration overhead.

Figure 9b shows our simulation results comparing Recursive Doubling (RD) with Harvest schedules, the ring algorithm on a static ring topology, and RD on a static ring topology. As an example, we consider $n = 64$ nodes, setup latency $\alpha = 500$ns, propagation delay $\delta =$ 50ns (e.g., in a scaleup domain), each GPU equipped with one link of

[5] All heatmaps (except Figure 9) use whitespace to indicate a speedup of exactly 1×. The representation in Figure 9 differs because it compares two different collective algorithms, namely Ring and Recursive Doubling with Harvest-derived topology schedules. When optimizing the ring algorithm, Harvest correctly identifies the static ring topology as optimal. Consequently, no additional speedup is obtained from topology reconfiguration, and the Ring algorithm remains at exactly 1× across the entire spectrum. We omit this plot for brevity.

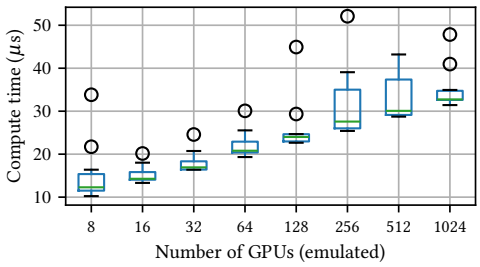


Figure 10: DP Compute time

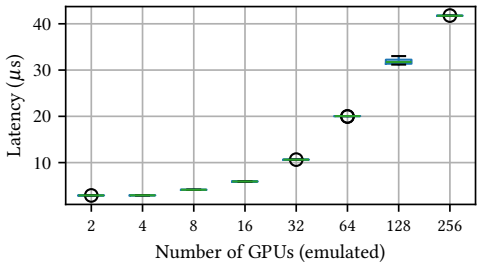


Figure 11: Synchronization latency

bandwidth $b = 800$Gbps. Figure 16, 17 in Appendix D show our results for $\alpha = 500$ns, $\delta = 500$ns, and $\alpha = 10\mu$s, $\delta = 500$ns. Our simulation further support the trend already established by the emulation results.

For small message sizes (16KB–256KB), RD with Harvest's schedule consistently outperforms the best static baseline at low reconfiguration delays. The advantage reaches up to 2.95× for a 256KB message at a 10ns delay, while 16KB, 64KB, and 256KB messages maintain gains of 1.66×–2.95× for delays up to 1μs. Notably, Harvest either outperforms or matches performance of the best of the static baselines for such small messages even for reconfiguration delays upto 10ms.

For medium message sizes (1MB–64MB), Harvest continues to provide benefits at low reconfiguration delays, achieving up to a 3.07× speedup for 1MB messages at 10ns. However, the gains decrease steadily as delay increases.

As message sizes increase further (256MB–4GB), Ring shows overall better performance while for delays up to 10μs, RD with Harvest's schedule generally matches the performance of the best static baseline.

The results highlight the tradeoff between reconfiguration overhead and communication cost, and the benefits of topology reconfiguration for algorithms with fewer communication steps such as Recursive Doubling. At the same time, the example also highlights the potential for topology reconfigurations to outperform both Ring as well as Recursive Doubling on static topologies.

Is Harvest practical?

While we do not focus on a full system implementation in this paper, we quantify the practical compute and synchronization overhead introduced by Harvest. Figure 10 reports the time required to solve the dynamic program for Recursive Doubling across a range of network sizes. For typical scale-up deployments of up to 64 GPUs, the solver completes within 20μs. Even for larger configurations of up to 1024 nodes, the average runtime remains under 35μs. These schedules can also be computed and cached for future use. Furthermore, Figure 11 shows the synchronization overhead measured among GPUs accessing an array in a shared memory space. Each GPU is programmed to flip a bit at its assigned index in the array and we measure the time it takes for all GPUS to complete this operation. To emulate larger systems, we increase the size of the array and initialize multiple threads on remote GPUs that each flip corresponding entries in the shared array. For an 8 GPU network, the average synchronization latency is approximately 4μs while a 256 one shows under 45μs.

Our results indicate that both the compute and synchronization overheads of the synthesis framework are modest relative to the performance gains it enables. These overheads are not fundamental — we can reduce them further through targeted hardware support e.g., through on-chip schedule synthesis, system-level optimizations where we overlap reconfiguration with computation when the stepwise communication is known *a priori*. We discuss the limitations, open questions, and future research directions in Appendix C.4.

7 Related Work

We briefly discuss the significant research efforts in the past in improving collective communication performance.

Topology-aware collectives: A substantial body of work designs collective algorithms specialized for fixed network topologies [16, 47, 66]. Algorithms such as Bruck [15], Sack and Gropp [64], Swing [65], and BineTrees [20] optimize communication for multiported static interconnects. Bruck's algorithm further generalizes AllReduce to $\log_d(n)$ steps for d-port networks, but does not model network congestion. More broadly, collective synthesis approaches [16, 47, 66, 80] assume a static topology throughout execution. In contrast, our work focuses on *topology synthesis*, enabling dynamic reconfiguration during a collective while explicitly balancing reconfiguration delay and congestion.

Reconfiguration-aware circuit-switching: Reconfigurable circuit-switched network topologies have been widely studied in datacenter settings [2, 3, 5, 6, 8, 9, 13, 22, 26, 30, 39, 44–46, 49, 50, 52, 56]. Early systems often assumed negligible reconfiguration delays [9, 50], while later work incorporated reconfiguration cost into optimization objectives [13, 44, 46]. Many of these approaches operate over discrete topology choices and model reconfiguration as a switching cost [13, 46], which limits their ability to capture congestion and routing flexibility. Opera [48] allows multi-hop forwarding within a topology but does not adapt to reconfiguration delays. Our approach bridges reconfiguration cost and network throughput via maximum concurrent flow, enabling multi-hop routing while explicitly accounting for reconfiguration delay.

Circuit-switching for collectives: Recent work has explored reconfigurable interconnects tailored to collective communication [7, 40]. Chronos [63] preschedules circuits using step-wise

collective structure, but does not explicitly consider reconfiguration delays. Actina [78] supports dynamic reconfiguration for ML workloads, but the topology remains static within collective execution. More recent efforts have explored in-collective reconfiguration frameworks. For example, Hammer et al. leverage circuit switching to "short-circuit" rings and accelerate recursive doubling via a threshold-based reconfiguration algorithm [31]. PCCL [41], a recent work, also studies dynamic reconfiguration through an optimization-based framework based on shortest paths. In contrast, Harvest's cost model builds on our prior work [1] and is grounded directly in the maximum concurrent flow objective. It captures the full routing flexibility of the network and supports both splittable and unsplittable flows, rather than restricting routing to a single shortest path. Instead of encoding reconfigurations via integer bitmap variables for each collective step and topology choice, we synthesize the optimal schedule for a fixed number of reconfigurations using dynamic programming, and then select the reconfiguration count that minimizes total cost. We show that this formulation yields globally optimal reconfiguration schedules. Moreover, our formulation exposes an optimal schedule analytically without invoking a solver for various algorithms including, Bruck's, Hypercube, Binomial, and Trivance.

High-throughput topologies: Datacenter network topologies have been extensively studied [4, 11, 27, 70, 74, 79]. Clos-based networks achieve full throughput at high cost [4, 24, 58]. Expander-based topologies reduce hardware cost but sacrifice throughput [51, 74]. Torus-based networks align well with modern parallel workloads despite low bisection bandwidth [33, 82], but can incur substantial congestion. Our work targets such scale-up domains, where adaptive reconfiguration can reduce congestion and improve collective performance.

8 Conclusion

Harvest is a systematic approach that optimizes reconfigurable topologies for collective communication. It explicitly balances propagation delay, congestion, and reconfiguration overhead. Harvest synthesizes reconfiguration schedules. We show it is general and finds provably optimal schedules for Recursive Doubling AllReduce. Our schedules significantly reduce collective completion time. We show the benefits of reconfiguration depend critically on when and how we apply it.

Our framework opens several directions for future work which include joint synthesis of collective communication and interconnect topologies. This work lays the foundation for adaptive photonic scale-up domains where collectives and topologies co-evolve.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. This work was supported by the NSF Future CoRe program under Award # 2551372.

References

[1] Vamsi Addanki. When light bends to the collective will: A theory and vision for adaptive photonic scale-up domains. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*, HotNets '25, page 326–334, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3772356.3772395.

[2] Vamsi Addanki, Chen Avin, Goran Dario Knabe, Giannis Patronas, Dimitris Syrivelis, Nikos Terzenidis, Paraskevas Bakopoulos, Ilias Marinos, and Stefan Schmid. Vermilion: A traffic-aware reconfigurable optical interconnect with formal throughput guarantees, 2025. URL: https://arxiv.org/abs/2504.09892, arXiv:2504.09892.

[3] Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1), mar 2023. doi:10.1145/3579312.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1402958.1402967.

[5] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Shale: A practical, scalable oblivious reconfigurable network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 449–464, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672248.

[6] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. Optimal oblivious reconfigurable networks. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1339–1352, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520020.

[7] Rukshani Athapathu and George Porter. Reconfigurability within collective communication algorithms. In *Proceedings of the 2nd Workshop on Networks for AI Computing*, NAIC '25, page 43–49, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3748273.3749203.

