R-Pingmesh: A Service-Aware RoCE Network Monitoring and Diagnostic System

Kefei Liu[1], Zhuo Jiang[3], Jiao Zhang[1,2,*], Shixian Guo[3], Xuan Zhang[1], Yangyang Bai[3], Yongbin Dong[3]
Feng Luo[3], Zhang Zhang[3], Lei Wang[3], Xiang Shi[3], Haohan Xu[3], Yang Bai[3], Dongyang Song[3]
Haoran Wei[3], Bo Li[3], Yongchen Pan[1], Tian Pan[1,2], Tao Huang[1,2]
[1]State Key Laboratory of Networking and Switching Technology, BUPT, China
[2]Purple Mountain Laboratories [3]Douyin Vision Co., Ltd.

ABSTRACT

RoCE services are sensitive to network failures and performance bottlenecks, which become more common as the RoCE network scales. In addition, some non-network problems behave like network problems and can waste troubleshooting time. However, existing mechanisms cannot quickly detect and locate network problems or determine whether the service problem is network-related.

In this paper, we propose R-Pingmesh, the first service-aware RoCE network monitoring and diagnostic system based on end-to-end probing. R-Pingmesh can accurately measure network RTT and end-host processing delay based on commodity RDMA NICs (RNICs), distinguish between RNIC and in-network packet drops, and judge whether a problem is network-related and assess its impact on services. We have deployed R-Pingmesh on tens of thousands of RNICs for over 6 months. One-month evaluation results show that 85% of the problems located by R-Pingmesh are accurate, where all 157 switch network problems are accurate. R-Pingmesh efficiently detects and locates 14 types of problems during deployment, and we share our experience in dealing with them.

CCS CONCEPTS

• **Networks** → **Data center networks**; **Network measurement**; **Network monitoring**; **Error detection and error correction**.

KEYWORDS

RDMA; Network troubleshooting; Service tracing probing

ACM Reference Format:
Kefei Liu[1], Zhuo Jiang[3], Jiao Zhang[1,2,*], Shixian Guo[3], Xuan Zhang[1], Yangyang Bai[3], Yongbin Dong[3], Feng Luo[3], Zhang Zhang[3], Lei Wang[3], Xiang Shi[3], Haohan Xu[3], Yang Bai[3], Dongyang Song[3], Haoran Wei[3], Bo Li[3], Yongchen Pan[1], Tian Pan[1,2], Tao Huang[1,2]. 2024. R-Pingmesh: A Service-Aware RoCE Network Monitoring and Diagnostic System. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3651890.3672264

The first two authors contributed equally to this paper. This work is done while Kefei Liu, Xuan Zhang, and Yongchen Pan are doing a joint research project at Douyin Vision Co., Ltd. (**Jiao Zhang is the corresponding author.*)

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672264

1 INTRODUCTION

Remote Direct Memory Access (RDMA) has been used by many data center services [2, 10, 19, 20, 40, 47] to achieve low latency and high throughput with low CPU overhead. Among all RDMA networking solutions in the data center, RDMA over Converged Ethernet (RoCE) is the most widely adopted.

Timely and accurately locating network problems is critical to ensuring stable and high-quality operation of RoCE services. The performance of some services, such as Distributed Machine Learning (DML), has a *barrel effect*. First, they are extremely sensitive to *single-point failures*. For example, packet drops caused by a single flapping switch port or RNIC[1] can severely degrade the training throughput of the entire cluster. Some failures, such as an accident RNIC or host down, can even cause the training task to fail. Second, *performance bottlenecks* that occur in both end hosts and networks can also severely degrade service performance, such as CPU overload and network congestion due to uneven load balancing. Over the past year, the size of the RoCE network used by services (service network) has grown rapidly from hundreds of RNICs to tens of thousands of RNICs, and this growth continues. As a result, both the frequency and impact of single-point problems are increasing.

In addition, when service problems occur, the network team should quickly determine if the problem is network-related. If so, resolve it. If not, the service team should work with other teams to find the problem. This is not easy. *First, when a service fails, service logs can be misleading.* Some of them, such as NCCL [36] "error code 12", look like a network problem. However, the root cause may be an *abnormal remote process exit* due to GPU down or GPU driver failure. Second, service performance degradation can also be caused by many non-network problems, such as degraded compute speed due to GPU underclocking or improper NCCL parameters. *These problems can degrade the average network throughput and also appear as network problems.*

However, existing mechanisms cannot quickly detect and locate network problems. In addition, when service suffers from non-network problems, operators often *waste a lot of time trying to find the root cause in the network.* Previously, the method of diagnosing service performance problems could be roughly divided into two steps: (1) inspecting the servers and switches in the service network for configurations, logs, and anomalous metrics (e.g., packet drops, Tx/Rx PFC pause frames); (2) running benchmark tools such as perftest [4] and nccl-test [3] between the RNICs in the service network. This approach works when the cluster size is small. As the RoCE network scales, these troubleshooting steps become time-consuming.

[1]The state of a flapping RNIC or port frequently fluctuates between up and down.

In this paper, we extend Pingmesh [13] to RoCE networks and propose R-Pingmesh, the first service-aware RoCE network monitoring and diagnostic system based on end-to-end active probing, to address the above problems. To fully meet our requirements, we need to achieve four design targets that cannot be achieved by Pingmesh. First, we need to detect problems specific to RoCE networks, such as PFC misconfigurations. Second, we need to distinguish between RNIC and in-network packet drops to accurately locate network failures. Third, we need to accurately measure end-host processing delay and service network RTT to detect and locate performance bottlenecks. Finally, we need to assess the impact of problems on services so that we can determine a solution for each problem to minimize service performance loss and determine if a service problem is caused by networks.

First, by constructing regular RoCE packets as probes and ACKs, R-Pingmesh can detect RoCE-specific problems. Second, with ToR-mesh (Top-of-Rack switch) probing, R-Pingmesh can detect anomalous RNICs and identify anomalous probes caused by RNICs in real time. Third, by probing with UD QP, R-Pingmesh reduces the consumption of RNIC connectivity resources and can accurately measure network RTT and end-host processing delay. And by performing the service tracing probing, R-Pingmesh can measure the service network RTT. Finally, by monitoring service performance degradation and judging whether a problem lies in the service network, R-Pingmesh can assess the impact of problems on services.

We have deployed R-Pingmesh for over 6 months on multiple DML RoCE clusters totaling tens of thousands of RNICs. These clusters include both on-premises and public cloud clusters. During deployment, R-Pingmesh can detect, categorize, and locate problems in 20s and determine their impact on services. We evaluate the performance of R-Pingmesh on a large-scale DML RoCE cluster with thousands of GPU servers. Data collected from a recent month in the DML cluster shows that 85% of problems detected and located by R-Pingmesh are accurate, where all 157 switch network problems are accurate. R-Pingmesh can efficiently detect and locate 14 types of problems caused by hardware failures, misconfigurations, network congestion, and intra-host bottlenecks during deployment. We share our experience in dealing with these problems. For example, for RNIC or switch port flapping, we increase the retransmission timeout and set the number of retransmissions to the maximum to prevent service failures. In summary, this paper makes the following contributions:

- We present the first RoCE network monitoring and diagnostic system based on end-to-end active probing. It can be easily deployed in data center RoCE clusters with low overhead. Instead of passive flow tracing, R-Pingmesh achieves service awareness by actively probing the service network. This approach is easy to deploy and has low overhead.
- R-Pingmesh innovatively proposes (1) a method to accurately measure network RTT and end-host processing delay with low overhead on commodity RNICs; (2) ToR-mesh probing to detect abnormal RNICs and distinguish anomalous probes caused by these RNICs in real time; and (3) an easy-to-deploy method to obtain the service flow 5-tuples in real time with low overhead.
- We summarize the problems detected and located by R-Pingmesh and share our experience in dealing with them.

2 BACKGROUND & MOTIVATION

2.1 Is It a Network Problem?

Service logs can be misleading. Some of our DML services rely on NCCL to communicate between GPUs. If the service fails, the service team will inspect the NCCL log to determine the root cause. One of the most common errors is "error code 12", which means that RDMA communication between the local and remote process experiences timeout. This sounds like a network problem, and it is true that many "error code 12" problems are caused by the network, such as RNIC down or persistent packet drops on switches. However, the cause could also be an *abnormal remote process exit* due to GPU down, GPU hang, and GPU out of memory.

In addition, service performance degradation can also be caused by non-network problems, such as degraded compute performance due to GPU underclocking. As DML services constantly repeat computation and communication, and each cycle takes only a few seconds, *these problems appear as degraded network throughput at coarse monitoring granularity and also look like network problems.* When these non-network problems occur and cause service failure or performance degradation, network operators often waste a lot of time trying to find the root cause in the network.

We lack an efficient end-to-end RoCE network measurement tool. Previously, methods for diagnosing service performance problems can be roughly divided into two steps: (1) inspecting the servers and switches in the service network for configurations, logs, and anomalous metrics (e.g., packet drops, Tx/Rx PFC pause frames); (2) running benchmark tools such as perftest [38] and nccl-test [37] between the RNICs in the service network. This approach works when the cluster size is small (e.g., tens to hundreds of RNICs). However, over the past year, the size of the service network has grown rapidly to tens of thousands of RNICs, and this growth continues. In this scenario, the above troubleshooting steps are time-consuming. In addition, *it is not sufficient to measure the network using only the RNICs used by the service*, because the service may randomly use a part of the network links in a cluster due to ECMP. Therefore, this tool should be able to *measure the performance of the network used by the service (i.e., the service network).*

2.2 Frequent and Harmful Packet Drops

In theory, lossless RoCE networks do not drop packets due to network congestion. However, we find that packet drops due to network failures are common. These network failures include (1) *anomalous RNICs*, such as flapping RNICs, and (2) *anomalous in-network devices*, such as flapping switch ports, damaged or aged fiber, and misconfigured switches.

