VEDRFOLNIR: RDMA Network Performance Anomalies Diagnosis in Collective Communications

Yuxuan Chen[1,2], Menghao Zhang[1,2], Xiheng Li[1], Fangzheng Jiao[1], Xiao Li[2], Jiaxun Huang[1], Shicheng Wang[2], Chunming Hu[1]

[1]School of Software, Beihang University [2]State Key Laboratory of Internet Architecture, Tsinghua University

***Abstract*—Collective communication has become increasingly crucial as large language models rapidly evolve, leveraging RDMA networks for inter-node connectivity to achieve high throughput and low latency. However, RDMA inevitably face Network Performance Anomalies (NPAs) due to mechanisms like line-rate start and PFC flow control. Collective communication usually utilizes multiple flows simultaneously, which puts new challenges for efficient diagnosis of NPAs. Existing works fall short in analyzing co-flow patterns, which cannot diagnose RDMA NPAs in collective communication accurately and efficiently. To address this, we propose VEDRFOLNIR, an accurate and efficient diagnosis system for RDMA NPAs in collective communication. VEDRFOLNIR (1) constructs waiting graphs through algorithm decomposition, (2) adaptively detects anomalies while efficiently collecting diagnostic data, and (3) precisely analyzes performance bottlenecks and identifies root causes. We implement an open-source prototype of VEDRFOLNIR and evaluation results show that VEDRFOLNIR can achieve accurate diagnosis results with low overhead.**

I. INTRODUCTION

With the rapid growth of large language models (LLMs), collective communication has become an essential component that determines the efficiency of data exchange and parallel computation among GPUs. RDMA (Remote Direct Memory Access), with its high bandwidth and low latency characteristics, is widely used in data transmission for collective communication. As data center network infrastructure evolves, more clusters are transitioning from InfiniBand (IB) networks to more versatile high-speed Ethernet, commonly using the RDMA over Converged Ethernet version 2 (RoCEv2) protocol [1] for data exchange [2]–[5], especially in areas like LLM training [6]–[12]. For example, Meta's 16,000-GPU cluster [6] utilizes RoCEv2 for LLM training.

To ensure high performance, in RoCEv2, PFC (Priority Flow Control) [13] is typically used to achieve a lossless network, along with congestion control algorithms such as DCQCN [14] and Swift [15] to manage flow control and ensure transmission efficiency. Complex collective communication operations performed on this basis introduce additional complexity for diagnosing performance anomalies. On one hand, collective communication is dynamic during execution, with data dependencies between flows and potential changes in the flows across different stages. On the other hand, RDMA mechanisms such as PFC flow control and line-rate start make network congestion more frequent and complex. As a result, collective communication is prone to encounter various types of network anomalies (e.g., flow contention, PFC backpressure), leading to degraded performance.

Existing research faces significant challenges in accurately and efficiently diagnosing RDMA Network Performance Anomalies (NPAs) in collective communication. Firstly, existing diagnostic approaches predominantly focus on single-flow level analysis (e.g., Hawkeye [16], [17], SpiderMon [18], Sonata [19], and *Flow [20]), neglecting complex communication paradigms such as collective communication. In contrast to the static nature of single-flow communication, flows in collective communication are dynamic and may evolve over time, potentially having dependencies on other flows as the communication progresses. For instance, in the Halving and Doubling [21] algorithm, the destination of a flow can change multiple times, and such changes depend on the data transmitted by other flows. Current diagnostic methods fail to capture these intricate dynamic behaviors and the provenance between flows within the collective communication, rendering them inadequate for accurately analyzing RDMA NPAs in collective communication. Second, due to the involvement of numerous nodes and flows in collective communication, achieving accurate diagnosis with minimal overhead presents another significant challenge. Existing approaches collect telemetry data across all switches (e.g., NetSight [22] and PINT [23]) or at all time (e.g., Hawkeye [16], [17]). While ensuring accuracy, these methods introduce substantial communication and computational overhead, leading to potential performance bottlenecks. Thus, minimizing the impact on network performance while maintaining diagnostic accuracy remains a critical issue that requires significant attention.

To address the above challenges, we propose VEDRFOLNIR, an accurate and efficient RDMA network performance anomaly diagnosis system for collective communication. In terms of data collection, VEDRFOLNIR monitors collective communication performance information on the host side and performs network tracing when performance anomalies occur. For diagnostics, VEDRFOLNIR primarily utilizes the waiting relationships between flows to construct a waiting relationship graph for analyzing performance bottlenecks in collective communication, and builds network provenance graphs for root cause analysis and flow impact assessment. VEDRFOLNIR is built with the following three key modules. First, VEDRFOLNIR characterizes collective communication performance by decomposing the communication algorithm into steps and defining a waiting relationship graph (§III-B).

Second, VEDRFOLNIR employs a step-aware adaptive detection mechanism to reduce redundant data collection (§III-C). Third, VEDRFOLNIR uses the collected data to construct waiting and network provenance graphs for comprehensive root cause analysis (§III-D). We implement a prototype on NS3, which is open-source at GitHub [24]. Evaluations show that VEDRFOLNIR can accurately diagnose collective communication NPAs while reducing overhead up to 90% compared to Hawkeye or full polling. The additional CPU overhead caused by VEDRFOLNIR is negligible.

Contributions. We analyze the complexity that collective communication introduces to RDMA NPA diagnosis (§II) and propose VEDRFOLNIR, an accurate and efficient RDMA NPA diagnosis system for collective communication (§III). We implement a prototype using NS-3 simulation and experimental results show that VEDRFOLNIR can accurately diagnose performance issues while reducing substantial overhead (§IV).

II. BACKGOUND AND MOTIVATION

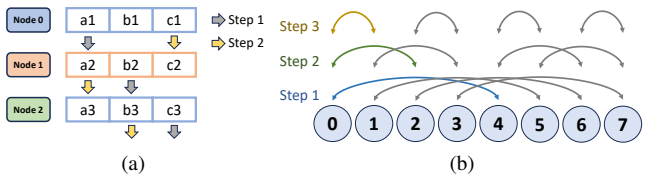
Collective communication adds significant complexity to the diagnosis of NPAs, making existing work inadequate for effective diagnosis. VEDRFOLNIR focuses on diagnosing collective communication performance anomalies resulted from RDMA NPAs, and aims to answer the following key question: *what network performance anomaly is responsible for the degraded collective communication?*

A. Network Complexity of Collective Communication

The network complexity of collective communication primarily consists of two main aspects:

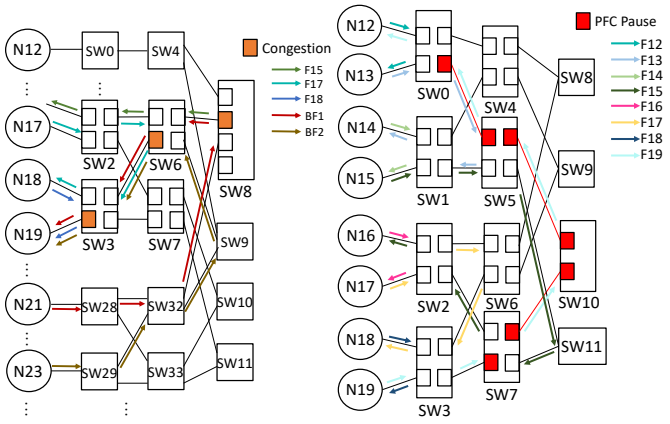
Co-flow patten in collective communication. Collective communication typically employs specific algorithms (e.g., Ring, Halving and Doubling) to enhance communication efficiency (Figure 1), thereby introducing additional complexity for network anomaly diagnosis. First, unlike single-flow applications, collective communication requires collaborative efforts from multiple flows originating at multiple nodes. Second, collective communication changes dynamically during execution. The specific flows involved differ across distinct algorithmic phases. Third, there are data dependencies between flows. Host B may receive the data transmitted by flow A before it can start flow B to transmit the data to other nodes.