[8] Chen Avin and Stefan Schmid. Revolutionizing datacenter networks via reconfigurable topologies. *Commun. ACM*, 68(6):44–53, June 2025. doi:10.1145/3708980.

[9] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 782–797, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3406221.

[10] Giacomo Bernardi, Ratul Mahajan, C. Seshadhri, Enrico Carlesso, Chinchu Merine Joseph, Saurabh Kumar, Pavan Manikonda, Luiza Popa, Randy Ram, Steven Robinson, and Elizabeth Tennent. Rng: Flat datacenter networks at scale, 2026. URL: https://arxiv.org/abs/2604.15261, arXiv:2604.15261.

[11] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359, 2014. doi:10.1109/SC.2014.34.

[12] Garrett Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacuman, Rev. Ser. A*, 5:147–151, 1946.

[13] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, page 75–88, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2896377.2901479.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[15] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, 1997. doi:10.1109/71.642949.

[16] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21, page 62–75, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3437801.3441620.

[17] A. Cevrero, I. Ozkaya, T. Morf, T. Toifl, M. Seifried, F. Ellinger, M. Khafaji, J. Pliva, R. Henker, N. Ledentsov, J.-R. Kropp, V. Shchukin, M. Zoldak, L. Halmo, I. Eddie, and J. Turkiewicz. 4×40 gb/s 2 pj/bit optical rx with 8ns power-on and cdr-lock time in 14nm cmos. In *Optical Fiber Communication Conference*, page M2D.3. Optica Publishing Group, 2018. URL: https://opg.optica.org/abstract.cfm?URI=OFC-2018-M2D.3, doi:10.1364/OFC.2018.M2D.3.

[18] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1206,

arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1206, doi:10.1002/cpe.1206.

[19] Gertjan Coudyzer, Michiel Verplaetse, Borre Van Lombergen, Robert Borkowski, Thibaut Gurne, Michael Straub, Yannick Lefevre, Peter Ossieur, René Bonk, Werner Coomans, et al. 100 gbit/s pam-4 linear burst-mode transimpedance amplifier for upstream flexible passive optical networks. *Journal of Lightwave Technology*, 41(12):3652–3659, 2023.

[20] Daniele De Sensi, Saverio Pasqualoni, Lorenzo Piarulli, Tommaso Bonato, Seydou Ba, Matteo Turisini, Jens Domke, and Torsten Hoefler. Bine trees: Enhancing collective operations by optimizing communication locality. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, page 1901–1916, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3712285.3759835.

[21] Eric Ding and Rachee Singh. Pipswitch: A circuit switch using programmable integrated photonics. In *Optical Fiber Communication Conference (OFC) 2025*, page W2A.41. Optica Publishing Group, 2025. URL: https://opg.optica.org/abstract.cfm?URI=OFC-2025-W2A.41, doi:10.1364/OFC.2025.W2A.41.

[22] Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. *SIGACT News*, 50(2):62–79, jul 2019. doi:10.1145/3351452.3351464.

[23] Alex Forencich, Valerija Kamchevska, Nicolas Dupuis, Benjamin G Lee, Christian W Baks, George Papen, and Laurent Schares. A dynamically-reconfigurable burst-mode link using a nanosecond photonic switch. *Journal of Lightwave Technology*, 38(6):1330–1340, 2020.

[24] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 57–70, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672233.

[25] Alexandru M. Gherghescu, Vlad-Andrei Bădoiu, Alexandru Agache, Mihai-Valentin Dumitru, Iuliu Vasilescu, Radu Mantu, and Costin Raiciu. I've got 99 problems but flops ain't one. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, HotNets '24, page 195–204, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3696348.3696893.

[26] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 216–229, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2934872.2934911.

[27] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 51–62, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1592568.1592576.

[28] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2934872.2934908.

[29] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

[30] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: a reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 319–330, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2619239.2626328.

[31] Sarah-Michelle Hammer, Stefan Schmid, Rachee Singh, and Vamsi Addanki. Short-circuiting rings for low-latency allreduce, 2025. URL: https://arxiv.org/abs/2510.03491, arXiv:2510.03491.

[32] Alexander Ishii and Ryan Wells. The Nvlink-Network Switch: Nvidia's Switch Chip for High Communication-Bandwidth Superpods . In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–23, Los Alamitos, CA, USA, August 2022. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/HCS55958.2022.9895480, doi:10.1109/HCS55958.2022.9895480.

[33] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3579371.3589350.

[34] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 761–772. IEEE, 2016.

[35] Anton Jürß, Vamsi Addanki, and Stefan Schmid. Trivance: Latency-optimal allreduce by shortcutting multiport networks. In *Proceedings of the ACM SIGCOMM 2026 Conference*, ACM SIGCOMM '26, Denver, CO, USA, 2026. Association for Computing Machinery.

[36] Anton Jürß and Stefan Schmid. Revisiting bruck: Phase-efficient all-to-all communication in reconfigurable networks. In *ACM SIGCOMM Workshop on Hot Topics in Optical Technologies and Applications in Networking*, HotOptics '26, Denver, CO, USA, 2026. ACM.

[37] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. Sip-ml: high-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 657–675, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452296.3472900.

[38] Dmitry Kolmakov and Xuecang Zhang. A generalization of the allreduce operation. *arXiv preprint arXiv:2004.09362*, 2020. URL: https://arxiv.org/abs/2004.09362.

[39] Janardhan Kulkarni, Stefan Schmid, and Paweł Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 318–327, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461786.

[40] Abhishek Vijaya Kumar, Arjun Devraj, Darius Bunandar, and Rachee Singh. A case for server-scale photonic connectivity. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, HotNets '24, page 290–299, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3696348.3696856.

[41] Abhishek Vijaya Kumar, Arjun Devraj, and Rachee Singh. Pccl: Photonic circuit-switched collective communication for distributed ml, 2025. URL: https://arxiv.org/abs/2509.15450, arXiv:2509.15450.

[42] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL: https://openreview.net/forum?id=qrwe7XHTmYb.

[43] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR, 18–24 Jul 2021. URL: https://proceedings.mlr.press/v139/lewis21a.html.

[44] Xin Li and M. Hamdi. On scheduling optical packet switches with reconfiguration delay. *IEEE Journal on Selected Areas in Communications*, 21(7):1156–1164, 2003. doi:10.1109/JSAC.2003.815843.

[45] Cong Liang, Xiangli Song, Jing Cheng, Mowei Wang, Yashe Liu, Zhenhua Liu, Shizhen Zhao, and Yong Cui. Negotiator: Towards a simple yet effective on-demand reconfigurable datacenter network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 415–432, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672222.

[46] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2716281.2836126.

[47] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. Rethinking machine learning collective communication as a multi-commodity flow problem. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 16–37, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672249.

[48] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1–18, Santa Clara, CA, February 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/mellette.

[49] William M. Mellette, Alex Forencich, Rukshani Athapathu, Alex C. Snoeren, George Papen, and George Porter. Realizing rotornet: Toward practical microsecond scale optical networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 392–414, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672273.

[50] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 267–280, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3098822.3098838.

[51] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. A throughput-centric view of the performance of datacenter topologies. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 349–369, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452296.3472913.

[52] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Optical Switching and Networking*, 41:100621, 2021. URL: https://www.sciencedirect.com/science/article/pii/S1573427721000187, doi:10.1016/j.osn.2021.100621.

[53] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3458817.3476209.

[54] DGX NVIDIA. Superpod: Next generation scalable infrastructure for ai leadership. 2023. URL: https://docs.nvidia.com/https:/docs.nvidia.com/dgx-superpod-reference-architecture-dgx-h100.pdf.

[55] Polatis. POLATIS® 7000 Optical Circuit Switch. https://www.hubersuhner.com/en/shop/product/other-systems/optical-switches/rack-mount-circuit-switches/85223159/polatis-7000-optical-circuit-switch.

[56] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 447–458, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486001.2486007.

[57] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 66–85, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544265.

[58] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 691–706, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672265.

[59] Rolf Rabenseifner. Optimization of collective reduction operations. In Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2004*, pages 1–9, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[60] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR, 17–23 Jul 2022. URL: https://proceedings.mlr.press/v162/rajbhandari22a.html.

[61] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2020. doi:10.1109/SC41405.2020.00024.

[62] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 81–92, 2020. doi:10.1109/ISPASS48437.2020.00018.

[63] Sundararajan Renganathan and Nick McKeown. Chronos: Prescheduled circuit switching for llm training. In *Proceedings of the 2nd Workshop on Networks for AI Computing*, NAIC '25, page 89–97, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3748273.3749210.

[64] Paul Sack and William Gropp. Collective algorithms for multiported torus networks. *ACM Trans. Parallel Comput.*, 1(2), February 2015. doi:10.1145/2686882.

[65] Daniele De Sensi, Tommaso Bonato, David Saam, and Torsten Hoefler. Swing: Short-cutting rings for higher bandwidth allreduce. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1445–1462, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/de-sensi.

[66] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding collective algorithm synthesis using communication sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 593–612, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/shah.

[67] Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, apr 1990. doi:10.1145/77600.77620.

[68] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[69] Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 29–41, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/singla.

[70] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, San Jose, CA, April 2012. USENIX Association. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla.