In DML, all participating GPUs must periodically synchronize their local gradients over the network [5, 19, 44]. The completion time of this process is determined by the slowest GPU (*barrel effect*). Since RDMA throughput is vulnerable to packet drops, either RNIC or in-network packet drops can cause severe throughput degradation of multiple flows, which can further degrade the average throughput of the DML cluster due to the barrel effect. As shown in Figure 1, a single flapping RNIC or switch port severely degrades the average training throughput of the DML cluster. However, *we lack an effective monitoring and diagnostic system to quickly detect packet drops and accurately locate the abnormal device.*

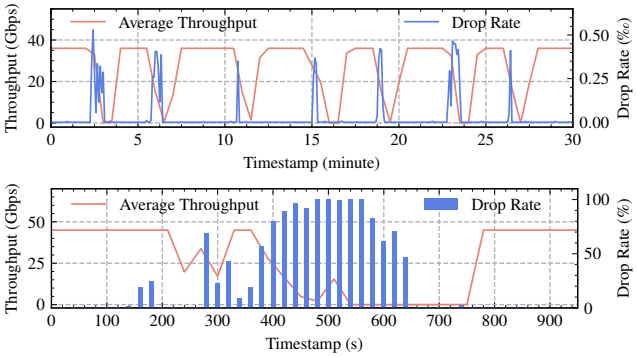

Figure 1: A single flapping (up) switch port or (down) RNIC severely degrades the average training throughput of the entire DML cluster, even to zero.

2.3 Hard to Detect and Locate Bottlenecks

In addition to network failures, *performance bottlenecks* can also degrade service performance. These bottlenecks can occur in *end hosts*, such as CPU overload, and in *inter-host networks*, such as switch port congestion caused by uneven load balancing. And as cluster size grows, both the frequency and impact of performance bottlenecks increase. Here are two bottlenecks that occur in our DML RoCE clusters.

Case 1: Network congestion due to uneven load balancing. Our RoCE clusters use ECMP to hash training flows to parallel paths. As the number of hosts involved in training increases, the number of RDMA connections between hosts increases dramatically, increasing the probability of hash collisions. When this happens, all flows on the bottleneck link suffer throughput degradation, further degrading the throughput of the entire cluster.

Case 2: CPU overload. Before training begins, our hosts in the training task need to load training models from the remote storage cluster. During this process, all participating hosts must complete the loading process before training can begin. The load process in our DML cluster is TCP-based, which is CPU intensive. A host with an overloaded CPU can slow down the loading process for all hosts, thereby degrading the training rate.

These bottlenecks can be detected by latency metrics such as the RTT of paths used by the service and the end-host processing delay. However, *we lack an effective system to monitor these metrics.* Furthermore, since the RoCE network RTT is typically in tens of microseconds, we need to measure the network RTT accurately. As shown in Figure 2, it is not enough to measure network latency at the application layer, which can be affected by CPU load.

2.4 Targets & Limitations of Pingmesh

We need an efficient RoCE network monitoring and diagnostic system to quickly detect and locate RoCE network failures and bottlenecks. It can also quickly determine if the network is to blame when service performance degrades. We are inspired by our TCP networks. Pingmesh [13] is a data center physical and TCP network monitoring and diagnostic system based on end-to-end RTT probing. Pingmesh Agent has been deployed on most of the servers in our data centers. *Can we extend Pingmesh to RoCE networks to meet our needs?* This allows us to build on existing systems with minimal development and deployment effort. However, *although RoCE NICs also support TCP communication, deploying Pingmesh directly in RoCE networks cannot meet our requirements.* In this section, we present our targets and show the limitations of Pingmesh.

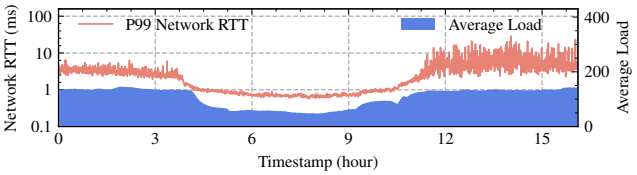

Figure 2: P99 TCP RTT measured by Pingmesh in one of our clusters. The measured software RTT fluctuates as the host average load changes.

Detect problems specific to RoCE networks. TCP probes in Pingmesh cannot detect all RoCE network issues. RoCE networks typically require lossless Ethernet, while traditional TCP networks are lossy. To ensure that RoCE and TCP traffic do not interfere with each other, data centers typically assign TCP and RoCE traffic to different traffic queues in both RNICs and switches, and enable PFC in the RoCE queue [12]. As a result, TCP probes cannot detect problems specific to RoCE queues, such as RoCE network congestion and packet drops caused by improper PFC configurations.

Distinguish between RNIC and in-network drops. By collecting 5-tuples of dropped Pingmesh probes and tracing their paths, operators can further locate anomalous network devices. However, Pingmesh cannot tell if a dropped probe is caused by a NIC or a switch, which limits the accuracy of using its path for troubleshooting. For example, a failed switch link drops some probes. At the same time, several anomalous NICs also drop some probes. In this case, the NIC and switch drops can interfere with each other during troubleshooting, leading to inaccurate localization of packet drops. As a result, *it can take several to tens of minutes for operators to further confirm the true location of the packet drops.* Therefore, we need to distinguish whether the RNIC or the switch is dropping packets in order to accurately locate RNIC and switch problems.

Accurately measure end-host processing delay and service network RTT. To detect and locate performance bottlenecks, we need to accurately measure the end-host processing delay and the RTT of the paths used by the service. However, Pingmesh measures RTT at the software layer, and the measured RTT has three components: network RTT, prober processing delay, and responder processing delay. As shown in Figure 2, although Pingmesh can detect an abnormal increase in RTT when performance bottlenecks occur, *it is difficult to locate them because the increase can come from either end hosts or networks.* In addition, as Pingmesh is service-oblivious, it cannot measure service network RTT.

Assess the impact of problems on services. Not all problems detected should be resolved immediately. Some problems do not severely affect service, such as packet drops caused by transient flapping. Whether to fix these problems should be based on benefit, because fixing them may also cause network fluctuations (e.g., isolating the anomalous switch port) or even require a service restart (e.g., replacing the anomalous host). Therefore, we need to prioritize each detected problem based on its impact and determine a solution to minimize the loss of service performance:

- *P0 (Priority 0)*: This problem has a severe impact on service performance and must be resolved immediately.
- *P1*: The impact of this problem on the service is not severe. Whether to fix it should be based on benefit.

- *P2*: This problem is not in the service network. However, the anomalous device should be isolated or repaired to prevent service performance degradation.

The priority also helps determine if a service problem is caused by the network. *If no P0 or P1 problems are detected when service performance degrades, we can confirm that the network is innocent.* However, Pingmesh is service-oblivious, it cannot assess the impact of problems on services.

3 R-PINGMESH OVERVIEW

3.1 Challenges and Opportunities

To achieve these targets, we need to solve three challenges.

Use commodity RNICs to accurately measure network latency and end-host processing delay. To ensure deployability, the measurement should use the standard interfaces of commodity RNICs without relying on special firmware or drivers. However, commodity RNICs do not provide timestamps when they send/receive packets. They only provide timestamps when generating Completion Queue Events (CQE). The challenge is to *use these implicit timestamps to determine the RNIC's send and receive times*, and measure network latency and end-host processing delay. We address this challenge in Section 4.2.1.

Probe service networks with low overhead. In order to accurately measure service network RTT, we need to probe the paths used by services. An intuitive idea is to construct probes with the same 5-tuples[2] as service flows. Then these probes will be routed to the same paths as the service flows by ECMP. In TCP networks, a flow corresponds to a process by its 5-tuple. While in RoCE networks, RDMA packets are encapsulated over UDP, and the process uses the internal 4-tuple[3] to identify a flow. In addition, the verbs API allows the application to determine the source port used in the outer 5-tuple. In this way, RoCE packets can be easily routed to the same paths as the service flow. The challenge then is to detect when service connections are established and closed in real time, and to obtain the service flow 5-tuple with low overhead. We address this challenge in Section 4.2.2.

Categorize anomalous data accurately. First, not all detected anomalies can be attributed to networks. For example, a timeout may be caused by a down host, which is not network-related. Using this data to locate network problems can be misleading. Thus, the first challenge in data analysis is to accurately determine whether an anomaly is caused by networks. We address this challenge in Section 4.3.1. In addition, network anomalies can be further attributed to RNICs or switches. It is also a challenge to accurately categorize network anomalies. We address this challenge in Section 4.3.2.