Complexity introduced by RDMA. Unlike traditional TCP networks, RDMA possesses characteristics such as line-rate start and PFC mechanisms that are more prone to cause network congestion. First, in RoCEv2, RDMA relies on PFC flow control to achieve a lossless network, which is fundamental to its superior transmission performance. With PFC, when a switch queue exceeds a certain threshold, it sends a PAUSE frame to its upstream devices. After that, the upstream device stops transmitting on the link until it receives a RESUME frame. However, PFC has a back-pressure effect when it encounters network congestion, cascading PAUSE frames upstream and causing widespread network congestion. Second, RDMA does not have a slow start process like TCP, but instead sends at line rate initially, which makes network congestion more frequent and transient given the shallow buffers of commercial RDMA switches.



(a) (b)

Fig. 1. Examples of collective communication algorithms. (a) Ring. (b) Halving and Doubling.



(a) (b)

Fig. 2. Anomaly cases. (a) Flow contention. (b) PFC storm.

B. NPAs in Collective Communication

Network performance anomalies are complex and diverse. Figure 2 illustrates two fundamental types of anomalies: 1) Flow contention. In large-scale clusters, many tasks share the same infrastructure, making flow contention in the network inevitable. Figure 2a shows an instance of collective communication using the Ring algorithm among 8 nodes (Nodes 12-19), accompanied by two background flows (BF1 and BF2). Due to path overlap, the background flows collide with the collective communication flows, causing congestion at multiple switch ports and degrading transmission performance. 2) PFC storms. A PFC storm refers to the sustained injection of PFC frames, often caused by hardware bugs, which triggers widespread propagation of PFC and leads to network-wide performance anomalies. Figure 2b displays the flows of an 8-node collective communication. Persistent PFC injection occurs at a port on Switch 0, causing Flow 12 to be halted across multiple switches, severely impacting performance.

In real network environments, flow contention and PFC propagation often occur in combination. Depending on their different causes or characteristics, some studies [17], [18] have also summarized the following other common anomalies: 1) Load imbalance. Due to misjudgments in load balancing policies (e.g., ECMP), traffic from collective communication or other flows may not be distributed ideally across the network topology, leading to contention between different flows. 2) Loops. Due to asynchrony among switches during network reconfiguration, some switches may form a forwarding loop. A flow may be forwarded from a switch port and, after

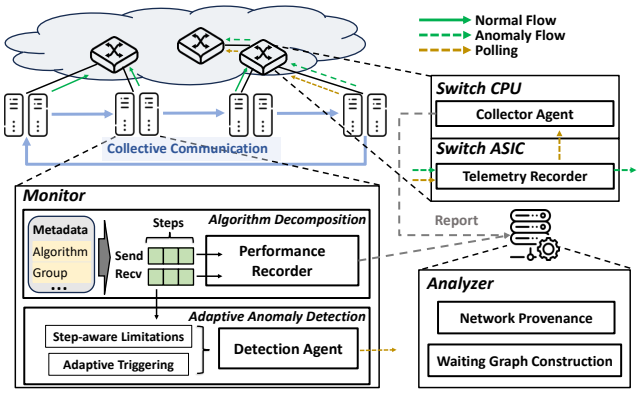


Fig. 3. VEDRFOLNIR framework.

multiple hops, return to the same port. 3) PFC backpressure by flow contention. PFC propagation caused by culprit flows can even affect flows that do not share a forwarding path with the culprits [14], [17], highlighting the particularity of diagnosing PFC-related anomalies. 4) PFC deadlock. When PFC propagation forms a cycle, all flows at the cyclic locations are halted, resulting in a deadlock.

C. Existing Solutions Fall Short

Lack of analysis capability for co-flow patterns. Existing work [16]–[20], [25], [26] primarily focuses on single-flow analysis and overlooks more complex paradigms such as collective communication, leading to insufficient co-flow analysis. Due to the complexity of collective communication, an anomaly in one flow is likely to delay the start time of another flow that depends on it. Therefore, diagnosing collective communication performance must consider multiple flows simultaneously and analyze them based on their interrelationships. However, existing host-based approaches [27]–[29] mainly focus on inferring anomalies in the network, while switch-side methods [18]–[20], [25] primarily use programmable switches to directly collect network data. These approaches focus solely on the network, with analysis conducted at the flow or packet level, without considering potential relationships at the application level.

Telemetry collection incurs significant redundant overhead. Some works [22], [23] collect telemetry information from all switches, ensuring accuracy but introducing substantial communication or analysis overhead. Other works [18], [30], while analyzing queue contention on only a subset of switches, fail to account for the PFC chaining propagation characteristics in RDMA, resulting in inaccurate information. Hawkeye [17] achieves single-flow-level accuracy and efficiency by devising a PFC tracing method, but it still lacks design considerations for collective communication. For example, when Hawkeye is applied to collective communication anomaly detection, its mechanically set anomaly detection trigger mechanism can cause repeated triggers at multiple communication nodes within a short period, collecting redundant data and introducing significant overhead.

Lack of precise root cause diagnosis at the collective communication level. Existing diagnostic approaches can be categorized into three types: flow statistics-based [19], [20], [25], [26], [29], [31], queue measurement-based [30], [32], and provenance-based [16]–[18], [33]. First, methods based on flow statistics and queue measurements contribute primarily by statistical analysis of flows or queues, but they struggle with precise anomaly localization (e.g., detecting switches related to loops or PFC deadlocks). Second, provenance-based methods address the aforementioned limitations, but still fail to consider the complexities of collective communication scenarios, unable to analyze the performance bottlenecks and locate the root causes.