[71] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005. arXiv:https://doi.org/10.1177/1094342005051521, doi:10.1177/1094342005051521.

[72] Luis Torrijos-Morán and Daniel Pérez-López. Industry insight: photonics to scale ai data centers. *npj Nanophotonics*, 3(1):8, 2026.

[73] Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, Oliver Curtis, Grigori Fursin, Miro Hodak, Hiwot Kassa, Anton Lokhmotov, Dejan Miskovic, Yuechao Pan, Manu Prasad Manmathan, Liz Raymond, Tom St. John, Arjun Suresh, Rowan Taubitz, Sean Zhan, Scott Wasson, David Kanter, and Vijay Janapa Reddi. Mlperf power: Benchmarking the energy efficiency of machine learning systems from μwatts to mwatts for sustainable ai. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1201–1216, 2025. doi:10.1109/HPCA61900.2025.00092.

[74] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 205–219, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2999572.2999580.

[75] Mark Wade, Erik Anderson, Shahab Ardalan, Pavan Bhargava, Sidney Buchbinder, Michael L. Davenport, John Fini, Haiwei Lu, Chen Li, Roy Meade, Chandru Ramamurthy, Michael Rust, Forrest Sedgwick, Vladimir Stojanovic, Derek Van Orden, Chong Zhang, Chen Sun, Sergey Y. Shumarayev, Conor O'Keeffe, Tim T. Hoang, David Kehlet, Ravi V. Mahajan, Matthew T. Guzy, Allen Chan, and Tina Tran. Teraphy: A chiplet technology for low-power, high-bandwidth in-package optical i/o. *IEEE Micro*, 40(2):63–71, 2020. doi:10.1109/MM.2020.2976067.

[76] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang.

[77] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. Astra-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 283–294, 2023. doi:10.1109/ISPASS57527.2023.00035.

[78] Zhenguo Wu, Benjamin Klenk, Larry Dennison, and Keren Bergman. Actina: Adapting circuit-switching techniques for ai networking architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, page 1211–1222, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3712285.3759842.

[79] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding lifecycle management complexity of datacenter topologies. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 235–254, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/nsdi19/presentation/zhang.

[80] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Jason Fantl, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. Efficient Direct-Connect topologies for collective communications. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 705–737, Philadelphia, PA, April 2025. USENIX Association. URL: https://www.usenix.org/conference/nsdi25/presentation/zhao-liangyu.

[81] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, zhifeng Chen, Quoc V Le, and James Laudon. Mixture-of-experts with expert choice routing. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 7103–7114. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/2f00ecd787b432c1d36f3de9800728eb-Paper-Conference.pdf.

[82] Yazhou Zu, Alireza Ghaffarkhah, Hoang-Vu Dang, Brian Towles, Steven Hand, Safeen Huda, Adekunle Bello, Alexander Kolbasov, Arash Rezaei, Dayou Du, Steve Lacy, Hang Wang, Aaron Wisner, Chris Lewis, and Henri Bahini. Resiliency at scale: Managing Google's TPUv4 machine learning supercomputer. In *21st*

USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), pages 761–774, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/zu.

Appendices are supporting material that has not been peer-reviewed.

A MISOCP Formulation for the Subproblem

We formulate a Mixed-Integer Second-Order Conic Program (MISOCP) to compute the optimal topology $G_{a,b}$ that minimizes the completion time of steps a through b without reconfiguration. Recall that this minimization objective is our subproblem in the schedule synthesis §4.1.

Variables:

- $x_{u,v} \in \mathbb{Z}_{\geq 0}$: number of directed links from node u to node v.
- $f_{s,t}^{(i)}(u,v) \geq 0$: flow routed on directed edge (u,v) for demand (s,t) in step i.
- $T_i \geq 0$: transmission time of step i.
- $\theta_i \geq 0$: throughput scaling factor for step i.

Parameters:

- d: maximum in-degree and out-degree per node.
- c: capacity of a single directed edge.
- $D_{s,t}^{(i)} = m_i \cdot \mathcal{M}_i(s,t)$: demand from node s to t in step i.
- β: inverse of the link bandwidth.

Our goal is to minimize the sum of transmission times T_i for the sequence of steps a through b i.e., minimizing the total transmission time for these steps together.

Objective:

$$\min \quad \sum_{i=a}^{b} T_i \tag{9}$$

Edges in the topology are binary variables in our formulation. To capture the maximum number of links available at each node, we impose node degree constraints as follows.

Degree constraints:

$$\sum_{v \in V} x_{u,v} \leq d \qquad \forall u \in V \tag{10}$$

$$\sum_{u \in V} x_{u,v} \leq d \qquad \forall v \in V \tag{11}$$

The rest of the formulation follows standard maximum concurrent flow formulation. In particular, we consider flow variables $f_{s,t}^{(}i)(u,v)$ sent on edge (u,v), corresponding to the demand between (s,t) in step i. To satisfy flow conservation, we incorporate the following constraints at source, destination, and intermediate nodes.

Flow conservation for each step:

$$\sum_{v} f_{s,t}^{(i)}(u,v) - \sum_{v} f_{s,t}^{(i)}(v,u) = \begin{cases} \theta_i \cdot D_{s,t}^{(i)}, & u = s \\ -\theta_i \cdot D_{s,t}^{(i)}, & u = t \\ 0, & \text{otherwise} \end{cases} \qquad \forall i,s,t,u \tag{12}$$

Further, the total flow between (u,v) must satisfy the available capacity between the two nodes given by $c \cdot x_{u,v}$, where $x_{u,v}$ is the variable indicating the number of edges between u,v.

Edge capacity constraints:

$$\sum_{s,t} f_{s,t}^{(i)}(u,v) \leq c \cdot x_{u,v} \quad \forall i,(u,v) \tag{13}$$

Finally, since our objective is to minimize transmission time, we express the following constraint as an inequality.

Transmission time:

$$T_i \geq \frac{\beta m_i}{\theta_i}, \quad \theta_i \geq 0, T_i \geq 0 \qquad \forall i. \tag{14}$$

The transmission time for step i is exactly $\frac{\beta m_i}{\theta_i}$. However, writing this as an inequality does not relax the solution: the constraint is tight at optimality, since any strictly larger value of T_i would increase the objective.

The above constraint is equivalent to the bilinear constraint

$$\theta_i \cdot T_i \geq \beta m_i, \quad \forall i. \tag{15}$$

Constraint (15) admits a second-order conic representation. In particular, it is equivalent to the following second-order cone (SOC) constraint:

$$\left\| \begin{bmatrix} 2\sqrt{\beta m_i} \\ \theta_i - T_i \end{bmatrix} \right\|_2 \leq \theta_i + T_i, \quad \forall i. \tag{16}$$

Variable type:

$$f_{s,t}^{(i)}(u,v) \geq 0$$
$$x_{u,v} \in \mathbb{Z}_{\geq 0}$$
$$\theta_i \in (0,1]$$
$$T_i \in \mathbb{R}_{\geq 0}$$

The solution to this formulation yields the optimal topology $G_{a,b}$, which is constructed from the decision variables $x_{u,v}$ indicating the number of directed edges between every node pairs. The formulation also returns the per-step completion times T_i, and the total completion time $\sum_{i=a}^{b} T_i$, which are used by the outer dynamic program.

We restrict the search space in our evaluations to a collection of topologies: *(i)* shifted rings, *(ii)* an expander, *(iii)* shifted torus topologies (by relabeling the nodes), and the synthesizer selects the best topology for each step of the collective. This restriction converts the MISOCP into SOCP without integer variables for finding optimal topologies.

B Switching Schedules for Other Collectives

Our Observations in §5 not only hold for Recursive Doubling, but also apply directly to a class of algorithms where the communication pattern is cyclic in nature, and the communication distance progresses geometrically across steps. In this section, we show that our observations hold for Hypercube All-to-All, Bruck's (radix r) AllGather, Reduce-Scatter, All-to-All, and Binomial Tree Broadcast. For completeness, we describe the algorithms and prove that the observations hold.

B.1 Hypercube All-to-All (Cyclic)

The initial state of All-to-All is a permutation of blocks across nodes, where each node u holds blocks (u,v) for all v. The goal is to route each block (u,v) to node v.

We assume the number of nodes is $n = 2^s$. For each step $i = 0,1,\ldots,s-1$:

- Node u sends to node $u+2^i \pmod n$.
- Block (u,v) moves in step i if and only if bit i of $(v-u) \bmod n$ is 1.

After $s=\log_2 n$ steps, every block reaches its destination. The crucial difference from cyclic Recursive Doubling for AllGather operation is that the data size in every step is now $\frac{m}{2}$.

We define the forward cyclic distance from source u to destination v as

$$\Delta(u,v) = (v-u) \bmod n.$$

This quantity represents the number of positions a block must advance around the cycle in order to reach its destination.

Let $\Delta(u,v) = \sum_{i=0}^{s-1} b_i(u,v)2^i$, where $b_i(u,v) \in \{0,1\}$ denotes the i-th bit in the binary representation of $\Delta(u,v)$. Equivalently, $b_i(u,v) = \left\lfloor \frac{\Delta(u,v)}{2^i} \right\rfloor \bmod 2$. A block (u,v) moves in step i if and only if $b_i(u,v)=1$.