3.2 R-Pingmesh Framework

Figure 3 shows the framework of R-Pingmesh. R-Pingmesh is an end-to-end RoCE network latency probing system. The coverage of R-Pingmesh is all RoCE NICs as well as switches and links where RoCE traffic may exist. A RoCE cluster typically serves a specific service team, such as distributed machine learning or distributed storage. There is no RDMA communication between our different clusters. Therefore, in R-Pingmesh, RNICs within a cluster probe each other, and there is no probing between different clusters. Since a host can have multiple RNICs, the minimum probing granularity is RNIC-to-RNIC.

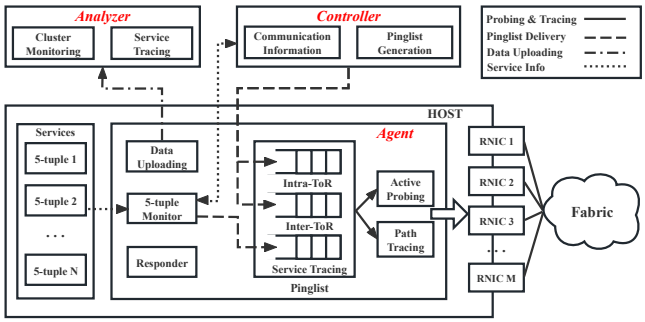

Figure 3: R-Pingmesh framework. Our RoCE servers have a TCP NIC for management. Interaction between the three modules is over the TCP network.

R-Pingmesh has three modules: *Agent*, *Controller*, and *Analyzer*. Agent is a service on each RoCE host and is responsible for constructing RoCE packets to perform probing tasks and for tracing the path of all probes it sends. Controller provides pinglists to all Agents to guide their probing targets. Analyzer is responsible for SLA tracking, detecting and locating anomalies, and assessing the impact of problems on services. R-Pingmesh has two functions: *Cluster Monitoring* and *Service Tracing*.

The purpose of *Cluster Monitoring* is to monitor the network SLA of RoCE clusters and locate cluster network problems. Agent constructs a UD (Unreliable Datagram) QP (Queue Pair) on each RNIC for probing and responding. The probe is a regular RoCE packet and suffers from all RoCE network problems. Probing with UD QP reduces the consumption of RNIC connectivity resources and can accurately measure network RTT and end-host processing delay. Controller provides two pinglists for each Agent: *ToR-mesh pinglist* and *inter-ToR pinglist*. ToR-mesh probing monitors the state of all RNICs under a ToR switch and identifies anomalous probes caused by RNICs in real time. Inter-ToR probing effectively covers all network links between ToR switches within the RoCE cluster to monitor switch and link problems. *Cluster Monitoring* tracks SLAs that include not only connectivity and packet drops, but also accurate network RTT and end-host processing delay. These metrics allow R-Pingmesh to detect and locate performance bottlenecks in addition to network failures. *Cluster Monitoring* is always on and independent of services. Therefore, it can detect network problems before services are running.

Service Tracing is independent of *Cluster Monitoring* and has three goals: monitor service network SLAs, measure service network RTT, and assess the impact of problems on services. The main difference between these two functions is that *the probing targets in Service Tracing are obtained by monitoring 5-tuples of service flows, not from Controller.* When Agent detects the establishment of RDMA connections, it parses the 5-tuples of these connections and constructs packets with the same 5-tuples to probe the service network. The execution of *Service Tracing* depends on whether there are RDMA connections on the local host. By probing the paths used by

[2]Source IP and port, destination IP and port, and transport layer protocol. For RoCE packets, the destination port is 4791 and the protocol is UDP.
[3]Source GID (Global Identifier), source QPN (Queue Pair Number), destination GID, and destination QPN.

service flows, R-Pingmesh can continuously track service network SLAs. In addition, by monitoring service performance degradation and judging whether a problem lies in the service network, R-Pingmesh can assess the impact of problems on services.

4 R-PINGMESH DESIGN

4.1 R-Pingmesh Controller

Store the latest RNIC communication information. The first function of Controller is to act as a central database to store the latest communication info for RNICs managed by Agents. When an RNIC wants to communicate with a remote RNIC over RoCE networks, it needs to obtain the remote RNIC's communication info, including the Global Identifier (GID) and Queue Pair Number (QPN), which are similar to the IP address and port number in the TCP network. We can reserve a TCP port on all hosts for Pingmesh Agent to listen on. However, the QPN can only be obtained after Agent starts and creates a QP on the RNIC. In addition, when an Agent restarts due to host reboot, the QPN of all RNICs it manages will change. Therefore, Controller must store the latest communication info of all RNICs to ensure that the pinglist it provides to each Agent is valid. To meet this end, each time Agent starts, it reports the communication info of all RNICs in the host to Controller.

Create ToR-mesh and inter-ToR pinglists for each RNIC in Cluster Monitoring. For each RNIC, the ToR-mesh pinglist contains the communication info of all RNICs within the same ToR switch. This allows R-Pingmesh to quickly identify anomalous RNICs and determine whether anomalous probes are caused by RNICs (described in Section 4.3.2). In addition, network links between different ToR switches must also be covered with sufficient probes. Since our data center networks are symmetric and use ECMP to hash flows to different paths, Controller first uses Equation (1) [51] to calculate the number of 5-tuples each ToR switch needs (k) to cover all cross-ToR links with high probability. In this equation, N is the number of parallel cross-ToR paths, and P is the probability that k 5-tuples can cover all N paths based on ECMP. In practice, we set P to 0.99 to get k.

$$\arg\min_{k} \sum_{i=1}^{N} (-1)^{i+1} \binom{N}{i} \left(1 - \frac{i}{N}\right)^{k} \leq 1 - P, \text{ where } k \geq N \qquad (1)$$

Then, Controller randomly selects k inter-ToR 5-tuples for each ToR switch, aggregates them into inter-ToR pinglists based on the source RNIC IP, and distributes these pinglists to the corresponding Agent. Based on the expected probe frequency on each link, Controller can further get the probe interval for each RNIC. In addition, Controller periodically changes the 5-tuples used in inter-ToR probing to detect problems that can only be triggered by certain 5-tuples, such as silent packet drops for certain 5-tuples. Since probing is performed within a RoCE cluster, this two-tier probing approach is sufficient in most scenarios.

Respond to Agent in Service Tracing. After the local Agent obtains 5-tuples used by service flows, it constructs RoCE packets with the same 5-tuples for probing. To ensure that the target RNIC can recognize the probe, the local Agent needs to request the latest communication info of the target RNIC from Controller.

Table 1: Feature comparisons of the three QP types

Features	RC	UC	UD
Accurate RTT Measurement	✗	✓	✓
Connection Overhead	High	High	**Low**

4.2 R-Pingmesh Agent

4.2.1 Probe RTT and Processing Delay with UD QP.

The first design consideration for active probing is which QP type to choose. We have two requirements: (1) it is able to accurately measure network RTT and end-host processing delay, and (2) it has low connection overhead. Table 1 compares the relevant features of all three QP types.

First, as shown in Figure 4, accurate measurement of network RTT requires four timestamps: ② prober RNIC sends the probe, ③ responder RNIC receives the probe, ④ responder RNIC sends the ACK, and ⑤ prober RNIC receives the ACK. Then we could use (⑤-②)-(④-③) to get RTT without clock synchronization between prober and responder RNICs. However, instead of providing explicit timestamps on sending and receiving packets, commodity RNICs only provide timestamps when generating CQEs.

For all three QP types, a CQE with an RNIC timestamp can be generated when an RDMA message is successfully received (③ and ⑤). However, in RC, the sender RNIC only generates a CQE after it receives the ACKs of all the message packets it has sent to the destination. As a result, we cannot get ② and ④ in RC. In UC and UD, the sender RNIC does not guarantee data reliability, and the sender RNIC generates a CQE immediately after the message is sent to the wire. Therefore, with UC or UD QP, Agent can accurately measure all four timestamps (②③④⑤). On this basis, the responder's processing delay can be calculated by ④-③. And by measuring ① and ⑥ at the prober application, Agent can get the prober processing delay by (⑥-①)-(⑤-②) without clock synchronization between the host CPU and RNIC.

In terms of connection overhead, if Agent uses connection-based QP (RC or UC), for each destination RNIC, the local RNIC must create a QP and connect it to the destination QP, which consumes the limited QP context (QPC) cache in the RNIC [21–23]. To ensure coverage of network links within a cluster, an RNIC can probe hundreds of other RNICs. With RC or UC QP, Agent will consume a lot of QPC cache in the RNIC, increasing the probability of QPC cache misses for service traffic, which will result in RNIC performance degradation. In contrast, UD is connectionless. With UD, Agent only needs to create one QP on each RNIC, significantly reducing the connection overhead. Therefore, Agent uses UD for active porbing, which can accurately measure RTT and processing delay with low connection overhead.

Next, we introduce the probing process for network RTT and end-host processing delay. Each Agent acts as both a prober and a responder. As a prober, Agent selects target RNICs from pinglists and constructs RoCE packets to probe these RNICs. As a responder, Agent responds to all received probes with ACKs, which carry the processing delay on the responder. As shown in Figure 4, the probing steps are as follows.

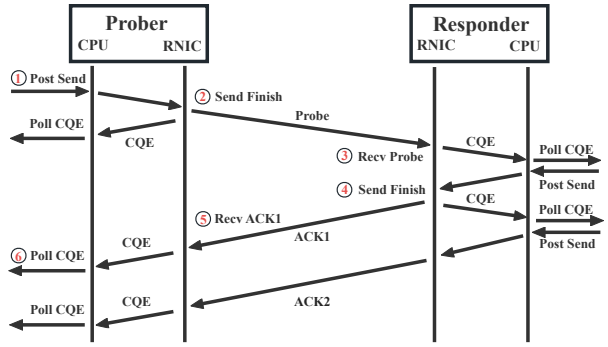

Figure 4: Probing process for network RTT and end-host processing delay with UD QP. The timestamps that need to be measured are marked with circles (①②⑤⑥ on the prober and ③④ on the responder).