III. VEDRFOLNIR DESIGN

A. Overview

To diagnose collective communication performance anomalies resulted from RDMA NPAs accurately and efficiently, VEDRFOLNIR consists of the following key steps. 1) To capture the dynamics of collective communication and the data dependencies of flows, VEDRFOLNIR algorithmically decomposes collective communications at a per-step granularity and constructs a directed graph to describe the entire collective communication process using the waiting relationships of flows (§III-B). 2) To ensure accuracy with low overhead, VEDRFOLNIR employs a step-aware adaptive mechanism to limit unnecessary detection triggers, and leverages coordinated data collection through host monitoring and network telemetry to efficiently and accurately support diagnostics of collective communication (§III-C). 3) To enable effective root cause analysis,VEDRFOLNIR constructs multi-level graphs for bottleneck analysis and root cause localization, and evaluates the primary contributing flows to flow contention (§III-D).

Figure 3 shows the overall architecture of VEDRFOLNIR. First, the monitor (located on each host) pre-executes algorithmic decomposition and monitors performance metrics of the flows by steps during collective communication in real time. These metrics are reported to the analyzer. Second, when performance anomalies occur, VEDRFOLNIR adaptively adjusts the trigger for anomaly detection in monitors. Monitors send notification packets based on their wait relationships to control the frequency of anomaly detection. During detection, the collective communication hosts initiate the process by transmitting polling query packets. These packets trigger the collection of telemetry data from all relevant switches involved in the anomaly. The collected telemetry data is then reported to the analyzer. Finally, the analyzer constructs both a waiting graph and network provenance graphs. Through comprehensive analysis of these graphs, VEDRFOLNIR provides structured diagnostic results.

B. Algorithm Decomposition and Graphical Description

Algorithm decomposition. VEDRFOLNIR decomposes the algorithm into multiple steps, describing the distinct phases of various flows in a collective communication. For a flow originating from a particular node, the transmitted data chunk or the destination address changes between each step. For example, in Figure 1a demonstrating the Ring algorithm, the data chunk transmitted by flow F0 (originating from node

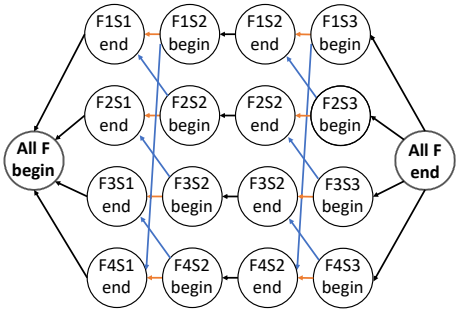


Fig. 4. Waiting graph of a Ring reduce-scatter communication.

0) changes from chunk A to chunk C during the transition from step 1 to step 2. Similarly, Figure 1b shows that in the Halving and Doubling algorithm, the destination of flow F0 shifts from node 4 in step 1 to node 2 in step 2. When either of these two transitions occurs, we say that F0 has gone through a step. Other collective communication algorithms can also be decomposed using a similar approach. The fundamental idea is to break them down based on the data dependencies between flows, where each step of a flow serves as a unit that either waits for or is waited for by other flows. We do not implement an automatic decomposition method applicable to all collective communication algorithms; instead, the steps of the collective communication algorithm need to be predefined prior to execution.

Waiting graph definition. After the step splitting of the flows, we construct a waiting graph to describe the collective communication process. Taking the Ring reduce-scatter illustrated in Figure 4 as an example, we define 1) vertices: the start or end of each step of each flow. F_iS_j represents step j of flow i. Specifically, the end of the final step for all flows serves as the graph's source, while the start of the first step for all flows serves as the sink. 2) directed edges: indicating waiting dependencies between vertices. For example, due to data dependency, the start of F2S2 needs to wait for the end of F1S1, which is represented by a blue edge; meanwhile, the start of F2S2 needs to wait for the end of the previous step F2S1, which is represented by an orange edge; the relationship between the start and the end of step execution within F2S2 itself is represented by a dark edge. 3) weight: indicating the waiting time. The weight of the light-colored (i.e., blue, orange) edge is 0, and the weight of the dark edge is the execution time of this step of the flow.

C. Logging Workflow

VEDRFOLNIR monitors collective communication performance metrics on the host side (§III-C1). Upon detecting performance anomalies, VEDRFOLNIR adaptively triggers network telemetry (§III-C2) and retrieves network information recorded in the switches (§III-C3), enabling subsequent anomaly diagnosis.

1) Performance Monitoring: On the host side, VEDRFOLNIR monitors the waiting state and records performance information.

TABLE I
WAITING STATE DETERMINATION

Index Comparison (state)	Meaning
Send Steps == Recv Steps (waiting)	The next send step waits for the current receive to complete.
Send Steps < Recv Steps (non-waiting)	Do not wait for others, execute the next send step as soon as current step is finished.

Waiting status awareness. As discussed in Section III-B, VEDRFOLNIR decomposes the collective communication process using the waiting relationships of flows. However, in actual execution, the waiting relationship occurs selectively. As shown in Figure 4, the start of F1S2 waits for both the end of F1S1 and F4S1, but due to the sequence of the two events, F1S2 actually waits for only one of them. To enable real-time awareness of the waiting status, VEDRFOLNIR's monitor maintains two queues: a Send Step Queue (SSQ) and a Receive Step Queue (RSQ). During the algorithm decomposition, the flow originating from the current host is divided into multiple steps. The target of each sending step is sequentially enqueued into the SSQ. Correspondingly, the source of the required data for each receiving step (i.e., the source of flow which the next send step waits) is sequentially enqueued into the RSQ. By tracking the index of the currently active step within these queues, VEDRFOLNIR can determine the waiting status of each flow (Table I).

Performance recording. To construct the waiting graph and analyze performance bottlenecks in collective communication, the monitor on each host collects performance metrics for its local flows. Upon completion of each flow step, the host reports its 5-tuple, data volume transferred, start time, end time, and the source host of the flow it is waiting for.

2) Step-Aware Adaptive Anomaly Detection: To address the issues of high redundant overhead and lack of multi-flow analysis in previous work (e.g., Hawkeye [16], [17] and SpiderMon [18]), VEDRFOLNIR primarily improves in two aspects: 1) Step-aware detection trigger constraints. Under ideal conditions, the execution time of a single step in a collective communication flow is typically short (e.g., approximately 30 ms for transmitting 300 MB of data at 100 Gbps). However, our testing indicates that, in commonplace flow contention, the communication flow execution time can easily experience a two-fold or greater delay. Previous approaches lack temporal management for detection triggers, repeatedly collecting telemetry within the same step and thus generating substantial redundant overhead. Hence, step-level detection granularity is more appropriate. 2) Adaptive detection opportunity allocation. As defined by the waiting graph (§III-B), the overall performance (i.e., total execution time) of collective communication is determined by its critical path, and the affected steps on this path tend to be prolonged. It is appropriate to allocate more detection opportunities to the affected steps based on their wait-for dependencies.

For detection triggering, VEDRFOLNIR primarily makes decisions dynamically based on network topology and user

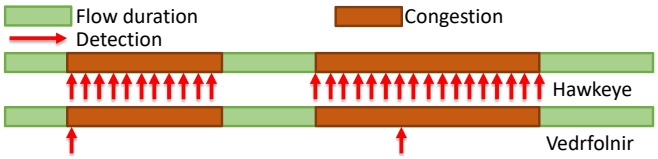


Fig. 5. Examples of detection triggers.