For completeness, we now prove the correctness of the cyclic version of Hypercube All-to-All (originally pairwise). Consider an arbitrary block (u,v). Initially, it is located at node u. At step i, the block advances by distance 2^i if and only if $b_i(u,v)=1$. Therefore, the total distance traversed by the block is

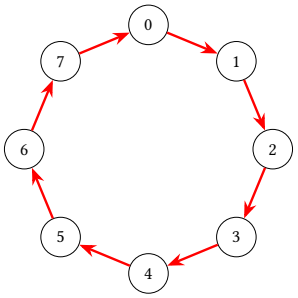
$$\sum_{i=0}^{s-1} b_i(u,v)2^i = \Delta(u,v) = (v-u) \bmod n.$$

The final location of the block is

$$u+\Delta(u,v) \equiv u+(v-u) \equiv v \pmod n.$$

Every block reaches its intended destination, i.e., a block (u,v) originating at node u eventually reaches node v, satisfying the final state of an all-to-all operation. In the following, we show an example of Hypercube All-to-All with 8 nodes.

Step 0 - distance 1 communication:



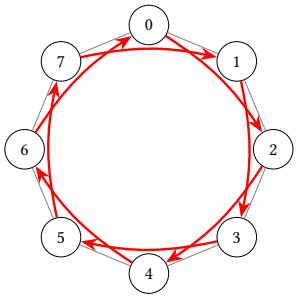
Step: $s=0$
Distance: $2^0=1$
Pattern: $i \to i+1$
Send if: bit 0 of Δ is 1
Data: $\frac{m}{2}$

$u \to u+1 \pmod 8$

Node	Blocks sent in step 0
0	01 03 05 07
1	10 12 14 16
2	21 23 25 27
3	30 32 34 36
4	41 43 45 47
5	50 52 54 56
6	61 63 65 67
7	70 72 74 76

Node	Blocks held after step 0
0	00 02 04 06 70 72 74 76
1	01 03 05 07 11 13 15 17
2	10 12 14 16 20 22 24 26
3	21 23 25 27 31 33 35 37
4	30 32 34 36 40 42 44 46
5	41 43 45 47 51 53 55 57
6	50 52 54 56 60 62 64 66
7	61 63 65 67 71 73 75 77

Step 1 - distance 2 communication:



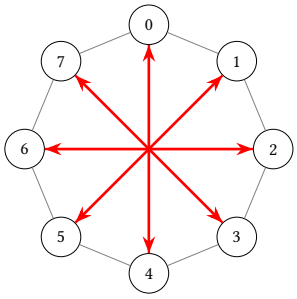
Step: $s=1$
Distance: $2^1=2$
Pattern: $i \to i+2$
Send if: bit 1 of Δ is 1
Data: $\frac{m}{2}$

$u \to u+2 \pmod 8$

Node	Blocks sent in step 1
0	02 06 72 76
1	03 07 13 17
2	10 14 20 24
3	21 25 31 35
4	32 36 42 46
5	43 47 53 57
6	50 54 60 64
7	61 65 71 75

Node	Blocks held after step 1
0	00 04 50 54 60 64 70 74
1	01 05 11 15 61 65 71 75
2	02 06 12 16 22 26 72 76
3	03 07 13 17 23 27 33 37
4	10 14 20 24 30 34 40 44
5	21 25 31 35 41 45 51 55
6	32 36 42 46 52 56 62 66
7	43 47 53 57 63 67 73 77

Step 2 - distance 4 communication



Step: $s=2$
Distance: $2^2=4$
Pattern: $i \to i+4$
Send if: bit 2 of Δ is 1
Data: $\frac{m}{2}$

$u \to u+4 \pmod 8$

Node	Blocks sent in step 2
0	04 54 64 74
1	05 15 65 75
2	06 16 26 76
3	07 17 27 37
4	10 20 30 40
5	21 31 41 51
6	32 42 52 62
7	43 53 63 73

Node	Final blocks after step 2
0	00 10 20 30 40 50 60 70
1	01 11 21 31 41 51 61 71
2	02 12 22 32 42 52 62 72
3	03 13 23 33 43 53 63 73
4	04 14 24 34 44 54 64 74
5	05 15 25 35 45 55 65 75
6	06 16 26 36 46 56 66 76
7	07 17 27 37 47 57 67 77

At this step, every node v now contains all blocks (u,v) destined to v.

Next, we prove that Observation 3, 4 also hold for Hypercube All-to-All.

Proof of Observation 3 for Hypercube All-to-All. The communication pattern of Hypercube All-to-All is identical to Recursive Doubling for AllGather, except the data size in each step, each node u communicates with node $u+2^{i-1}$ in step i (indexed from 1). The proof follows from the same argument as in the proof of Observation 3 for Recursive Doubling. □

Proof of Observation 4 for Hypercube All-to-All. Similar to the case of Recursive Doubling, we characterize the interval of steps a through b as: in step a, each node u communicates with $u+2^{a-1}$ (with steps indexed from 1). The sequence of minimum path lengths for steps a through b is $\langle 1,\ldots,2^{b-a}\rangle$. Likewise, the minimum congestion we incur is at least $\langle 1,\ldots,2^{b-a}\rangle$, which corresponds to each step from a through b. The topology that establishes direct connections according to step a's communication pattern, i.e., a direct link between u and $u+2^{a-1}$, achieves exactly the minimum path lengths and congestion for every step a through b. Specifically, for step a, the communication distance is reduced to 1. For step $a+1$, node u communicates with $u+2^a$, which is at distance 2: in our chosen topology, u connects to $u+2^{a-1}$, which in turn connects to $u+2^{a-1}+2^{a-1}=u+2^a$. The proof follows for both path lengths and congestion.

We can now express the completion time of steps $a-b$ where the communication pattern matches that of step a as:

$$t_c(a,b)=\sum_{i=a}^{b} DCT(m_i\cdot\mathcal{M}_i, G_{a,b})$$
$$=\alpha\cdot(b-a+1)+\delta\cdot\sum_{i=a}^{b}2^{i-a}+\beta\cdot m\cdot\sum_{i=a}^{b}\frac{1}{2}\cdot 2^{i-a}$$
$$=\alpha\cdot(b-a+1)+\delta\cdot\left(2^{(b-a+1)}-1\right)+\beta\cdot m\cdot\frac{2^{b-a+1}-1}{2} \quad (17)$$

□

B.2 Bruck's AllGather/Reduce-Scatter

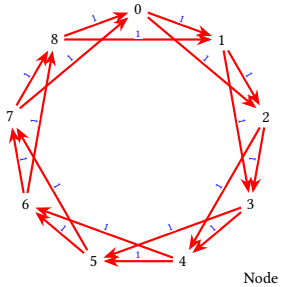
Bruck's AllGather [15] operates on a cyclic communication pattern and the general case is expressed in radix-r form. We next consider Bruck's concatenation (AllGather) algorithm and show that our observations also hold in this setting. Since Bruck's Reduce-Scatter follows the reverse communication pattern of Bruck's AllGather, the same analysis applies. We briefly describe the algorithm for completeness.

The initial state before AllGather consists of $n=r^s$ nodes labeled $0,\ldots,n-1$, where node u initially holds block uu. Bruck's AllGather begins with an initial cyclic shift so that node u holds data beginning at index u. This rearrangement allows subsequent communication steps to follow a regular cyclic pattern. The algorithm then proceeds in $s=\log_r n$ communication steps. At step $i=0,1,\ldots,s-1$, each node u communicates with the nodes, $u+jr^i \pmod n$, where $j=1,\ldots,r-1$.

The local data is partitioned into r contiguous groups of size r^i. One group is retained locally, while the remaining $r-1$ groups are transmitted to the corresponding neighbors. Thus, step i communicates along offsets of size r^i, progressively resolving one additional radix-r digit of the final block placement. After step i, the lowest $i+1$ radix-r digits of the final block positions are correct. Consequently, after all $s=\log_r n$ steps, every node contains all n blocks. Finally, an inverse cyclic shift is performed to restore the original ordering of blocks.

The algorithm completes in $\log_r n$ steps, where r is the radix parameter. The choice of r determines the multicative factor for the distance of communication in each step. In the following, we show an example of Bruck's AllGather with 9 nodes and radix $r=3$.