1 The prober sends a probe to the target RNIC and gets a CQE with timestamp ② when the RNIC finishes sending.
2 Upon receiving the probe, the responder gets a CQE with timestamp ③ and responds to the prober with an ACK packet. When the RNIC finishes sending the ACK packet, the responder gets a CQE with RNIC timestamp ④.
3 The responder obtains its *processing delay* according to ④-③, writes the delay into the second ACK packet's payload, and sends the packet to the prober.
4 After receiving the first ACK, the prober gets a CQE with timestamp ⑤ and gets its *processing delay*: (⑥-①)-(⑤-②).
5 After receiving the second ACK, the prober gets the *responder's processing delay* ④-③, and the *network RTT*: (⑤-②)-(④-③).

Note that the responder can only get the timestamp ④ after sending the first ACK. Therefore, the responder has to send the second ACK to carry its processing delay. In addition, in steps 4 and 5, if the prober does not receive all two ACKs in a certain period of time, the probe or ACK packets may be dropped for some reason, and this probing will be marked as a timeout.

4.2.2 *Monitor the 5-tuples of Service Flows.*

To obtain the 5-tuple of service flows in real time with low overhead, we use eBPF (extended Berkeley Packet Filter) to trace the call to the kernel functions `modify_qp` and `destroy_qp`. Our services typically use RC QP to ensure reliable delivery. When connecting RC QP on two RNICs, `modify_qp` is called to modify the QP attribute including the 5-tuple. In addition, when the service closes a connection, it calls `destroy_qp`. By tracing these two verbs, Agent can know exactly when a connection is established or closed and obtain its 5-tuple. Since these kernel verbs are called only when connections are established or closed, which is not frequent, tracing these verbs has little impact on service performance. Note that this method can easily be deployed on RDMA servers without relying on special hardware or firmware.

After obtaining 5-tuples of service flows, Agent sends the destination RNIC IP to Controller to obtain its communication info. Agent then uses these 5-tuples and destination communication info to create pinglist entries and pushes them to the service tracing pinglist. When Agent's active probing module finds targets in this pinglist, it probes these targets using the given 5-tuples. If Agent detects that a connection is closed, it removes the corresponding entry from the pinglist. After all service connections are closed, the service tracing pinglist will be empty and Service Tracing in Agent will be paused until a new connection is established.

4.2.3 *Path Tracing and Data Uploading.*

The path tracing module is responsible for tracing the paths of probes and their ACKs so that Analyzer can locate network problems and assess their impact on service performance. This raises a design discussion: *should we trace probes continuously or only when they encounter a network problem?* We choose the former, and for each probe 5-tuple and its ACK, Agent periodically `Traceroute` their latest paths. Our reason is two-fold. First, in the case of a persistent failure, such as a link down, the replayed dropped packets will be rehashed to other normal links, leading to inaccurate fault localization. In addition, continuous path tracing allows Analyzer to locate network problems immediately as they occur. However, since `Traceroute` consumes switch CPU, data center switches typically limit the frequency of responses to it. Therefore, we limit the frequency of `Traceroute` in Agent to avoid overloading the switches. The data uploading module is responsible for uploading the network RTT, timeout events, end-host processing delay, and each probe's information and path to Analyzer for analysis.

4.3 R-Pingmesh Analyzer

4.3.1 *Is a Timeout Probe Caused by Network Problems?*

Among all uploaded end-to-end probe results, anomalous data can be categorized into two types: *timeout* and *high RTT*. High RTT is usually caused by network problems, such as network congestion. However, timeout can be caused by *network problems*, such as flapping ports/RNICs, *non-network problems*, such as an accidental host down, and *probe noise*, such as QPN reset. QPN reset occurs when the target host or Agent reboots, in which case the Agent QPN of the target RNIC changes. And if the probe is still using the outdated QPN, the remote RNIC will drop it and trigger a timeout. Using timeout caused by non-network problems and probe noise to troubleshoot network problems can be misleading. Therefore, **Analyzer must first rule out non-network timeout and noise.**

For a timeout probe, we can first check whether Agent on the target host is still uploading data. If not, the timeout probe is caused by *host down*. Next, for probe noise caused by QPN reset, an intuitive solution is to update all pinglists with the latest QPN in real time. However, this introduces a high processing overhead for both Agent and Controller. As a workaround, Analyzer can compare the target RNIC's QPN in the probe packet with the one stored in Controller. If they do not match, this timeout is caused by a *QPN reset* problem. And in the next round of pinglist updates, the outdated QPN will be replaced with the latest ones.

4.3.2 *Detect Anomalous RNICs in Real Time.*

After excluding non-network timeout and probe noise, all anomalous probes are caused by networks, including *RNIC problems*[4] and *switch network problems*. If we can **detect anomalous RNICs**, we can further **filter out anomalous probes caused by RNICs** and accurately locate switch network problems.

[4]It is hard to use probing to directly judge whether the problem is in the RNIC, the RNIC link, or the switch port connected to the RNIC. Thus, Analyzer considers all these problems as *RNIC problems* and other network problems as *switch network problems*.

Our experience in detecting anomalous RNICs is to introduce minimal uncertainty. In Cluster Monitoring, *ToR-mesh probing* monitors RNIC states in real time. Since probing within a ToR switch involves only two links, if a probe from RNIC A to RNIC B shows anomalies, the problem is either at A, B, or the ToR switch. If many probes to an RNIC show anomalies at the same time, that RNIC is probably abnormal. Meanwhile, any anomalous probe data involving that RNIC should not be used to locate switch network problems. The premise of this approach is that for each RNIC, *there should be enough probes from other RNICs under the same ToR switch over a fine-grained period of time* so that we can monitor the state of each RNIC in real time. Although Pingmesh has also adopted intra-ToR probing, it cannot detect anomalous RNICs in real time because it does not guarantee the sending frequency of probe packets.

4.3.3 Locate Switch Network Problems.

According to Sections 4.3.1 and 4.3.2, we can **get anomalous probes caused only by switch network problems**. As shown in Algorithm 1, with the paths of these probes and their ACKs, we can use a simple voting mechanism to determine abnormal switches[5] and links. The core idea, derived from *binary network tomography* [4, 6, 17], is to find common links in the paths of anomaly probes, and these links are most likely to be anomalous. When Analyzer detects that the number of uploaded anomalous probes caused by switch network problems exceeds a threshold, it traverses the paths of these probes and their ACKs one by one and counts the number of times each link is traversed. After this process, the link with the highest number of votes is the most suspicious. Note that for anomalous probes uploaded by Cluster Monitoring and Service Tracing, Analyzer analyzes them individually.

4.3.4 Assess the Impact of Problems on the Service.

After the above analysis, it is straightforward to assess the impact of a problem on services. First, to assess whether the service performance is severely affected, we need to monitor its typical performance metrics, such as the training rate or the average network throughput in DML, and set a *maximum tolerable threshold* [6] for metric degradation. If the metric degradation exceeds this threshold, we consider that the service performance is severely affected and it is necessary to address this problem immediately.

If a problem is (1) detected by Service Tracing or (2) detected by Cluster Monitoring and lies in the service network, then this problem can affect the service performance (*P0* or *P1*). At the same time, if the performance metric degradation exceeds the threshold, the problem should be resolved immediately (*P0*). Otherwise, operators should consider both the performance loss during solving this problem and the benefits gained after solving this problem to determine whether this problem should be resolved immediately (*P1*). If a problem is detected by Cluster Monitoring and does not occur in the service network, then it is a problem that does not affect service performance (*P2*). In addition, if no *P0* or *P1* problem is detected when service performance degrades, then the service network is innocent.

[5]By simply replacing all "$link$" in Algorithm 1 with "$switch$", Analyzer can identify the most suspicious switches.

[6]The form of the threshold is not fixed. For example, it could be "metric degradation to a certain value" or "the total number/duration of metric degradations reaches a certain value". The setting of this threshold needs to be discussed with the service team.

Algorithm 1 Identify the Most Suspicious Switch Links

Input: *abnormal paths*
Output: *the most suspicious links*
1: **function** DetectAbnormalLinks()
2: $InitLinkState()$
3: **for** $path_j$ in *abnormal paths* **do**
4: **for** $link_i$ in $path_j$ **do**
5: $link_i.state \leftarrow abnormal$
6: $link_i.abnormal\_cnt++$
7: **return** *abnormal links* with the largest $abnormal\_cnt$
8: **function** InitLinkState()
9: **for** $link_i$ in *all network links* **do**
10: $link_i.state \leftarrow normal$
11: $link_i.abnormal\_cnt \leftarrow 0$

5 IMPLEMENTATION

In R-Pingmesh, Controller and Analyzer were quickly developed based on Pingmesh. We spent most of our time on Agent, which was built from scratch. This section presents some key implementation details and running parameters.