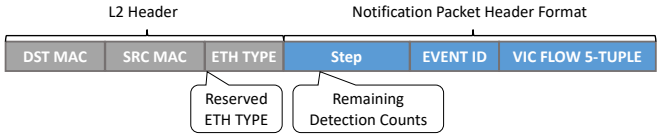


Fig. 6. Notification Packet Format.

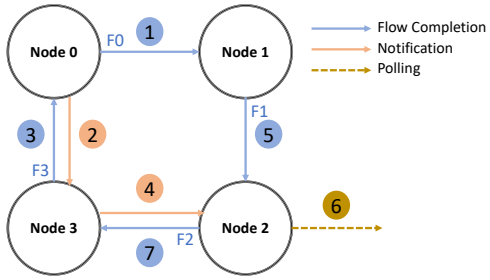


Fig. 7. Example of the adaptive mechanism. The numbers represent the order of events.

configurations, specifically including the following two aspects. First, host-side triggering based on RTT thresholds. Unlike Hawkeye's fixed RTT threshold settings, VEDRFOLNIR recalculates RTT thresholds based on network topology before each step initiation. This avoids inappropriate RTT settings caused by changes in flow paths (e.g., the algorithm in Figure 1b) or failure to distinguish between different flows. When the monitor detects that the RTT surpasses the threshold, an anomaly detection is triggered. Second, triggering constraints. As shown in Figure 5, to avoid redundancy, VEDRFOLNIR uses configurable parameters to define the number of anomaly detection triggers for each step of a collective communication flow. Based on the estimated FCT (Flow Completion Time, which can be calculated from the network topology or set empirically), it sets a triggering time interval limit. This ensures that triggering opportunities are evenly distributed, thus ensuring coverage as comprehensively as possible.

For the adaptive mechanism, VEDRFOLNIR mainly achieves adaptation by allowing monitors to observe the waiting states of their respective flows. Specifically, when a certain step of a flow is completed, its monitor sends a notification packet to inform the monitor of the flow waiting for it (if any), transferring its remaining detection opportunities. The notification packet contains detection count information (Figure 6) and is assigned the highest priority to avoid being affected by network congestion. An example is illustrated in Figure 7. F0 completes first, and node 0 transfers its detection opportunities to host 3 by sending a notification packet. Subsequently, F3 completes, and the opportunities are further transferred to host 2. Before F2 completes, F1 finishes earlier. As a result, the slowest host, host 2, obtains the most detection opportunities. In this way, VEDRFOLNIR strives to collect telemetry data for

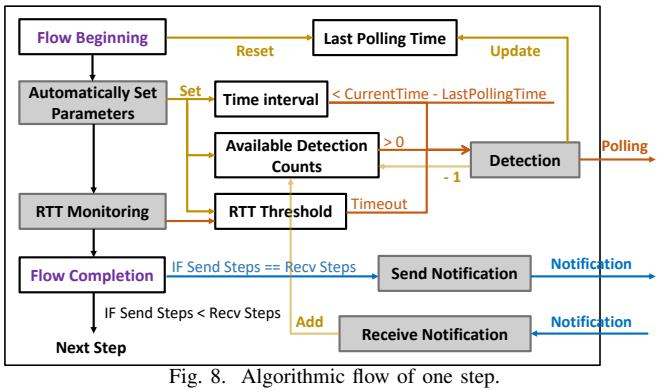


Fig. 8. Algorithmic flow of one step.

the relatively slowest flow at each step, while theoretically ensuring an upper bound on overhead.

VEDRFOLNIR implements the aforementioned design on the host side through a detection agent, which is responsible for the dynamic updating of various parameters, the sending and receiving of notification packets, the judgment of triggering conditions, and other functions. The specific algorithmic flow is illustrated in Figure 8.

3) Telemetry Collection: On the network side, VEDRFOLNIR follows the same methodology as that of Hawkeye [16], [17] for telemetry collection. Specifically, switches periodically record flow-level telemetry (e.g., flows' 5-tuple, packet count per flow, queue depth, etc.) and port-level telemetry (e.g., traffic size between ports, number of packets paused by PFC per port, etc.) while monitoring port PFC states. Upon receiving a polling packet, the switch data plane propagates it along both the flow paths and the PFC spreading paths. Concurrently, the switch controller assists in data collection and reports the telemetry information to the analyzer.

D. Comprehensive Root Cause Analysis

Considering the complex flow patterns of collective communication, the root cause diagnosis of VEDRFOLNIR should be able to answer the following questions: 1) Where are the performance bottlenecks in collective communication? Which specific anomalous flows are slowing down the collective communication? 2) What is the underlying network root cause of the performance anomaly? Which flows or switches are involved? 3) If other flows are causing the decrease in collective communication performance, how significant is the impact of these flows on collective communication performance?

1) Graph Construction: VEDRFOLNIR constructs a waiting graph to analyze the performance bottlenecks of collective communication and identify critical flows. For each step of the collective communication, it constructs provenance graphs to locate network root causes and evaluate the impact of flows. **Waiting graph of collective communication.** The definition of the waiting graph is described in Section III-B. During runtime, the analyzer queues the collected data entries in order of their completion time and constructs the waiting graph sequentially according to the queue order. Based on the real-time collected data regarding waiting relationships and their corresponding timestamps, the analyzer constructs corresponding nodes and edges. For simplicity, upon determining that a

node is not being waited for (i.e., has an in-degree of zero), the analyzer can recursively prune nodes with an in-degree of zero. After constructing the complete waiting graph, the analyzer calculates the critical path, which represents the performance bottleneck of the collective communication.

Provenance graph of network. VEDRFOLNIR constructs the provenance graphs based mainly on packet-level waiting relationships between flows and ports. Previous works [16]–[18] are unable to evaluate the contribution between flows while performing PFC traceability. To support identifying the primary contributing flows, VEDRFOLNIR redefines the graph construction based on Hawkeye [16], [17]. The vertex set V consists of F (flows), P (ports), and CF (collective communication flows), where $CF \in F$ and $F \cup P = V$. The directed edge set E describes waiting relationships between vertex, which are categorized into the following three types:

(1) $e(f, p)$: Describes the waiting relationship from a flow to a port. Essentially, a flow waits on a port due to congestion caused by contention among multiple flows at that port. We use the queue depth encountered by the flow's packets while queued at the port as the edge weight. Suppose that a packet pkt of flow f_i enters the queue, and define the number of packets of flow f_j in front of it as $x_j(pkt)$. Then, the waiting weight of flow f_i for f_j is defined as $w(f_i, f_j) = \sum_{pkt \in f_i} x_j(pkt)$, and the weight of the flow for the port is $w(f_i, p) = \sum_{j \neq i} w(f_i, f_j)$.