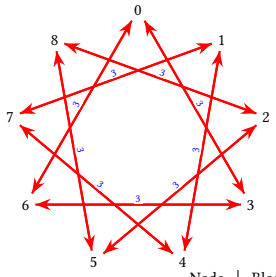
Step 0, $r=3$:



Offset: 1
Neighbors: 2
Data: 2 blocks of size $\frac{m}{9}$

Node	Blocks
0	00 77 88
1	11 88 00
2	22 00 11
3	33 11 22
4	44 22 33
5	55 33 44
6	66 44 55
7	77 55 66
8	88 66 77

Step 1, $r=3$:



Offset: 3
Neighbors: 2
Data: 6 blocks of size $\frac{m}{9}$

Node	Blocks
0	00 77 88 33 11 22 66 44 55
1	11 88 00 44 22 33 77 55 66
2	22 00 11 55 33 44 88 66 77
3	33 11 22 66 44 55 00 77 88
4	44 22 33 77 55 66 11 88 00
5	55 33 44 88 66 77 22 00 11
6	66 44 55 00 77 88 33 11 22
7	77 55 66 11 88 00 44 22 33
8	88 66 77 22 00 11 55 33 44

Proof of Observation 3 for Bruck's AllGather. For Bruck's with radix r, we prove that the observation holds for a network with degree r. In step i, each node u communicates with nodes $u+jr^{i-1} \pmod n$, where $j=1,\ldots,r-1$. Establishing these direct links does not break connectivity for any later step $k\geq i$. In step k, node u must communicate with $u+jr^{k-1} \pmod n$, where $j=1,\ldots,r-1$. Since $jr^{k-1}=r^{k-i}(jr^{i-1})$, starting from node u, follow the step-i link with offset jr^{i-1} exactly r^{k-i} times. After ℓ such hops, the current node is $u+\ell jr^{i-1} \pmod n$. Setting $\ell=r^{k-i}$ gives $u+r^{k-i}jr^{i-1}=u+jr^{k-1} \pmod n$, which is exactly the destination required in step k. Therefore every communication edge required in step k corresponds to a path in the topology established for step i. Hence the topology established by the communication pattern of step i preserves connectivity for all subsequent steps $k\geq i$.

□

Proof of Observation 4 for Bruck's AllGather. Similar to Recursive Doubling, we characterize the interval of steps a through

b. In Bruck's AllGather, step a requires each node u to communicate with nodes $u + jr^{a-1} \pmod n$, where $j = 1,\ldots,r-1$ and steps are indexed from 1. The topology that establishes direct connections according to step a's communication pattern therefore contains direct links from u to $u+jr^{a-1} \pmod n$ for every $j=1,\ldots,r-1$.

Now consider any later step $i \geq a$. In step i, node u must communicate with nodes $u+jr^{i-1} \pmod n$. Since $jr^{i-1} = r^{i-a}(jr^{a-1})$, this destination can be reached by repeatedly traversing links of length jr^{a-1}. After ℓ such hops, the current node is $u + \ell jr^{a-1} \pmod n$. Setting $\ell = r^{i-a}$ gives $u + r^{i-a} jr^{a-1} = u + jr^{i-1} \pmod n$, which is exactly the destination required in step i.

Thus the sequence of minimum path lengths for steps a through b is $\langle 1,\ldots,r^{b-a}\rangle$. The same topology also achieves the corresponding minimum congestion for each step, since each step-i communication is routed along paths of length r^{i-a} over the step-a links. The proof follows for both path length and congestion.

We can now express the completion time of steps a through b, where the topology matches the communication pattern of step a, as

$$\begin{aligned} t_c(a,b) &= \sum_{i=a}^{b} DCT(m_i \cdot \mathcal{M}_i, G_{a,b}) \\ &= \alpha\cdot(b-a+1) + \delta\cdot\sum_{i=a}^{b} r^{i-a} + \beta\cdot\sum_{i=a}^{b} m_i r^{i-a} \\ &= \alpha\cdot(b-a+1) + \delta\cdot\frac{r^{b-a+1}-1}{r-1} + \beta\cdot\sum_{i=a}^{b} m_i r^{i-a}. \end{aligned} \tag{18}$$

If each node holds total data size m after AllGather and each block has size $\frac{m}{n}$, then in Bruck's radix-r AllGather, step i sends r^{i-1} blocks along each offset. Since the $r-1$ offsets are used in parallel, the transmission delay, i.e., the β term, is determined by the per-offset message size rather than the aggregate data sent across all offsets. The per-offset data size in step i is then, $m_i = r^{i-1}\cdot\frac{m}{n}$. Substituting this gives,

$$\begin{aligned} t_c(a,b) &= \alpha\cdot(b-a+1) + \delta\cdot\frac{r^{b-a+1}-1}{r-1} + \beta\cdot\frac{m}{n}\sum_{i=a}^{b} r^{i-1}r^{i-a} \\ &= \alpha\cdot(b-a+1) + \delta\cdot\frac{r^{b-a+1}-1}{r-1} + \beta\cdot\frac{mr^{a-1}}{n}\sum_{i=a}^{b} r^{2(i-a)} \\ &= \alpha\cdot(b-a+1) + \delta\cdot\frac{r^{b-a+1}-1}{r-1} + \beta\cdot\frac{mr^{a-1}}{n}\cdot\frac{r^{2(b-a+1)}-1}{r^2-1}. \end{aligned} \tag{19}$$

□

The optimal switching schedule for Bruck's AllGather can now be computed directly based on Observation 3, 4 i.e., for a range of steps a through b, the optimal topology is the one that establishes direct connections according to step a's communication pattern. The schedule can be computed by dynamic programming over the steps using Algorithm 1, where the cost of a range of steps a through b is computed according to the above expression for $t_c(a,b)$. The schedules for Bruck's Reduce-Scatter can be computed similarly, since the communication pattern is the reverse of Bruck's AllGather, and AllReduce is a sequence of Reduce-Scatter followed by AllGather.

B.3 Bruck's All-to-All & Binomial Tree

Bruck's All-to-all algorithm is a direct extension of Bruck's AllGather, where the communication pattern in step i is the same as that of step i in Bruck's AllGather, except that each node sends $\frac{m}{r}$ blocks along each offset. The same analysis applies to show that our observations hold for Bruck's All-to-All. Binomial tree broadcast follows the communication pattern of cyclic Recursive Doubling from a single root node perspective, and the observations hold for this case as well.

B.4 Trivance

Trivance [35] completes AllGather and Reduce-Scatter in $\log_3(n)$ steps, similar to Bruck's $r=3$, but significantly reduces congestion and achieves faster completion times for small message sizes in Torus topologies. Interestingly, Trivance also follows a cyclic communication pattern, and the distance triples in each step. Our observations hold directly for Trivance as well. We omit the formal proof for brevity, as it follows the same arguments as the proof for recursive doubling. Specifically for Trivance, the left and right communications can be decomposed to two separate rings. In each ring, the distance triples in each step. For observation 3, the topology that establishes direct connections according to step i of trivance preserves connectivity for later steps. For observation 4, the topology that establishes direct connections according to step a's communication pattern achieves the minimum path lengths and congestion for every step a through b. The optimal schedule can then be computed by dynamic programming over the steps, while accounting for the reconfiguration delay. Our observations also hold for the All-to-All extension of Trivance [36].

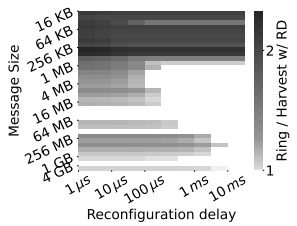
C Discussion

C.1 Generalizing the Observations

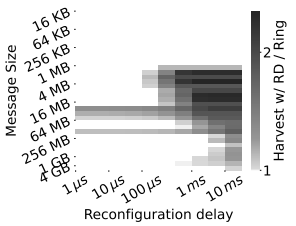
We notice that the same proof for Observation 3, 4 in fact holds for any collective algorithm that satisfies two properties. First, it must be cyclic i.e., the communication pattern in each step is represented as $u \rightarrow (u+d) \mod n$ where each node u communicates with a node at offset d. This property allows applying our Observation 3 on connectivity. Second, the communication distance follows a genometric progression e.g., Recursive Doubling progresses in steps of 2, Bruck's progresses in steps of radix r, Binomial tree progresses in steps of 2. This property allows us to apply Observation 4 on optimality of the topology. For such algorithms, our observations hold and the optimal schedule can be computed by dynamic programming over the steps, while accounting for the reconfiguration delay.

C.2 Ring Algorithm

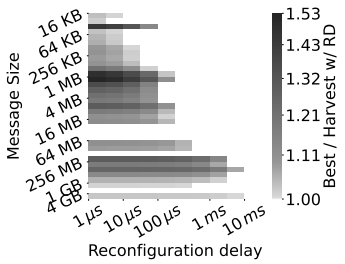
Naturally, for ring-based collectives (AllGather, Reduce-Scatter, AllReduce, and All-to-All variants), Observation 3 implies that the topology matching the communication pattern of any step i, which is a ring, preserves connectivity for all subsequent steps. Likewise, Observation 4 implies that the optimal topology for any range of steps a through b remains a ring. Consequently, the optimal schedule for ring algorithms is a static ring topology with no further reconfigurations. The tradeoff is that ring algorithms require a larger number of communication steps, resulting in a higher latency cost (α), whereas algorithms such as Recursive Doubling and Bruck's communication



(a) When Recursive Doubling AllGather with Harvest outperforms Ring AllGather on a static topology.

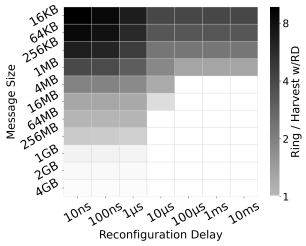


(b) When Ring AllGather on a static topology outperforms Recursive Doubling AllGather with Harvest.

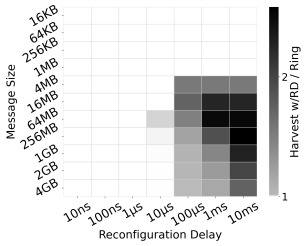


(c) When Recursive Doubling AllGather with Harvest outperforms the best among Recursive Doubling and Ring on a static ring topology.