Agent is implemented based on the verbs API [28]. It launches four independent threads on each RNIC, responsible for *ToR-mesh probing*, *inter-ToR probing*, *service tracing probing*, and *responding to probes*. Each probing thread periodically selects a target RNIC from the corresponding pinglist and probes it using the source port specified in the pinglist, which is done by setting the *flow label* in the verbs API. After receiving a probe, the responder uses the same source port as the probe to send ACKs, mimicking how the RNIC sends ACKs in RC QP used by services. All running parameters in Agent are set by Controller through pinglists. The timeout for each probe is 500ms. And the payload of the probe and ACK is 50 bytes, including some necessary fields such as the responder processing delay. To ensure that the communication info in the service tracing pinglist is valid, Agent pulls the latest communication info for all target RNICs from Controller every 5 minutes. Every 5 seconds, Agent uploads all probe information to Analyzer.

Controller updates the ToR-mesh/inter-ToR pinglists in each Agent every 5 minutes to ensure that the target info is valid. And it changes 20% of the 5-tuples in each inter-ToR pinglist every hour. The ToR-mesh probing frequency is 10 packets per second to detect anomalous RNICs at 100ms granularity. The inter-ToR probing frequency varies with cluster network topologies. For each cluster, Controller calculates a probing frequency to ensure that each link above ToR switches sends more than 10 probes per second per direction to detect switch network problems at 100ms granularity. And the probing interval in Service Tracing is 10ms to better capture network congestion. The probing frequency used in R-Pingmesh can effectively detect service-impacting problems with low overhead and does not introduce probing noise due to oversensitivity.

Analyzer uses 20s as the analysis period to detect and locate problems in real time. Every 20s, it processes all data uploaded by Agents in the last 20s. It first checks each timeout probe. If an Agent has not uploaded data for more than 20s, that host is considered down, and all timeout probes to RNICs on that host are considered caused by *host down*. It then filters out timeout probes caused by *QPN reset* as described in Section 4.3.1. Next, it analyzes the ToR-mesh probing

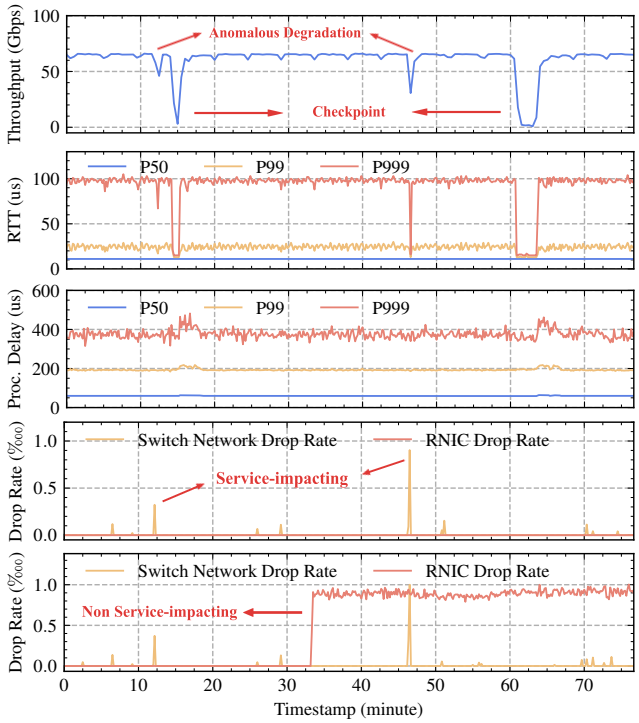

Figure 5: The monitoring of network SLAs over a period of time. From top to bottom are (a) the average throughput of the training task; (b) the probed RTT of the service network; (c) the measured end-host processing delay in the service network; (d) the probe drop rate of the service network; and (e) the probe drop rate of the cluster network.

results over the last 20s to determine the state of each RNIC. If more than 10% of the probes to an RNIC experience a timeout, that RNIC is considered anomalous. In this analysis, and for the next minute, Analyzer considers all timeout probes to or from that RNIC caused by *RNIC problems*. After this step, the remaining timeout probes are considered caused by *switch network problems*. Next, in addition to locating switch network problems, Analyzer aggregates all latency and timeout data in this analysis period to determine (1) *RNIC drop rate*, (2) *switch network drop rate*, and the distribution (P50 to P999) of both (3) *end-host processing delay* and (4) *network RTT* for each RoCE cluster and service network. In this way, Analyzer tracks SLAs for both cluster networks and service networks.

6 EVALUATION

We have deployed R-Pingemsh for over 6 months on multiple DML RoCE clusters totaling tens of thousands of RDMA NICs. These clusters include both on-premises and public cloud clusters. We evaluate the performance of R-Pingmesh on a large-scale DML RoCE cluster from the following perspectives: (1) SLA monitoring, (2) localization accuracy, and (3) Agent overhead.

Network topology. The RoCE cluster has 3 tiers of switches to accommodate thousands of GPU servers, and different tiers are interconnected in a CLOS-like topology. The switches use Broadcom Tomahawk 4 chips, and each switch has 64 × 400Gbps ports for a total bandwidth of 25.6Tbps. The oversubscription ratio at each tier is 1:1 (i.e., 32 ports for downlink and 32 ports for uplink).

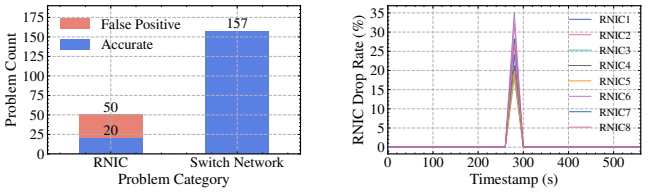

Figure 6: (left) Problems detected and located by R-Pingmesh. (right) Occupation of the Agent CPU by the service application causes probe packets to be "dropped" simultaneously by multiple RNICs on a host.

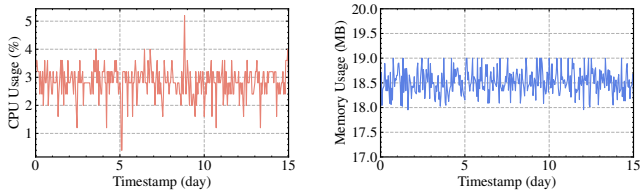

Figure 7: CPU and memory usage of R-Pingmesh Agent in half a month. The CPU usage is based on a single CPU core.

SLA monitoring. Figure 5 shows the monitoring of both cluster network SLAs and service network SLAs over a period of time in the cluster. To ensure fault tolerance, (a) our hosts periodically use TCP to upload their trained models to the storage cluster in checkpoints. During this time, the RoCE network is briefly idle, and (b) the network RTT simultaneously decreases. As TCP is CPU intensive, (c) the end-host processing delay also increases. *This indicates that R-Pingmesh can* ***accurately measure RTT and processing delay and use them to reflect service states.*** In addition to the checkpoints, (a) there are two anomalous throughput degradations during this period, and (b) the RTT also degrades. With Analyzer's real-time analysis and accurate distinction between RNIC and switch network problems via ToR-mesh probing, both (d) Service Tracing and (e) Cluster Monitoring *detect packet drops on switches in an analysis period (i.e., 20s).* This confirms that both degradations are caused by packet drops in the service network (P0 or P1). In addition, during this time, (e) Cluster Monitoring detects an anomalous RNIC that continues to drop packets. However, (d) Service Tracing does not see any RNIC drops. Therefore, we conclude that the problem is outside the service network and does not affect the service (P2). ***With SLA monitoring, we can detect and categorize problems in real time and determine their impact on services.***

Localization accuracy. We use data from a recent month in the DML cluster to evaluate the localization accuracy of R-Pingmesh. As shown in Figure 6 (left), R-Pingmesh reported 207 problems that it detected and located in this month. *All of these problems are quickly located in one analysis period (i.e., 20s).* By identifying each one, ***85% of reported problems are accurate, where all 157 reported switch network problems are accurate*** *due to the accurate distinction between RNIC and switch network problems by ToR-mesh probing.* However, only 20 of the 50 RNIC problems are confirmed, and we find no RNIC or host anomalies in the remaining 30 problems. After analysis, we realize that the root cause is the occupation of Agent CPU by the service, resulting in a high processing delay in responding ACKs, and thus a timeout in the prober. As shown in Figure 6 (right), a common feature of these problems is that *probes to multiple RNICs of a host are transiently "dropped" at the same time.* Since the probability of multiple RNICs on a host being abnormal

Table 2: Problems found by R-Pingmesh during deployment. (∗) indicates that this problem will cause services to fail.

No.	Problem Categories	Root Cause Categories	Root Causes
1	Failures	Hardware Failures	RNIC or switch port flapping.
2			Packet drops on RNICs or switches due to packet corruption.
3			Accident RNIC down. ∗
4			Accident host down. ∗
5			PFC deadlock caused by anomalous switches. ∗
6		Misconfigurations	Lack of additional routing configurations for RNICs. ∗
7			RNIC GID index missing. ∗
8			Switch ACL (Access Control List) configuration error. ∗
9			PFC unconfigured or misconfigured PFC headroom on switches.
10	Performance Bottlenecks	Network Congestion	Uneven load balance.
11			Interference between services.
12		Intra-host Bottlenecks	CPU overload.
13			The speed or width of the RNIC or GPU PCIe link is downgraded.
14			Incorrect PCIe or RNIC configurations (e.g., wrong ACS or ATS configurations [29]).