(2) $e(p, f)$: Describes the waiting relationship from a port to a flow. We mainly use this type of edge to quantify the contribution of the flow to port congestion. Similarly, we use the number of packets contributed by the flow to the port queue depth as the weight of the edge. Assuming that the queue depth of port p is $qdepth(p)$ and the number of packets detected within a certain period is $pkt\_num(p)$, the weight of the port for the flow is calculated as $w(p, f_i) = \frac{pkt\_num(f_i)}{pkt\_num(p)} \times qdepth(p)$.

(3) $e(p_i, p_j)$: Describes the waiting relationship between ports. We use this type of edge to capture PFC causal relationships. When a PFC pause event occurs, an egress port p_j on a downstream switch may halt an egress port p_i on an upstream switch, establishing a waiting relationship from p_i to p_j. Since the measure of the wait is based on the queue depth, in this case, we consider p_i to be waiting for all packets in the p_j queue. However, PFC backpressure can propagate along multiple paths. In order to accurately evaluate the contribution ratio of the path from p_j to p_i, we define the weight from p_i to p_j as the traffic ratio $w(p_i, p_j) = \sum_k \frac{meter(p_i, p_j)}{meter(p_k, p_j)}$.

2) Anomaly Breakdown: Similar to Hawkeye [17] and SpiderMon [18], VEDRFOLNIR diagnoses specific root causes based on the signatures of different anomaly types. Similarly, VEDRFOLNIR can extend the range of detectable anomaly types by adding more signature definitions. Below, we provide examples of diagnosing two anomaly types: (1) *Flow contention.* VEDRFOLNIR identifies flow contention by checking the port vertices in the network provenance graph. When $\exists\, p \in P, \{f_i, cf\} \subseteq F \wedge \{e(f_i, p), e(cf, p)\} \subseteq E \wedge f_i \neq cf$, it is considered that f_i contend with the collective communication flow cf at port p. (2) *PFC backpressure.* VEDRFOLNIR identifies PFC backpressure by tracing the PFC spreading path. This occurs when $\exists\, p \in P, cf \in F, e(cf, p) \in E \wedge \exists\, p_j \in P, e(p, p_j) \in E$. Following the PFC spreading path. VEDRFOLNIR can pinpoint the root cause location of the PFC event. PFC storms typically also exhibits backpressure phenomena and is identified analogously.

3) Contributor Rating: In scenarios where collective communication contends with other flows, VEDRFOLNIR is capable of quantifying the contribution of these contending flows to identify the main contributors. In this way, VEDRFOLNIR can make recommendations on the priority of the actions of the network operations personnel.

Evaluating contribution to a collective communication flow. First, starting from all collective communication flows (CF), we obtain the largest connected subgraph of the network provenance graph, then all flows $f \notin CF$ belong to the evaluation object. For each flow, we calculate the contribution of the flow to a vertex by traversing along the reverse direction of directed edges (i.e., the direction of being waited for). Suppose that the neighboring vertices of flow f_i are p_1 and p_2, where p_2 is downstream of p_1. Then the impact score of f_i is $R(f_i, p_2) = w(p_2, f_i)$, $R(f_i, p_1) = w(p_1, f_i) + R(f_i, p_2) \times w(p_1, p_2)$. Generally, the contribution of a flow to a port is:

$$R(f_i, p_j) = w(p_j, f_i) + R(f_i, p_k) \times w(p_j, p_k), \quad e(p_j, p_k) \in E \tag{1}$$

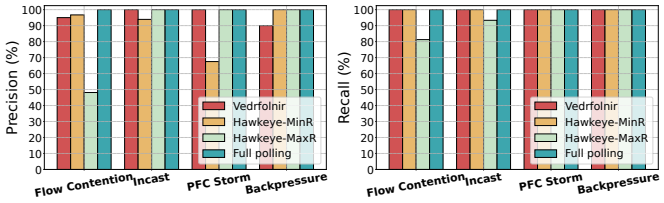
Given a collective communication flow cf, its neighboring vertices set P_{cf}, and $R(f_i, p_k)$ for any $p_k \in P_{cf}$. Since cf and f_i may directly cause flow contention at port p_k, in this case, the impact of f_i on cf is $w(cf, f_i)$ rather than $w(p_k, f_i)$. Therefore, the contribution of flow f_i to cf is calculated as:

$$R(f_i, cf) = \sum_{p_k \in P_{cf}} [(w(cf, f_i) - w(p_k, f_i)) \times \mathbb{I}(e(f_i, p_k) \in E) + R(f_i, p_k)] \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function that determines whether f_i and cf contend at p_k; $w(cf, f_i)$ can be derived via a replay algorithm [17].

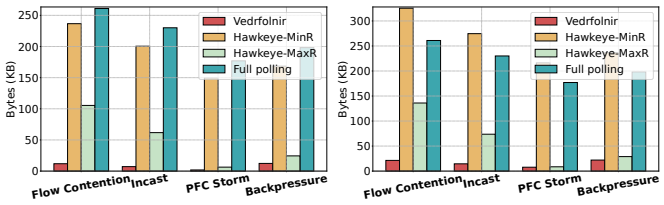
Evaluating contribution to collective communication. Since the performance of collective communication is mainly determined by the flow on the critical path in the waiting graph, we obtain the impact score of flow f_a to the overall collective communication by weighting the sum of the critical flow scores $R(f_a, cf_i)$ at step i. Let the execution time of step i be $exec\_time(i)$ and the expected execution time be $expect\_time(i)$, the contribution of flow f_a to the overall collective communication is:

$$R(f_a) = \sum_i R(f_a, cf_i) \times \frac{exec\_time(i) - expect\_time(i)}{\sum_k (exec\_time(k) - expect\_time(k))} \tag{3}$$



(a) Precision v.s. baselines. (b) Recall v.s. baselines.

Fig. 9. Precision & recall v.s. baselines.



(a) Processing overhead. (b) Bandwidth overhead.

Fig. 10. Overhead v.s. baselines.

where $expect\_time(i)$ can be derived theoretically or empirically. By scoring each flow, VEDRFOLNIR finds the main contributors to flow contention.

IV. EVALUATION

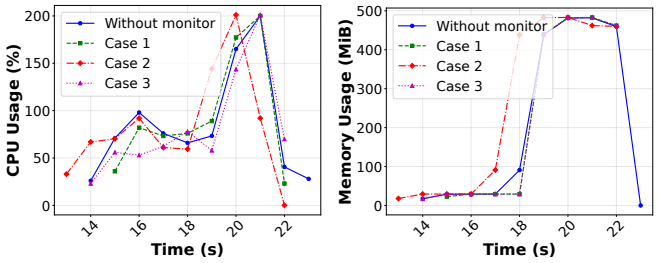
The prototype of VEDRFOLNIR in NS-3 simulator is implemented with about 1000 lines of additional C++ code and the source code is publicly available at Github [24]. The VEDRFOLNIR prototype in real testbed is implemented with about 500 lines of additional C++/Python codes based on the system of our partner company. Through both large-scale simulation and real testbed experiments, we evaluate VEDRFOLNIR to answer the following questions: (1) Can VEDRFOLNIR diagnose RDMA NPAs in collective communications accurately and efficiently (§IV-B)? (2) How effective is design techniques in VEDRFOLNIR (§IV-C)? (3) How does VEDRFOLNIR performance under real-world cases (§IV-D)?