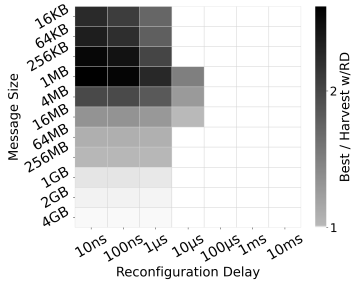
Figure 12: [Hardware emulation] Ring vs Recursive Doubling for AllGather. White space indicates a speedup[5] less than or equal to 1×.



(a) When Recursive Doubling AllReduce with Harvest outperforms Ring AllReduce on a static topology.



(b) When Ring AllReduce on a static topology outperforms Recursive Doubling AllReduce with Harvest.



(c) When Recursive Doubling AllReduce with Harvest outperforms the best among Recursive Doubling and Ring on a static ring topology.

Figure 13: [Simulations] Ring vs Recursive Doubling for AllReduce. White space indicates a speedup[5] less than or equal to 1×.

patterns complete in fewer steps at the cost of higher congestion ($\frac{1}{\theta}$) and potentially additional reconfiguration overhead (α_r).

Figure 12 (hardware emulation) compares the performance of a ring algorithm executed on a static ring topology against Recursive Doubling executed using the optimal schedule synthesized by Harvest. Here, the ring topology is established as described in §6 where 8 BlueField-3 NICs are connected in a ring with 100Gbps links. Since the physical topology is a ring (unless reconfigured), running a ring algorithm incurs no bandwidth sharing. We observe that the ring algorithm performs favorably against Recursive Doubling when reconfiguration costs dominate and when the message size is small, whereas Recursive Doubling benefits from substantially fewer communication steps when reconfiguration overhead is sufficiently small. This highlights the fundamental tradeoff between communication steps, congestion, and topology reconfiguration overhead.

Figure 13 shows our simulation results comparing Recursive Doubling with Harvest schedules, the ring algorithm on a static ring topology, and Recursive Doubling on a static ring topology. As an example, we consider $n = 64$ nodes, setup latency $\alpha = 500$ns, propagation delay $\delta = 50$ns (e.g., in a scaleup domain), each GPU equipped with one link of bandwidth $b = 800$Gbps. Figure 16, 17 in Appendix D show our results for $\alpha = 500$ns, $\delta = 500$ns, and $\alpha = 10\mu$s, $\delta = 500$ns.

We observe that the performance of Harvest with Recursive Doubling over the static ring topology shows the highest gains for low reconfiguration delays and smaller message sizes. The performance against ring algorithm is especially pronounced for small message sizes (16KB-256KB), with the advantage reaching up to 9.72× for a 16KB message size at a 10ns reconfiguration delay. Even at a high reconfiguration delay of 10ms, Harvest continues to show gains, delivering a 2.29× performance improvement over the ring algorithm for 256KB messages.

For small message sizes (16KB-256KB), Recursive Doubling with Harvest's schedule shows up to a 2.95× performance improvement at 10ns reconfiguration delay, while maintaining the same performance as the best baseline once the delay reaches 10μs or higher. The peak speedup is achieved at 1MB message sizes, delivering a substantial 3.07× advantage at 10ns delay.

For medium message sizes (4MB-64MB), Harvest with Recursive Doubling maintains a steady advantage over the ring algorithm for lower reconfiguration delays, achieving up to a 1.98× speedup at 10ns. However, as the reconfiguration delay increases past 10μs, the static ring algorithm begins to show better performance.

[5] All heatmaps (except Figure 12, 13, 16, 17) use whitespace to indicate a speedup of exactly 1×. The representation in Figure 13 differs because it compares two different collective algorithms, namely Ring and Recursive Doubling with Harvest-derived topology schedules. When optimizing the ring algorithm, Harvest correctly identifies the static ring topology as optimal. Consequently, no additional speedup is obtained from topology reconfiguration, and the Ring algorithm remains at exactly 1× across the entire spectrum. We omit this plot for brevity.

Table 1: Effect of reconfiguration delay on Recursive Doubling AllGather synthesis. Larger reconfiguration delay (α_r) reduces the number of reconfigurations at the expense of increased communication cost. Example with message size 4MB, $n=64$ nodes, $\alpha=500$ns setup delay, and $\delta=50$ns propagation delay.

α_r	# Reconfigurations	Completion time (ns)
10	5	44637.97
100	5	45087.97
1,000	4	49293.26
10,000	2	73357.95
	Ring	75937.68
	Recursive Doubling	131979.12

As the message size increases further into the gigabyte range, the Ring algorithm generally outperforms Recursive Doubling Harvest's schedule at higher reconfiguration delays. Notably, for large 4GB message size, Recursive Doubling Harvest's schedule is able to match ring's performance for reconfiguration delays up to 1μs.

When compared against both Ring and Recursive Doubling on static topology, Recursive Doubling with Harvest's schedule continues to outperform for low reconfiguration delays and smaller message sizes. These consistent gains across the entire spectrum further strengthen our findings from hardware emulation, confirming the interplay between communication steps, congestion, and topology reconfiguration overheads.

Consider the case with message size $m = 4$MB. With $\alpha_r = 10ns$, Harvest synthesizes a schedule with 5 reconfigurations, essentially reconfiguring after every step. For $n=64$, there are 6 steps in total for AllGather. The total completion time can be expressed as,

$$6\cdot\alpha+\sum_{i=1}^{6}\delta\cdot 2^{6-i}+\beta\cdot m\cdot\sum_{i=1}^{6}\frac{1}{2}\cdot 2^{6-i}+5\cdot\alpha_r = 44637.97ns$$

The completion time closely matches our results from simulations. Similarly, for $\alpha_r = 1000$ns, we obtain a schedule with 4 reconfigurations, and a completion time of 49293.26ns. As the reconfiguration delay increases, Harvest synthesizes schedules with fewer reconfigurations trading off against increased communication cost. For $\alpha_r = 10000$ns, the schedule contains only 2 reconfigurations, and the completion time increases to 73357.95ns.

In contrast, Ring AllGather on a static ring topology completes in 75937.68ns, and Recursive Doubling on a static topology completes in 131979.12ns. Ring algorithm mainly suffers due to excessive number of steps, whereas Recursive Doubling suffers due to excessive congestion incurred due to multi-hop routing and bandwidth sharing. The results highlight the tradeoff between reconfiguration overhead and communication cost, and the benefits of topology reconfiguration for algorithms with fewer communication steps such as Recursive Doubling. At the same time, the example also highlights the potential for topology reconfigurations to outperform both Ring as well as Recursive Doubling on static topologies.

C.3 MISOCP & Maximum Concurrent Flow

From a topology perspective, this paper generalizes the classic dynamic demand-aware network design problem to account for multi-hop routing and splittable flow. Prior works [13, 46] also account for reconfiguration delay in demand-aware networks, but restrict communication to direct single-hop transfers, often relying on Birkhoff-von Neumann (BvN)-style schedules. Such restrictions significantly limit both the space of feasible schedules and the potential benefits of topology reconfiguration. Eclipse [13], for example, expresses the optimization objective as follows (adapted to the notations used in §3):

$$\begin{aligned} \text{maximize} \quad & \left\| \min\left(\textstyle\sum_{i=1}^{s}\frac{m_i}{b}\cdot\mathcal{M}_i, \mathcal{M}\right)\right\|_1 \\ \text{s.t.} \quad & \tfrac{m_1}{b}+\tfrac{m_2}{b}+\cdots+\tfrac{m_s}{b}+s\cdot\alpha_r \le W, \\ & s\in\mathbb{N}, \quad \tfrac{m_i}{b}\ge 0\,\forall i\in\{1,2,\ldots,s\}. \end{aligned}$$

Here, W denotes the scheduling window, and $\frac{m_i}{b}$ represents the amount of time allocated to communication pattern $\mathcal{M}_i$. Given a demand matrix $\mathcal{M}$, the objective maximizes demand coverage, i.e., the total amount of demand satisfied within the scheduling window, while trading it off against the reconfiguration overhead incurred by introducing additional communication patterns. In the context of collective communication, the decomposition into communication patterns is fixed and determined by the collective algorithm itself. Furthermore, because communication is restricted to direct single-hop transfers, the optimization is limited to selecting a schedule over these predefined patterns. Recall that m_i is the amount of data transmitted in step i of the collective, and $\frac{m_i}{b}$ is the time required to complete the transfer when the full bandwidth b is available, i.e., under direct single-hop communication. As a result, switching schedules derived from this formulation closely resemble BvN decompositions of the collective, since no alternative topologies or routing strategies can be exploited.

In contrast, our formulation permits both multi-hop routing and splittable flow, enabling a substantially richer space of feasible schedules while explicitly accounting for reconfiguration delays. Harvest removes the restriction imposed by the minimum number of matchings, and consequently reconfigurations, required to satisfy demand under a BvN decomposition. Instead, it allows traffic to traverse multi-hop paths in the topology and captures collective completion time through maximum concurrent flow within each interval.