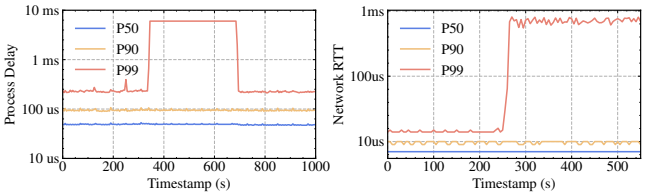

Figure 8: (left) CPU overload results in high processing delay on some hosts. (right) A PFC storm results in a high P99 network RTT.

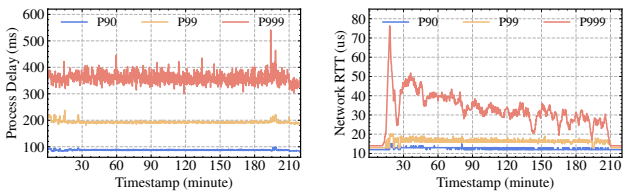

Figure 9: The service training throughput continues to decrease. At the same time, the network RTT also decreases and the processing delay is stable, indicating that no network or CPU bottleneck occurs.

at the same time is very low, we can use this feature to filter out probe noise. Another option is to check if the responder has an abnormally high processing delay. If so, the timeout is possibly caused by CPU overload. In subsequent deployments, Analyzer uses these methods to effectively eliminate false positives caused by CPU overload and accurately measure network timeout.

Agent overhead. In Agent, CPU is primarily consumed by sending/receiving probes and ACKs. For each RNIC, the probe frequency is typically less than 150 packets per second, which is not CPU intensive. Memory is primarily consumed by temporary local caching of probe results. Therefore, the overhead of Agent scales linearly with the number of RNICs on the host. During deployment, we continuously monitored the Agent's CPU and memory usage. As shown in Figure 7, ***the average CPU and memory consumption of Agent on hosts with 8 RNICs is approximately 3% and 18.5MB, respectively***. In addition, because the probes and ACKs are small packets, the bandwidth consumption on each RNIC is less than ***300Kbps***, which is negligible for mainstream 100/200 Gb/s RNICs in the data center.

7 EXPERIENCES AND FUTURE WORK

7.1 Problems Found by R-Pingmesh

As shown in Table 2, we summarize the problems detected and located by R-Pingmesh during the deployment. Based on the phenomena caused by these problems, we categorize them as *failures* and *performance bottlenecks*. Failures typically result in packet drops and unreachable hosts or RNICs. In these cases, Agent can detect probe packet timeout. On the other hand, performance bottlenecks result in increased network latency, increased processing delay, and degraded throughput. In these cases, Agent will detect high network RTT or processing delay. Based on the root cause of these problems, they can be further categorized as *hardware failures*, *misconfigurations*, *network congestion*, and *intra-host bottlenecks*.

Hardware failures. Of all the hardware failures, (#1) is the most common. When flapping occurs, the state of the RNIC or switch port changes frequently between *up* and *down*, causing packet drops. Early in the deployment, we noticed frequent RNIC flapping. After a long-term analysis with our RNIC vendor, we realized that the root cause was *an incompatibility between the RNIC and the cable*. After replacing the cable, the frequency of RNIC flapping decreased significantly. Severe flapping can cause service flow timeout. In this case, if the RDMA connection retransmits more than a certain number of times but cannot succeed, it will be broken, causing training tasks to fail. To ensure stable operation, our service team sets the retransmission count to the maximum (i.e., 7 times in RDMA) and sets the retransmission timeout to a higher value to avoid consuming all retransmission counts during flapping.

Damaged or aged optical fibers and dust in the optical module can cause packet corruption, resulting in packet drops on switches or RNICs (#2). These problems will degrade throughput and training rates. In addition, hardware failures can cause an accidental RNIC or host down (#3 and #4). When these problems occur in the service network, service connections will be broken, causing training tasks to fail. In public clouds, we found several network connectivity problems caused by PFC deadlock (#5). In one of them, two switch ports on a link continue to send PFC pause frames to each other, completely blocking all traffic going through that link. From the service perspective, some RDMA connections cannot communicate, and our tenant suspects that the root cause lies in the ACL (Access Control List) configuration on the switches. Meanwhile, R-Pingmesh detects a large number of timeout probes. Based on their 5-tuples, R-Pingmesh accurately diagnosed the abnormal link. Normally, the PFC watchdog on switches will temporarily disable PFC when it detects a deadlock. However, these anomalous switches were not functioning properly in these cases.

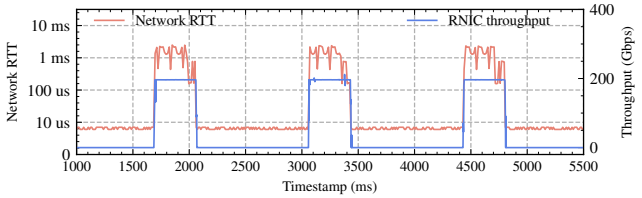

Figure 10: Service Tracing probes sent by an RNIC can accurately capture periodic All2All traffic and network congestion in a DML cluster.

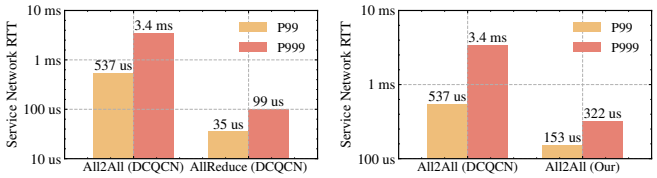

Figure 11: Network RTT measured by R-Pingmesh reflects (left) the degree of network congestion in different communication modes and (right) the performance of congestion control algorithms.

Misconfigurations. The most common misconfiguration is (#6). Our RoCE NICs require additional routing configuration for RDMA communication, which is configured by scripts after the host boots. However, these configurations may fail for some reason. In addition, our RoCE network uses a specific GID index for RDMA communication. However, (#7) RNICs may lose this index in some cases. In (#6) and (#7), these anomalous RNICs are unreachable, and the task will fail if it selects a host with these RNICs to participate in the training. R-Pingmesh uses this specific GID index to probe and can quickly detect anomalous RNICs. In addition to RNIC misconfigurations, switch misconfigurations occur occasionally. In our public cloud clusters, different tenants are isolated from each other by the switch ACL. As a cluster has multiple tenants, and tenants may frequently increase or decrease the number of hosts they own, the ACL needs to be configured frequently. And the ACL can be misconfigured due to operator oversight or errors in the configuration script, resulting in connectivity failures between different RNICs of the same tenant (#8). R-Pingemsh can effectively detect ACL anomalies by randomly probing between all RNICs within the cluster. In addition, operator oversights can lead to unconfigured PFC or misconfigured PFC headroom on switches (#9), resulting in packet drops during heavy congestion. By analyzing the timeout 5-tuples, R-Pingmesh can accurately locate abnormal switches.

Network congestion. Network congestion is common in large clusters. In addition to the congestion caused by load imbalance (#10) mentioned in Section 2.3, we also found congestion caused by interference between different services. Although ACL isolates servers from different tenants in public clouds, traffic from different tenants can still share some network links and cause congestion (#11). During the deployment, R-Pingmesh found that the Service Tracing results from two different tenants indicated the same congested link, and we quickly diagnosed the root cause of the problem. Furthermore, by detecting network hotspots and identifying the flows passing through them, R-Pingmesh can guide the load balancing of service flows. We discuss this in Section 7.3.

Intra-host bottlenecks. During deployment, R-Pingmesh detected three types of intra-host bottlenecks. The first is the CPU overload (#12) mentioned in Section 2.3. As shown in Figure 8 (left), R-Pingmesh can easily detect this problem by accurately measuring the end-host processing delay. R-Pingmesh also detected intra-host bandwidth degradation caused by PCIe downgrades (#13) and misconfigurations (#14), where PCIe downgrades can be caused by loose PCIe interfaces, dust on PCIe interfaces, or hardware failures. Intra-host bandwidth degradation causes RNICs to send PFC pause frames and can result in a PFC storm [29]. R-Pingmesh has found several such cases, and Figure 8 (right) shows one of them. In this case, Service Tracing found a high P99 network RTT, and ToR-mesh probing also found a high RTT to an anomalous RNIC.

7.2 Is It a Network Problem?

When a service fails, R-Pingmesh can determine if the root cause is in the network by checking if there is a large number of probe timeout in the service network. And when service performance degrades, R-Pingmesh can determine if there is a bottleneck in the network or CPU by checking network RTT and end-host processing delay, as shown in Figure 8. In one case, the service training rate and average throughput continue to decrease, and our service team suspects that the network is congested due to uneven load balancing. We then check the metrics in R-Pingmesh. As shown in Figure 9, the RTT is also decreasing and the processing delay is stable, proving that there is no bottleneck in network or CPU processing. Upon further analysis, our service team finds bugs in the training code, which result in continuous degradation of compute performance and appear as continuous degradation of network throughput at coarse monitoring granularity.

There are some end-host problems that R-Pingmesh cannot directly detect, such as GPU underclocking, GPU out of memory, and GPU down, which can degrade training performance or even cause training tasks to fail. Nevertheless, if Service Tracing finds no anomalies in service networks, operators can be confident that *the problem is not caused by host down, CPU overload, anomalous RNICs or networks.* In this way, R-Pingmesh effectively helps to narrow down the possible causes of service problems and avoid wasting time diagnosing root causes in the wrong directions.