A. Experimental Setup

Topology. We set up a standard Fat-Tree (K=4) topology with 20 switches on NS3. The link bandwidth is 100 Gbps and the link delay is 2 us. Our real testbed experiments use four H100s connected via RoCE NICs, each machine having a total of 192 CPU cores and 50 GB of memory.

Workload. We simulate an empirical collective communication workload derived from an analysis of LLM training [34]. Specifically, 97% of collective communication operations are AllReduce or AllGather, each with a data size of 360 MB per traffic. For simplicity, we adopt 360 MB per flow in a similar way and use the Ring algorithm for AllGather as the workload.

Anomaly construction. We construct four anomaly scenarios for comprehensive testing. `1) Flow contention` are constructed by injecting flows into the existing workload. We construct 60 cases, with the number of flows uniformly distributed in 1-6, sizes randomly chosen in 20 MB-1 GB, and start times randomly distributed in 0-200 ms. These flows are placed randomly but deliberately set to collide with collective communication flows. For diagnosis, we define detecting all injected flows as a true positive, detecting only some flows as a false positive, and failing to detect any anomaly as a false negative. `2) Incast` includes 60 cases in a similar way to flow contention. Specifically, the number of flows changes to 3-8, with random positions but multiple flows targeting the same node. Flow sizes are randomly selected in 20-200M, and all flows start simultaneously. The diagnostic criteria are the same as for flow contention. `3) PFC storm` is constructed by injecting continuous PFC frames. In 40 cases, the injection point is selected from the switch ports along the paths of 4 collective communication flows. The start time is random in 0-150 ms, and the duration is random in 10-100 ms. For diagnosis, tracing to the source port where the PFC occurred is defined as a true positive, merely reporting the presence of PFC is defined as a false positive, and failing to detect any anomaly is defined as a false negative. `4) PFC Backpressure` includes 60 cases. It is similar to PFC storms. The unique feature is that the PFC does not originate on the collective communication path but affects collective communication through multi-hop propagation. We generat PFC using incast traffic and design propagation paths partially overlapping collective communication flows. The diagnostic criteria are the same as for PFC storms.



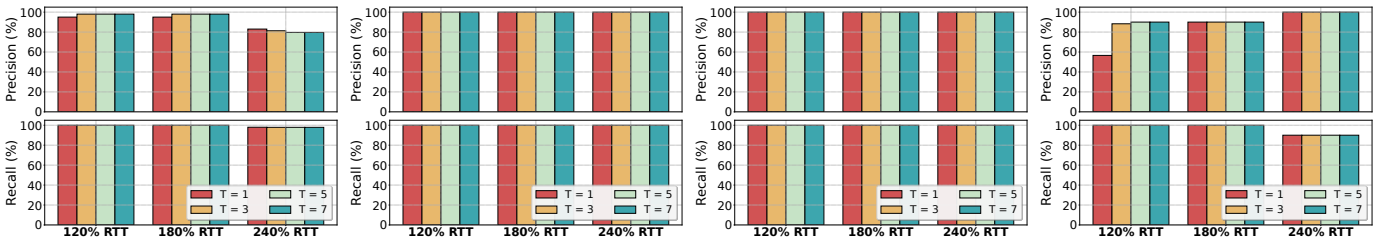
(a) CPU overhead. (b) Memory overhead.

Fig. 11. Overhead in real testbed.

Baseline systems. We compare VEDRFOLNIR with two baseline systems. 1) Hawkeye [17]: A state-of-the-art RDMA NPA diagnosis system that collects telemetry data from PFC-related paths and diagnoses root causes. When anomaly detection, Hawkeye sets a fixed RTT threshold for all flows, whereas the RTTs of different flows in collective communication vary. For accurate evaluation, we use Hawkeye-MaxR to denote setting Hawkeye's RTT threshold to 120% of the maximum RTT, and Hawkeye-MinR to denote setting the RTT threshold to 120% of the minimum RTT. 2) Full polling: Continuously collects telemetry data from all switches in the network.

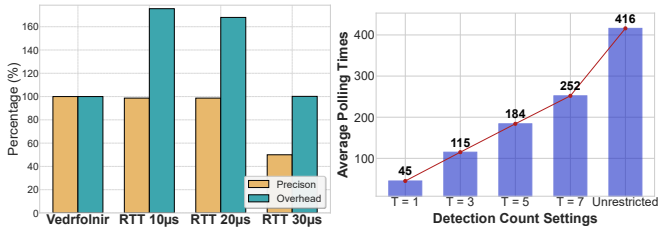
B. Diagnosis Effectiveness

Precision and recall. Our initial analysis focuses on the precision and recall of different solutions. For each scenario, we demonstrate the precision and recall with optimal parameters. The experimental results are presented in Figure 9a and Figure 9b. VEDRFOLNIR demonstrates excellent precision and recall across all scenarios. In flow contention, Hawkeye-MaxR tends to easily overlook flows with small RTTs, resulting in poor performance. Additionally, due to its detection-triggering



(a) Flow contention. (b) Incast. (c) PFC storm. (d) PFC backpressure.

Fig. 12. Precision & recall over different RTT thresholds and detection counts.



(a) Precision and overhead. (b) Number of polling triggers.

Fig. 13. Ablation studies of our methods.

design, Hawkeye is prone to collecting several pieces of telemetry data within tens of microseconds, though only one of them may actually be needed. In the source code of Hawkeye, to avoid excessive processing overhead, only one piece of data is retained every 50 microseconds, which can inadvertently lead to valid data being discarded. As a result, it can be observed that Hawkeye-MinR collects more redundant data and performs even worse. VEDRFOLNIR does not have this issue due to its restricted detection design.

Processing overhead. We utilize the size of telemetry packets collected for diagnostic purposes to evaluate the processing overhead of different approaches. Figure 10a shows the average processing overhead for all samples in each scenario. VEDRFOLNIR demonstrates the lowest processing overhead across all tested scenarios, with telemetry data volume consistently maintained around 10KB, achieving 60%–98% savings compared to Hawkeye. Due to the low RTT threshold, Hawkeye-MinR continuously triggers detection for flows with long RTTs, resulting in high overhead. For the reason illustrated in Figure 5, Hawkeye-MaxR also performs poorly. However, Hawkeye-MaxR shows better performance in PFC anomalies because Hawkeye measures RTT through ACKs for each packet. When persistent PFC halts an entire flow, no packets are sent, and thus no detection is triggered, leading to lower overhead. In contrast, Full polling demonstrates the upper bound of overhead by continuously triggering detection during collective communication.