Eclipse [13] also considers multi-hop routing, but only across multiple timeslots, thereby implicitly requiring additional reconfigurations. Furthermore, its formulation does not explicitly model the data dependencies between communication steps that arise in collective algorithms. Opera [48] considers multi-hop routing within a timeslot to favor short flows under skewed flow-size distributions. However, Opera does not account for reconfiguration delay and does not consider the data dependencies between communication steps. In contrast, Harvest incorporates these dependencies directly into the optimization, enabling it to jointly optimize communication, routing, and topology evolution.

The connection between Harvest and prior single-hop formulations becomes clearer when viewing completion time through the lens of maximum concurrent flow. This perspective naturally generalizes prior formulations to multi-hop settings while preserving the data dependencies inherent in collective communication algorithms. For instance, Eclipse counts the transmission delay of a matching

as $\frac{m_i}{b}$ for m_i amount of data with bandwidth b, thereby enforcing single-hop communication. In contrast, we model the transmission delay as $\frac{m_i}{b} \cdot \frac{1}{\theta}$, where $\frac{1}{\theta}$ captures the slowdown due to bandwidth sharing arising from multi-hop forwarding. Consequently, Harvest can jointly reason about topology selection, routing, congestion, and reconfiguration overhead within a unified optimization framework. Moreover, unlike prior approaches, Harvest can determine when topology reconfiguration is beneficial and when a static topology is preferable. As reconfiguration delays increase, the optimizer naturally converges toward static schedules; conversely, when reconfiguration delays are sufficiently small, it can exploit dynamic topologies to reduce collective completion time.

C.4 Limitations, Open Questions & Future Research Directions

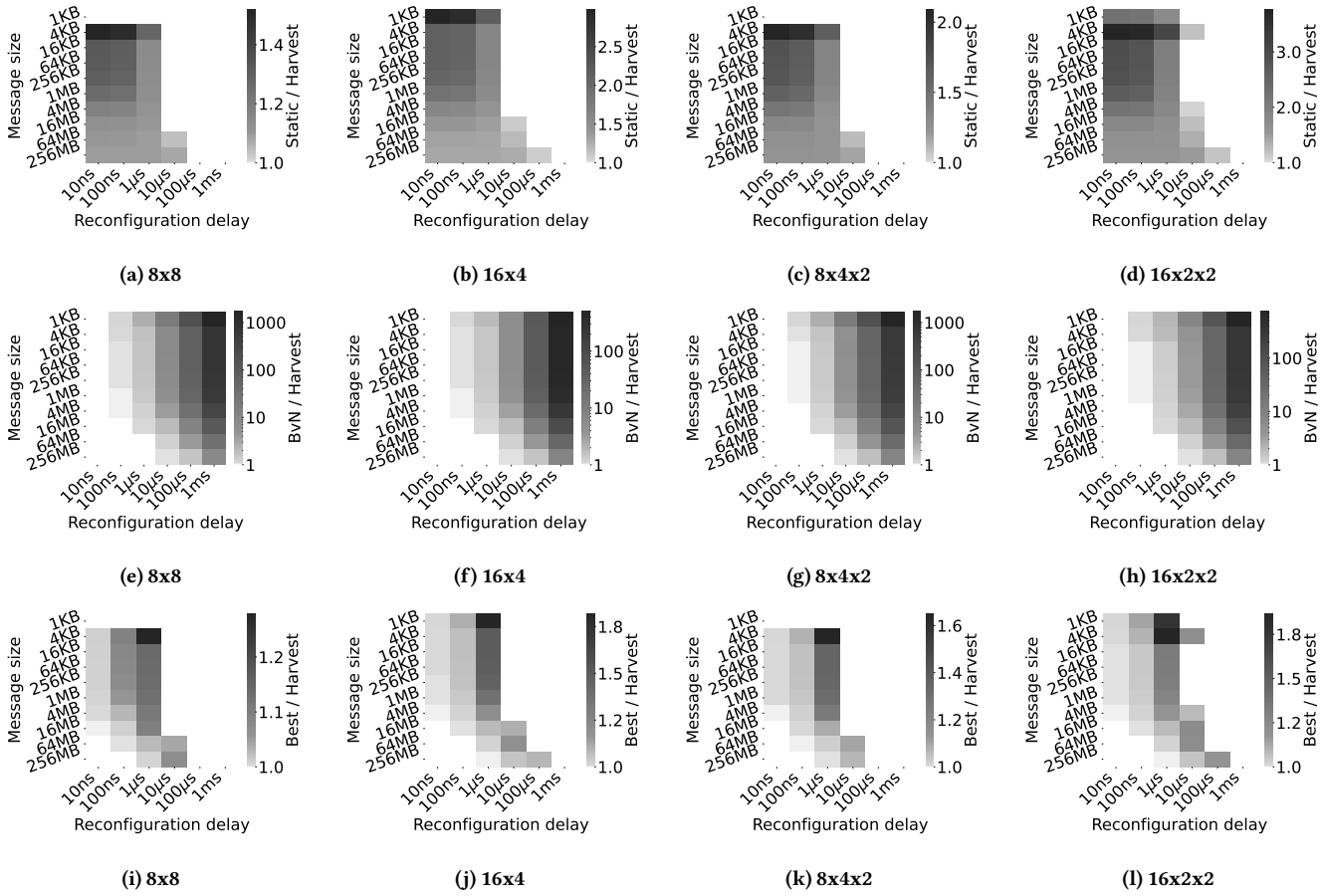
Collective algorithms: While Harvest can synthesize topology schedules that minimize the completion time of a given collective algorithm, the resulting schedule is not necessarily globally optimal across all collective algorithms. Given a pool of existing algorithms, one can already use Harvest to derive an optimized topology schedule for each algorithm, estimate the corresponding completion time using our cost model, and then select the algorithm-schedule pair with the lowest completion time. This observation raises a broader question: can we design new collective algorithms that are inherently topology-aware and optimized specifically for dynamic topologies? More fundamentally, can the principles underlying Harvest be used not only to optimize topology schedules for existing algorithms, but also to guide the synthesis of new communication patterns that better exploit reconfigurable interconnects? We leave this as an interesting direction for future research.

Multi-tenant and shared infrastructure: We have focused on optimizing collective communication in settings where GPUs communicate over a circuit-switched interconnect without cross traffic. This setting can be viewed from two perspectives. First, a multi-tenant scale-up domain in which no two jobs share the same GPU hardware or network I/O. In this case, the photonic interconnect must support independent reconfiguration of subsets of ports without affecting the connectivity of other ports. For instance, Sirius [9] performs passive switching and allows independent tuning of transceiver wavelengths, while interconnects built from electro-optic switches allow each individual MZI to be controlled by an independent electrical pulse. As a result, jobs remain fully isolated not only at the GPU level but also within the photonic interconnect, since circuit switching and reconfiguration affect only the ports associated with a given job. Second, a single-tenant scale-up domain in which a single job has exclusive access to all GPUs and interconnect resources. In this setting, the photonic interconnect can be reconfigured freely without concern for interference from other jobs. This setting also applies to switching technologies that affect all ports of a switch during reconfiguration events.

A limitation of the current work is that we do not consider multi-tenant environments in which reconfiguration impacts all ports of a switch, potentially affecting multiple jobs simultaneously. This setting opens several interesting research questions. Can we derive global topology schedules that jointly optimize the completion times of all active jobs rather than optimizing each job independently? Can topology reconfiguration decisions be coordinated across jobs to balance reconfiguration overhead and congestion? More broadly, can Harvest be extended from a per-collective optimizer into a multi-job topology scheduler that jointly reasons about communication patterns, topology, and reconfiguration events across multiple concurrent workloads? We leave these questions to future work.

End-to-end workload: A training job not only consists of collective communication such as AllReduce, All-to-All, but also includes point-to-point communications e.g., in Hybrid Tensor-Pipeline parallelism. One limitation of the current work is that we do not consider point-to-point communications, that may also potentially overlap AllReduce events in certain scenarios. We believe this can be tackled in three ways: *(i)* a hybrid architecture with electrical interconnect on the side that exclusively serves point-to-point communication, and a photonic interconnect that serves collective communication. In this case, the point-to-point communication can be scheduled on the electrical interconnect to avoid interference with collective communication on the photonic interconnect, *(ii)* dedicating one outgoing and one incoming link from each GPU among the available degree d, to create a static ring that exclusively serves point-to-point communications, *(iii)* by extending Harvest to jointly optimize topology schedules for both collective and point-to-point communication events, while accounting for potential overlaps and interference between them. We leave this as an interesting direction for future research.