7.3 RTT Measurement and Application

Use service traffic characteristics to find hotspots. DML services constantly repeat computation and communication. During computation, the network is idle; while during communication, the network is heavily loaded. Each cycle takes only a few seconds. We have considered setting the probing interval in Service Tracing to 1ms or even lower to effectively capture the network hotspot. However, this has a high probing overhead. In fact, we can still efficiently detect hotspots with a lower probing frequency. In DML, connections remain unchanged during the training process, and their paths are fixed unless some links or switches fail. Therefore, *if there is a network bottleneck, it will appear in every communication period.* By probing for multiple rounds, the probes that go through the hotspot must experience congestion several times, and the tail RTT of these probes will be high. *A key insight is to probe randomly.* We need to prevent probes for some paths from being sent only when the network is idle or heavily loaded. Therefore, Agent shuffles the pinglist in each round of service tracing probing.

Use tail RTT to reflect the degree of network congestion. Our DML services typically use two communication modes, AllReduce [18] and All2All [27], where AllReduce is less congested and All2All

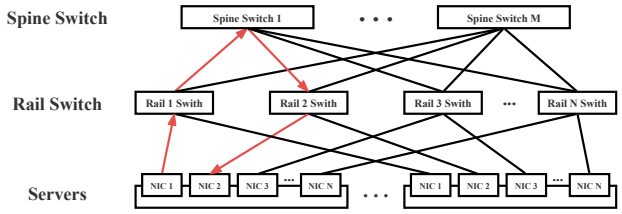

Figure 12: A two-tier rail-optimized cluster. Each host has multiple NICs connected to different rail switches, and all rail switches are connected to all spine switches in a full-bisection fashion. Traffic between NICs on the same host traverses the top-tier switches in the cluster. The red lines with arrows show such an example.

can cause severe network congestion. Figure 10 shows that Service Tracing probes sent by an RNIC can accurately capture the periodic All2All traffic and network congestion, ensuring that R-Pingmesh can effectively detect network hotspots. As shown in Figure 11 (left), by continuously tracking the network RTT of services using All2All and AllReduce, R-Pingmesh can reflect their difference in network congestion with the tail RTT. Our experience shows that All2All has a large room for congestion control optimization. As shown in Figure 11 (right), compared with the default DCQCN algorithm [55] on commodity RNICs, our self-developed congestion control algorithm can effectively reduce the tail RTT and improve the training throughput. R-Pingmesh can effectively evaluate this optimization using RTT metrics.

Use Service Tracing to guide load balancing. With Service Tracing, R-Pingmesh can accurately measure the RTT of all paths used by a service and determine which service flows are contributing to the congested link. This information can be used to further guide load balancing: *the service can modify the source port of some congested flows by calling* `modify_qp` *to reroute them to parallel paths and alleviate congestion.* This centralized load balancing mechanism is not responsive enough, but it is still effective for the service that uses long connections, such as DML. In DML, connections are typically established before training begins and closed when training is complete. In this scenario, congestion caused by hash collisions can be effectively resolved by rerouting congested flows.

7.4 Discussion

Use the network topology feature to optimize the system. The framework of R-Pingmesh introduced in this paper ensures that it can adapt to clusters with any network topology. In practice, we can leverage the feature of network topology in a cluster to simplify the system design and improve the localization accuracy. For example, for hosts with multiple NICs, there is an emerging networking method called *rail-optimized networking* [9, 33, 34, 45]. Figure 12 shows a small-scale two-tier *rail-optimized* cluster. In this topology, instead of connecting all NICs on a host to the same ToR switch, different NICs on a host are connected to different ToR switches (rail switches), and NICs with the same index on different hosts form a rail. This topology takes advantage of multiple NICs on a host and *allows more hosts to be under the same ToR switch, minimizing cross-ToR traffic and hash collisions.* In a rail-optimized cluster, inter-rail traffic (e.g., probes between NICs on the same host) must traverse the top-tier switches in the cluster. Therefore, *in Cluster Monitoring, we can have RNICs on a host probe each other.* With enough 5-tuples, all links in the cluster can be covered. *Since the target RNIC is on the local host, Cluster Monitoring does not need to rely on Controller to generate pinglists.* In addition, the responder RNIC does not need to send ACKs, and Agent can detect *one-way timeout* and measure *one-way RTT*. With one-way probe results, R-Pingmesh can locate network problems more accurately.

Hierarchically aggregated probe results can be misleading. In Cluster Monitoring, R-Pingmesh aggregates probe results at different tiers (e.g., servers, ToR switches, and clusters) to evaluate network SLAs at each tier, and we also tried to use hierarchical aggregation in Service Tracing. However, the aggregated data in Service Tracing can be misleading due to the uneven distribution of servers in the service network. For example, a service may select only two servers under a ToR switch. In this case, if one of the two servers fails, the aggregated drop rate for that ToR switch is 50%, misleading us to believe that the ToR switch is anomalous. The root cause of the misleading result is that the samples being aggregated are too few. Therefore, in Service Tracing, we only evaluate SLAs for the entire service network and for each server.

Comparison of different path tracing methods. To ensure that R-Pingmesh can be easily deployed, it uses `Traceroute` for path tracing instead of other advanced switch capabilities such as ERSPAN (Encapsulated Remote Switch Port Analyzer) or INT (Inband Network Telemetry), which are not supported by some legacy switches. However, `Traceroute` consumes switch CPU and therefore the switch limits the response frequency to it. In contrast, ERSPAN and INT do not consume switch CPU, and switches do not impose frequency constraints on them, allowing them to trace path information in more real time. In addition, INT allows R-Pingmesh to obtain queuing information on switch ports, which can help locate bottlenecks more accurately when R-Pingmesh detects network congestion. We have decoupled the path tracing module from the active probing module so that R-Pingmesh can easily adopt these more powerful switch capabilities in new clusters.

7.5 Limitations and Future Work

Adapt R-Pingmesh to IB clusters. In addition to the dominant RoCE clusters, we have some Infiniband (IB) clusters for DML services. Although IB clusters are highly integrated and use dedicated RNICs and switches, our experience shows that they are far from trouble-free. IB clusters also suffer from RNIC and switch network problems and require an effective network monitoring and diagnostic system. Since IB clusters also support the verbs API, R-Pingmesh can be deployed directly in IB clusters and is still effective in detecting IB network problems. However, IB clusters may use Adaptive Routing [35] for load balancing, and a packet may be randomly routed to parallel paths, making it difficult to accurately trace probe paths to further locate switch network problems. How to accurately locate a switch network problem detected by R-Pingmesh in IB clusters is our future work.

Detect anomalous GPUs. R-Pingmesh uses CPU memory for RDMA probing. As a limitation, it cannot directly detect some GPU problems, such as GPU down and high GPU read/write latency due to GPU failures. In training clusters, if R-Pingmesh can probe the path of training traffic within RDMA servers (i.e., RNIC <-> GPU) in

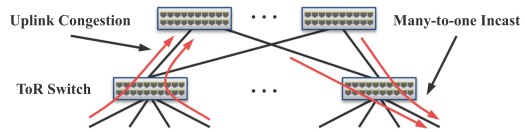

Figure 13: Two of the most common causes of congestion detected by R-Pingmesh: *many-to-one incast* and *uplink congestion caused by hash collisions.*

real time, it can further effectively detect these GPU problems. An intuitive solution is using GPU memory to perform RDMA probing. However, a GPU process uses at least a few hundred MB of GPU memory (measured on an Nvidia A100 GPU [32], and varies by GPU type), most of which is consumed by the process context. Because GPU memory is valuable, service teams typically cannot afford this overhead. How to monitor the path between RNIC and GPU in the end host with low overhead is our future work.

Automatically diagnose root causes. While R-Pingmesh can quickly detect and locate anomalies, it is hard to diagnose the root cause solely from the probing results. For example, if ToR-mesh probing detects that an RNIC keeps dropping packets, the anomaly may occur on the RNIC, the cable, or the switch port connected to the RNIC. In addition, packet drops on switches can be caused by port flapping or packet corruption. Currently, inferring the root cause of these anomalies requires our operators to further examine anomalous counters and logs and rely on their experience. Integrating the probing results with more information, such as counters and logs, and using efficient methods, such as decision trees, for automatic root cause diagnosis is our future work.

Minimize the impact of hardware failures. Although R-Pingmesh can effectively detect hardware failures such as RNIC down, RNIC or port flapping, and packet corruption, if these problems occur during service runtime, they can cause severe service performance degradation or even task failure. Resolving these problems to restore service performance typically requires operator intervention or even a task restart, which is inefficient. To minimize the impact of hardware failures on service performance, we have three directions for future work. (1) Monitor anomalous metrics on RNICs and switch ports, such as CRC (Cyclic Redundancy Check) errors, to *predict or detect anomalous RNICs and ports as early as possible* and replace or isolate them. (2) If a switch port drops packets anomalously, *automatically determine whether to isolate the anomalous port* based on the impact of the problem to minimize performance loss. (3) If an RNIC goes down or drops packets anomalously, isolate the anomalous RNIC in the service application without restarting the training task.