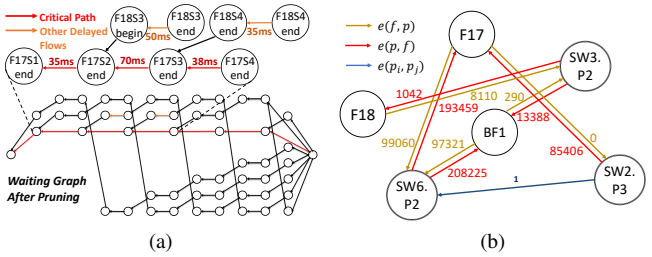
Bandwidth overhead. We measure the additional bandwidth overhead introduced by different methods during monitoring. The bandwidth overhead here includes polling during detection, notification packets, and switch telemetry reports. As illustrated in Figure 10b, VEDRFOLNIR still achieves the lowest bandwidth overhead across all scenarios. Compared to processing overhead, the bandwidth overhead of VEDRFOLNIR and Hawkeye is higher. However, Hawkeye's overhead increases more noticeably due to repeated anomaly detection. Full polling simulates the scenario where switches continuously and autonomously report telemetry data, and thus excluding detection overhead.

Real testbed evaluation. Compared to Hawkeye, VEDRFOLNIR introduces an additional host-side monitor, which incurs extra CPU overhead during collective communications. We conduct real testbed experiments using NCCL, executing a 4-node AllGather operation with 1 GB transmission volume. Performance collection, notification packet transmission, and data reporting are triggered during the corresponding phases. Figure 11 illustrates the computational and memory overhead. When comparing three test cases with scenarios without the monitor, the additional CPU and memory overhead is practically negligible.

C. Ablation Study

Step-aware mechanism. We validate the effectiveness of the step-aware mechanism from two aspects: 1) Step-grained RTT thresholds settings. In collective communication, RTT varies not only across different flows but may also change across different steps. Figure 13a presents the precision and overhead of VEDRFOLNIR under different fixed RTT thresholds. The experiment is conducted in the flow contention scenario, limiting the number of detections per step to a maximum of 3. 2) Detection triggering count allocation. We evaluate different settings for the number of detections to observe the polling frequency. Figure 13b shows the average results under different configurations in the flow contention scenario. Compared to unrestricted triggering similar to Hawkeye, VEDRFOLNIR achieve significant gains in overhead reduction.

RTT threshold and detection counts. We evaluate two important detection parameters in VEDRFOLNIR: the RTT threshold and the number of detections per step. Figure 12 shows the precision and recall of VEDRFOLNIR for each scenario under different parameter settings. Theoretically, a larger RTT threshold results in slower detection responses after an anomaly occurs. This is evident in flow contention and PFC backpressure, where performance is poorer at 240% RTT. Increasing the detection count improves accuracy, as clearly demonstrated in the PFC backpressure experiments at 120% RTT. Note that precision is relatively poor when the detection count is set to 1. This occurs because, unlike PFC storms, PFC backpressure is not continuous but triggered intermittently, which may cause VEDRFOLNIR 's detection to occur precisely during the recovery intervals of PFC. As a result, only some



(a) (b)

Fig. 14. Case study. (a) Waiting graph. (b) Network provenance example.

switches along the PFC backpressure path are traced, and the root cause is not identified. This issue can be resolved by increasing the number of detections.

D. Case Study

In the topology shown in Figure 2a, a collective communication running the Ring algorithm is executed with each step involving 360M flow size, and the two interference flows (BF1 and BF2) depicted in the figure are injected. BF1 is approximately 90M in size, and BF2 is approximately 450M in size. Figure 14a shows the diagnostic results of the waiting graph, where nodes with an in-degree of 0 (i.e., nodes that are not waited for by any other) are removed. The graph visually reveals the dependency relationships among flows during execution, facilitating the analysis of latency and critical paths. In this example, F17 is identified as the key factor slowing down the entire collective communication. Figure 14b is one of the provenance graphs, providing further analysis of F17. It can be observed that F17 collides with BF1 at Switch 6 Port 2, which triggers PFC, causing Switch 2 Port 3 to be halted and further impacting F17. Our rating design assigns higher scores to flows that have a greater impact. In Figure 14b, BF2 scores 248,489 for F17 and 8,110 for F18. For the collective communication in this example, BF1 scores 698 and BF2 scores 104,095. Details are available on GitHub [24].

V. Discussion

Non-network root cause diagnosis. Beyond network performance anomalies, collective communication anomalies may stem from other sources such as slow computation. VEDRFOLNIR provides a novel perspective for diagnosing network performance anomalies and can seamlessly integrate with host-based diagnostic systems to achieve comprehensive root cause diagnosis. We leave a full integration of these two methdologies as our future work.

Extensibility to anomaly types. Our current evaluations cover four anomaly types, yet VEDRFOLNIR enables straightforward incorporation of additional types based on their distinctive signatures. Although VEDRFOLNIR's adaptive mechanism may be ineffective for anomalies such as PFC deadlocks (which stall multiple collective flows), its flexibility allows simple fixes, e.g., immediately triggering an investigation when a flow remains stalled for an extended period.

Extensibility of algorithm decomposition. VEDRFOLNIR primarily characterizes execution flows through an algorithm's wait-dependency relationships. Given that synchronization is fundamental to most collective communications, VEDRFOLNIR applies broadly across nearly all collective algorithms.

Applicability in large-scale scenarios. The computational and storage complexity of the waiting graph is $O(N_n S)$, where N_n denotes the number of nodes in the set communication and S represents the number of steps. The computational and storage complexity of the provenance graph is $O(N_s T)$, where N_s denotes the number of relevant switches and T represents the number of switch reports. In large-scale scenarios, the centralized analyzer may become a bottleneck, requiring further design to address storage demands.

VI. Related Work

RDMA performance anomalies. Hostping [35] detects performance anomalies in different PCIe and NVLink pathways within the machine by performing internal loopback tests. R-Pingmesh [36], through full mesh connectivity testing, diagnoses performance anomalies between machines. However, most performance anomalies are derived from internal machine issues or misconfigurations, and they do not address the impact of complex traffic patterns in the network.

Model training performance anomalies. Holmes [37] and Aegis [38] detect anomalous nodes in the training cluster by analyzing the patterns of collective communication calls. The goal of this detection is to identify problematic nodes or links, considering both computational and communication performance. However, there is no in-depth analysis of the root causes of network performance anomalies.

Root cause diagnosis. Both Hawkeye [17] and SpiderMon [18] can perform root cause localization based on different anomaly types. However, when evaluating contributions, SpiderMon treats all detected flows equally and cannot evaluate the contribution of individual flows. Hawkeye traces root causes via PFC causality but fails to quantify the impact of contributions under collective communications.