Integration with scale-out networks: Two broad classes of scale-out networks have emerged over the last decade: *(i)* traditional RoCE/InfiniBand networks [24, 28, 58], e.g., multi-GPU servers connected through a fat-tree topology, and *(ii)* emerging optical interconnects, e.g., TPU cubes connected by reconfigurable optical switches [33]. We primarily focus on scale-up networks in this paper. A natural question is how to integrate Harvest with scale-out networks. One approach is to treat the scale-up network as a black box and apply Harvest to optimize collective communication within each server, while relying on existing approaches to optimize communication across servers. A broader direction is to extend Harvest to jointly optimize communication within and across servers while accounting for the interaction between the two levels of the network hierarchy. For instance, collective communication spanning multiple servers is often decomposed into hierarchical algorithms consisting of intra- and inter-server communication phases, where Harvest could be applied to optimize each level of the hierarchy. Applying Harvest to scale-out settings also presents new opportunities and research challenges. Compared to scale-up networks, scale-out deployments exhibit larger propagation delays, broader synchronization domains, more distributed control planes, and multi-tenancy all of which can influence the coordination of topology reconfigurations across servers. Nevertheless, these effects are naturally captured by the reconfiguration-delay parameter in Harvest's model. As advances in photonic switching technologies, control-plane design, transceiver architectures, and hardware–software co-design continue to reduce these overheads, the optimization framework can directly quantify and exploit the resulting benefits. We believe that exploring the integration of Harvest with well-established techniques for spineless scale-out networks, such as static random graph topologies [10, 69, 70] and traffic-engineering solutions [57],



(a) 8x8 (b) 16x4 (c) 8x4x2 (d) 16x2x2

(e) 8x8 (f) 16x4 (g) 8x4x2 (h) 16x2x2

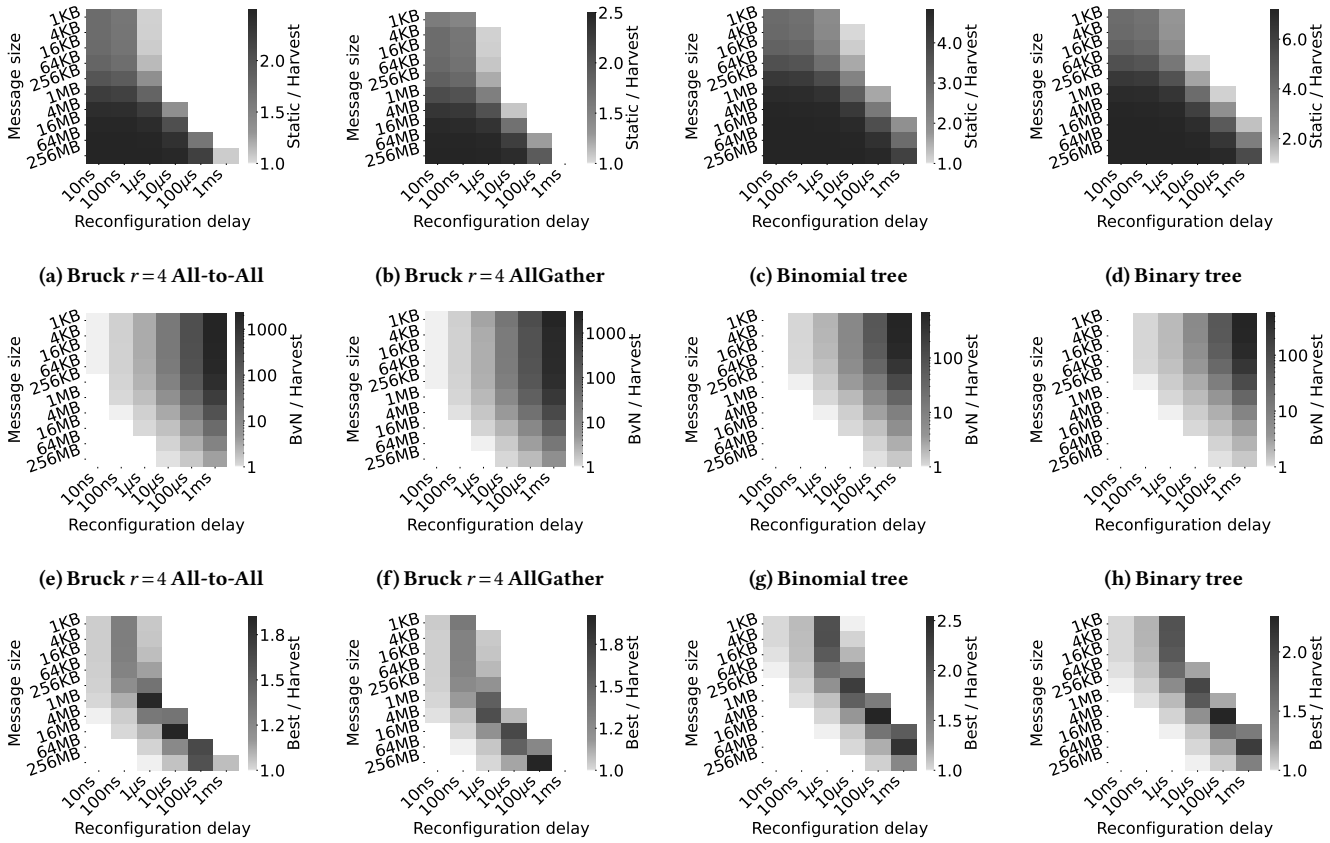
(i) 8x8 (j) 16x4 (k) 8x4x2 (l) 16x2x2

Figure 14: [Numerical optimization] Heatmaps showing the speedup in collective completion time achieved by Harvest relative to static Torus, BvN (reconfiguring at every step), and best among both, for Swing AllReduce for various Torus configurations. We use multi-port Swing, along with mirroring for all multi-dimensional topologies.

as well as extending it to hierarchical and large-scale reconfigurable networks, is a promising direction for future research.

End-to-end reconfiguration delay: We model the reconfiguration delay as a fixed cost α_r incurred for each topology change. Harvest treats this as a parameter when determining switching schedules. In practice, α_r not only includes the optical rise/fall time, but also electrical (for electro-optic MZIs) or thermal (for thermo-optic devices) rise/fall times, control-plane delay, synchronization delays, transceiver locking overheads after reconfiguration, and potential overheads from the software stack, e.g., to adjust routing. There is active research across all of these components, including faster photonic switching technologies, low-latency control planes, improved synchronization mechanisms, and tighter hardware–software co-design [9, 17, 19, 23]. As a result, the end-to-end reconfiguration delay is expected to decrease over time. Since Harvest explicitly models α_r as a parameter, it can naturally adapt to future advances in reconfigurable network technologies and quantify the benefits of reduced reconfiguration overheads on collective communication performance.

D Additional Results

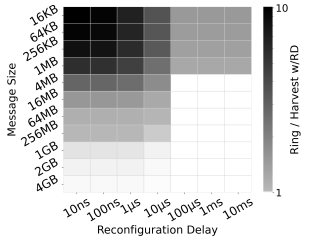


(a) Bruck $r=4$ All-to-All (b) Bruck $r=4$ AllGather (c) Binomial tree (d) Binary tree

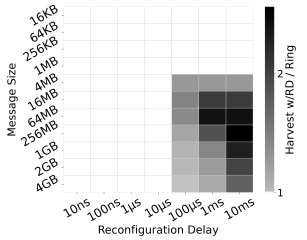
(e) Bruck $r=4$ All-to-All (f) Bruck $r=4$ AllGather (g) Binomial tree (h) Binary tree

(i) Bruck $r=4$ All-to-All (j) Bruck $r=4$ AllGather (k) Binomial tree (l) Binary tree

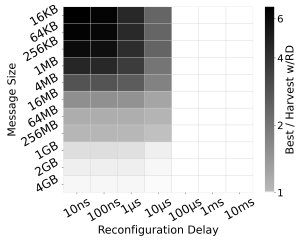
Figure 15: [Numerical optimization] Heatmaps illustrating the speedup in collective completion time achieved by Harvest compared to static topologies, BvN schedules (reconfigured at every step), and the best of the two. Results are shown for Bruck's algorithm for All-to-All and AllGather, as well as binomial tree and binary tree algorithms for broadcast. In the case of Bruck's algorithm with $r=4$, each node performs four send/receive operations per step.



(a) When Recursive Doubling AllReduce with Harvest outperforms Ring AllReduce on a static topology.

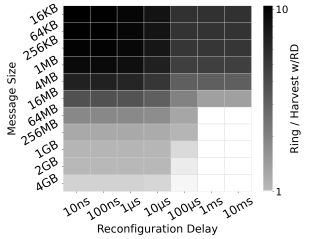


(b) When Ring AllReduce on a static topology outperforms Recursive Doubling AllReduce with Harvest.

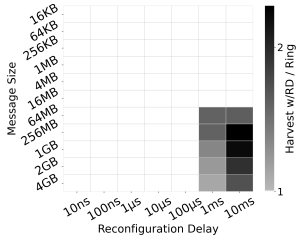


(c) When Recursive Doubling AllReduce with Harvest outperforms the best among Recursive Doubling and Ring on a static ring topology.

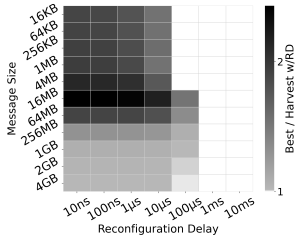
Figure 16: [Simulations] Ring vs Recursive Doubling for AllReduce with link bandwidth=800Gbps, α=500ns, and δ=500ns. White space indicates a speedup[5] less than or equal to 1×.



(a) When Recursive Doubling AllReduce with Harvest outperforms Ring AllReduce on a static topology.



(b) When Ring AllReduce on a static topology outperforms Recursive Doubling AllReduce with Harvest.



(c) When Recursive Doubling AllReduce with Harvest outperforms the best among Recursive Doubling and Ring on a static ring topology.

Figure 17: [Simulations] Ring vs Recursive Doubling for AllReduce with link bandwidth=800Gbps, α=10000ns, and δ=500ns. White space indicates a speedup[5] less than or equal to 1×.