Congestion alleviation and link isolation. During the deployment of R-Pingmesh, we found that more effort is needed to optimize network performance. As shown in Figure 13, of all the network congestion detected by R-Pingmesh, two causes are the most common. The first is *ToR switch downlink congestion* caused by *many-to-one incast*, which should be alleviated by tuning parameters in congestion control algorithms or designing better congestion control algorithms. The second is *ToR switch uplink congestion* caused by *ECMP hash collision*, which can be avoided or mitigated by designing better load balancing algorithms. Furthermore, in a public cloud, in addition to isolating the servers of different tenants, their network links also need to be isolated to prevent traffic contention between different tenants.

8 RELATED WORK

Traditional network monitoring and diagnosis. There is a large body of literature on monitoring and diagnosing physical networks [1, 3, 8, 11, 13–16, 26, 31, 39, 41, 43, 46, 48, 49, 52, 53, 56]. With the growth of cloud services in recent years, there is also a need to monitor and diagnose virtual networks, and some literature works on this [7, 42, 54]. However, these solutions do not focus on monitoring RoCE packets or use RoCE packets for probing, and therefore cannot detect all RoCE network problems.

RDMA network monitoring and diagnosis. RDMA Pingmesh [12] is the first monitoring and diagnostic system proposed for RDMA networks, but apart from using RDMA packets for probing, there is no subsequent exploration and experience. R-Pingmesh systematically illustrates how to build an efficient and stable RoCE network monitoring and diagnostic system based on active probing, including many critical design details and experiences, such as the choice of QP type, how to accurately measure network latency, how to deal with probe noise, and how to probe service networks. Estat [2] tracks the state of the service network by recording key transmission timestamps in RDMA operations. R-Pingmesh achieves the same target by actively probing the service network. X-RDMA [30] pings all machines in the ToR layer to reflect network states. Pangu [10] detects and locates RDMA network problems by tracking NACK events.

Diagnosing RNICs and RDMA servers. End hosts and RNICs have become potential bottlenecks in RDMA communication. Some literature is devoted to finding, understanding, and diagnosing problems in RNICs and RDMA servers. Collie [25] helps RDMA operators and developers uncover performance anomalies in RNICs. Husky [24] is a test suite for systematically evaluating RNIC performance isolation solutions. Lumina [50] tests the correctness and performance of RDMA hardware network stacks. Hostping [29] exploits the feature of RNIC loopback traffic to detect and diagnose intra-host network bottlenecks.

9 CONCLUSION

In this paper, we propose R-Pingmesh, the first service-aware RoCE network monitoring and diagnostic system based on end-to-end active probing. It is based on end-host and commodity RNICs, and can therefore be easily deployed in data center RoCE clusters with low overhead. R-Pingmesh helps to accurately detect and locate failures and performance bottlenecks in both networks and end hosts. R-Pingmesh can also quickly determine if a problem is network-related and assess its impact on services. We have deployed R-Pingmesh on tens of thousands of RNICs and it has become the most important monitoring and diagnostic system for our RoCE networks. *This work does not raise any ethical issues.*

ACKNOWLEDGMENTS

We are grateful to our shepherd, Gianni Antichi, and the anonymous reviewers who helped us improve the quality of this paper. This work is partly supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62132022 and 62372053 and the Natural Science Foundation of Shandong Province under Grant No. ZR2023LZH011.

REFERENCES

[1] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 419–435.

[2] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. 2023. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 49–67.

[3] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. 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*. 662–680.

[4] Ítalo Cunha, Renata Teixeira, Nick Feamster, and Christophe Diot. 2009. Measurement Methods for Fast and Accurate Blackhole Identification with Binary Tomography. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 254–266.

[5] Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng, Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, et al. 2020. EFLOPS: Algorithm and System Co-Design for a High Performance Distributed Training Platform. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 610–622.

[6] Nick Duffield. 2006. Network Tomography of Binary Network Performance Characteristics. *IEEE Transactions on Information Theory* 52, 12 (2006), 5373–5388.

[7] Chongrong Fang, Haoyu Liu, Mao Miao, Jie Ye, Lei Wang, Wansheng Zhang, Daxiang Kang, Biao Lyv, Peng Cheng, and Jiming Chen. 2020. VTrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network. 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*. 31–43.

[8] Seyed K Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient network reachability analysis using a succinct control plane representation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 217–232.

[9] Hao Gao and Nikolai Sakharnykh. 2021. Scaling Joins to a Thousand GPUs.. In *ADMS@ VLDB*. 55–64.

[10] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When Cloud Storage Meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 519–533.

[11] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A simple and scalable method for sensing, inference and measurement in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 549–564.

[12] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.

[13] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, and Hua and Chen. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. *Computer communication review* 45, 4 (2015), 139–152.

[14] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 357–371.

[15] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 71–85.

[16] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. 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*. 404–421.

[17] Yiyi Huang, Nick Feamster, and Renata Teixeira. 2008. Practical Issues with Using Network Tomography for Fault Diagnosis. *ACM SIGCOMM Computer Communication Review* 38, 5 (2008), 53–58.

[18] Sylvain Jeaugey. 2017. NCCL 2.0. In *GPU Technology Conference (GTC)*.

[19] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 463–479.

[20] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. {MegaScale}: Scaling Large Language Model Training to More Than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[21] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 1–16.

[22] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2014. Using RDMA Efficiently for Key-Value Services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 295–306.

[23] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 185–201.

[24] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R Lebeck, and Danyang Zhuo. 2023. Understanding RDMA microarchitecture resources for performance isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 31–48.

[25] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding Performance Anomalies in RDMA Subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 287–305.

[26] Jonatan Langlet, Ran Ben Basat, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. 2023. Direct Telemetry Access. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 832–849.

[27] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*. 583–598.

[28] Linux. 2024. rdma-core. https://github.com/linux-rdma/rdma-core.

[29] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing intra-host network bottlenecks in RDMA servers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 15–29.

[30] Teng Ma, Tao Ma, Zhuo Song, Jingxuan Li, Huaixin Chang, Kang Chen, Hai Jiang, and Yongwei Wu. 2019. X-rdma: Effective rdma middleware in large-scale production environments. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–12.

[31] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 129–143.

[32] NVIDIA. 2022. Nvidia DGX A100. https://images.nvidia.com/aem-dam/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf.

[33] NVIDIA. 2023. Doubling all2all Performance with NVIDIA Collective Communication Library 2.12. https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/.

[34] NVIDIA. 2023. NVIDIA DGX SuperPOD: Next Generation Scalable Infrastructure for AI Leadership. https://docs.nvidia.com/https:/docs.nvidia.com/dgx-superpod-reference-architecture-dgx-h100.pdf.

[35] NVIDIA. 2023. NVIDIA InfiniBand Adaptive Routing Technology—Accelerating HPC and AI Applications. https://resources.nvidia.com/en-us-cloud-native-supercomputing-dpus-campaign/infiniband-white-paper-adaptive-routing.

[36] NVIDIA. 2024. NCCL. https://github.com/NVIDIA/nccl.

[37] NVIDIA. 2024. nccl-test. https://github.com/NVIDIA/nccl-tests.

[38] OFED. 2024. perftest. https://github.com/linux-rdma/perftest.

[39] Yanghua Peng, Ji Yang, Chuan Wu, Chuanxiong Guo, Chengchen Hu, and Zongpeng Li. 2017. deTector: a Topology-aware Monitoring System for Data Center Networks. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 55–68.

[40] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 16–29.

[41] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. 2014. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 407–418.

[42] Arjun Roy, Deepak Bansal, David Brumley, Harish Kumar Chandrappa, Parag Sharma, Rishabh Tewari, Behnaz Arzani, and Alex C Snoeren. 2018. Cloud datacenter sdn monitoring: Experiences and challenges. In *Proceedings of the Internet Measurement Conference 2018*. 464–470.

[43] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 599–614.

[44] Xinchen Wan, Hong Zhang, Hao Wang, Shuihai Hu, Junxue Zhang, and Kai Chen. 2020. Rat-Resilient Allreduce Tree for Distributed Machine Learning. In *4th Asia-Pacific Workshop on Networking*. 52–57.

[45] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2023. Optimized Network Architectures for Large Language Model Training with Billions of Parameters. *arXiv preprint arXiv:2307.12169* (2023).

[46] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and TS Eugene Ng. 2022. Closed-loop network performance monitoring and diagnosis with SpiderMon. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 267–285.

[47] Jilong Xue, Youshan Miao, Cheng Chen, Ming Wu, Lintao Zhang, and Lidong Zhou. 2019. Fast Distributed Deep Learning over RDMA. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–14.

[48] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[49] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. 2019. dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 207–220.

[50] Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, and Xin Jin. 2023. Understanding the micro-behaviors of hardware offloaded network stacks with lumina. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 1074–1087.

[51] Hongyi Zeng, Ratul Mahajan, Nick McKeown, George Varghese, Lihua Yuan, and Ming Zhang. 2015. Measuring and troubleshooting large operational multipath networks with gray box testing. *Mountain Safety Res., Seattle, WA, USA, Rep. MSR-TR-2015-55* (2015).

[52] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. 2021. LightGuardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 991–1010.

[53] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. Flow event telemetry on programmable data plane. 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*. 76–89.

[54] Shunmin Zhu, Jianyuan Lu, Biao Lyu, Tian Pan, Chenhao Jia, Xin Cheng, Daxiang Kang, Yilong Lv, Fukun Yang, Xiaobo Xue, et al. 2022. Zoonet: a proactive telemetry system for large-scale cloud networks. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 321–336.

[55] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.

[56] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.