VII. Conclusion

The co-flow pattern of collective communications introduces new complexities for RDMA NPA diagnosis. We present VEDRFOLNIR, an accurate and efficient anomaly diagnosis system for RDMA NPAs in collective communications. It employs algorithm decomposition to characterize collective communication processes, step-aware adaptive detection to minimize redundant overhead, and comprehensive diagnosis to identify the root causes. Evaluation demonstrates that VEDRFOLNIR accurately generates diagnostic results with low overhead. The authors have provided public access to their code and data at https://github.com/Networked-System-and-Security-Group/Vedrfolnir.

Acknowledgments

We thank the anonymous INFOCOM reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (No. 62402025), the Open Research Fund of State Key Laboratory of Internet Architecture (HLW2025ZD01) and the Huawei-BUAA Joint Lab. Menghao Zhang is the corresponding author.

REFERENCES

[1] InfiniBand Trade Association, "InfiniBand Architecture Specification Release 1.4 Annex A17: RoCEv2," 2020.

[2] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan, F. Feng, Y. Zhuang, F. Liu, P. Liu, X. Liu, Z. Wu, J. Wu, Z. Cao, C. Tian, J. Wu, J. Zhu, H. Wang, D. Cai, and J. Wu, "When cloud storage meets RDMA," in *NSDI 2021*, 2021, pp. 519–533.

[3] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye *et al.*, "Rdma over commodity ethernet at scale," in *ACM SIGCOMM 2016*, 2016, p. 202–215.

[4] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl *et al.*, "Empowering azure storage with RDMA," in *NSDI 23*, pp. 49–67.

[5] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. K. Martin, M. McLaren, P. Chandra, R. Cauble, H. M. G. Wassel, B. Montazeri, S. L. Sabato, J. Scherpelz, and A. Vahdat, "1rma: Re-envisioning remote memory access for multi-tenant datacenters," in *SIGCOMM 2020*, 2020, p. 708–721.

[6] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang *et al.*, "Rdma over ethernet for distributed training at meta scale," in *ACM SIGCOMM 2024*, 2024, pp. 57–70.

[7] J. Cao, Y. Guan, K. Qian, J. Gao, W. Xiao, J. Dong, B. Fu, D. Cai, and E. Zhai, "Crux: Gpu-efficient communication scheduling for deep learning training," in *ACM SIGCOMM 2024*, 2024, p. 1–15.

[8] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *OSDI 20*, 2020, pp. 463–479.

[9] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu, "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *NSDI 2024*, 2024, pp. 745–760.

[10] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. Kewitsch, "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," in *NSDI 23*, 2023, pp. 739–767.

[11] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang, and L. Zhou, "Fast distributed deep learning over rdma," in *EuroSys 2019*, 2019.

[12] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba hpn: A data center network for large language model training," in *ACM SIGCOMM 2024*, 2024, p. 691–706.

[13] IEEE, "IEEE 802.1 Qbb - Priority-based Flow Control," https://1.ieee802.org/dcb/802-1qbb/, 2011.

[14] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *ACM SIGCOMM 2015*, 2015.

[15] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *ACM SIGCOMM 2020*, 2020, pp. 514–528.

[16] X. Li, S. Wang, M. Zhang, Z. Wang, M. Xu, and J. Yang, "Poster: Rdma network performance anomalies diagnosis with hawkeye," in *ACM SIGCOMM 2024*, 2024.

[17] S. Wang, M. Zhang, X. Li, Q. Peng, H. Yu, Z. Wang, X. Hu, J. Yang, and X. Shi, "Hawkeye: Diagnosing rdma network performance anomalies with pfc provenance," in *ACM SIGCOMM 2025*, 2025.

[18] W. Wang, X. C. Wu, P. Tammana, A. Chen, and T. S. E. Ng, "Closed-loop network performance monitoring and diagnosis with SpiderMon," in *USENIX NSDI 2022*, Apr. 2022.

[19] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: query-driven streaming network telemetry," in *ACM SIGCOMM 2018*, 2018.

[20] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow," in *USENIX ATC 2018*, 2018.

[21] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, p. 49–66, Feb. 2005.

[22] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: using packet histories to troubleshoot networks," in *USENIX NSDI 2014*, 2014.

[23] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *ACM SIGCOMM 2020*, 2020.

[24] Vedrfolnir, "Vedrfolnir: Rdma network performance anomalies diagnosis in collective communication," https://github.com/Networked-System-and-Security-Group/Vedrfolnir, 2025.

[25] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: a better netflow for data centers," in *USENIX NSDI 2016*, 2016.

[26] P. Tammana, R. Agarwal, and M. Lee, "Distributed network monitoring and debugging with switchpointer," in *USENIX NSDI 2018*, 2018.

[27] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data plane performance diagnosis of tcp," in *ACM SOSR 2017*, 2017.

[28] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM 2015*, 2015.

[29] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *ACM SIGCOMM 2016*, 2016.

[30] Y. Lei, L. Yu, V. Liu, and M. Xu, "Printqueue: performance diagnosis via queue measurement in the data plane," in *ACM SIGCOMM 2022*, 2022.

[31] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. Liu, J. Padhye, B. T. Loo, and G. Outhred, "007: democratically finding the cause of packet drops," in *USENIX NSDI 2018*, 2018.

[32] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang, "Fine-grained queue measurement in the data plane," in *ACM CoNEXT 2019*, 2019.

[33] Y. Wu, A. Chen, and L. T. X. Phan, "Zeno: diagnosing performance prob- lems with temporal provenance," in *USENIX NSDI 2024*, 2024.

[34] H. Liao, B. Liu, X. Chen, Z. Guo, C. Cheng, J. Wang, X. Chen, P. Dong, R. Meng, W. Liu *et al.*, "Ub-mesh: a hierarchically localized nd-fullmesh datacenter network architecture," *arXiv preprint arXiv:2503.20377*, 2025.

[35] K. Liu, Z. Jiang, J. Zhang, H. Wei, X. Zhong, L. Tan, T. Pan, and T. Huang, "Hostping: Diagnosing intra-host network bottlenecks in {RDMA} servers," in *USENIX NSDI 2023*, 2023, pp. 15–29.

[36] K. Liu, Z. Jiang, J. Zhang, S. Guo, X. Zhang, Y. Bai, Y. Dong, F. Luo, Z. Zhang, L. Wang, X. Shi, H. Xu, Y. Bai, D. Song, H. Wei, B. Li, Y. Pan, T. Pan, and T. Huang, "R-pingmesh: A service-aware roce network monitoring and diagnostic system," in *ACM SIGCOMM 2024*, 2024, p. 554–567.

[37] Z. Yao, M. C. Hu, Pengbo and, X. Jia *et al.*, "Holmes: Localizing irregularities in llm training with mega-scale gpu clusters," in *USENIX NSDI 2025*, 2025.

[38] J. Dong, K. Qian, P. Zhang *et al.*, "Evolution of aegis: Fault diagnosis for ai model training service in production," in *USENIX NSDI 2025*, 2025.