Hostmesh: Monitor and Diagnose Networks in Rail-optimized RoCE Clusters

Kefei Liu[1], Jiao Zhang[1,2,*], Zhuo Jiang[3], Xuan Zhang[1], Shixian Guo[3], Yangyang Bai[3], Yongbin Dong[3]
Zhang Zhang[3], Xiang Shi[3], Lei Wang[3], Haoran Wei[3], Zicheng Wang[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 failures and bottlenecks, which become more common as the RoCE network scales. To effectively detect and locate these problems independent of service traffic, RoCE networks require a monitoring and diagnostic system based on active probing. However, existing active probing schemes typically rely on a controller to design the probing plan for each server, which is difficult to deploy and has high synchronization overhead in *multi-tenant clusters*. Fortunately, *rail-optimized clusters* have become more common in recent years to improve network performance. In these clusters, the controller is unnecessary.

In this paper, we propose Hostmesh, the first network monitoring and diagnostic system for rail-optimized RoCE clusters based solely on full-mesh probing between RDMA NICs on the same host. Hostmesh uses the feature of rail-optimized networks and does not rely on a controller to generate pinglists. We deployed Hostmesh for over three months on a multi-tenant rail-optimized RoCE cluster with hundreds of servers. During the deployment, Hostmesh effectively detected and located 8 types of problems caused by hardware failures, misconfigurations, network congestion, and intra-host bottlenecks. 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, Rail-optimized networks, Network troubleshooting

ACM Reference Format:
Kefei Liu[1], Jiao Zhang[1,2,*], Zhuo Jiang[3], Xuan Zhang[1], Shixian Guo[3], Yangyang Bai[3], Yongbin Dong[3], Zhang Zhang[3], Xiang Shi[3], Lei Wang[3], Haoran Wei[3], Zicheng Wang[3], Yongchen Pan[1], Tian Pan[1,2], Tao Huang[1,2]. 2024. Hostmesh: Monitor and Diagnose Networks in Rail-optimized RoCE Clusters. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 3–4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663426

*Jiao Zhang is the corresponding author.

APNet 2024, August 3–4, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08
https://doi.org/10.1145/3663408.3663426

1 INTRODUCTION

Stable and high-quality operation of RoCE (RDMA over Converged Ethernet) services requires quick detection and accurate localization of network problems. Some services, such as distributed machine learning (DML), are sensitive to *single-point failures*. Packet drops caused by a single-point failure, such as a flapping RNIC or switch link, can significantly degrade overall training throughput and, in severe cases, cause the task to fail. In addition to single-point failures, *performance bottlenecks* that occur in end hosts, such as CPU overload, or in inter-host networks, such as network congestion due to uneven load balancing, can also degrade service performance. As the size of the RoCE cluster grows, both the frequency and impact of these network problems increase.

RoCE networks require an effective monitoring and diagnostic system to quickly detect and locate RoCE network failures and performance bottlenecks. To ensure that these problems can be detected independent of service traffic, this system should be based on *active probing*. However, existing active probing schemes [7, 12, 30, 33, 41] typically *rely on a controller to design the probing plan (pinglist) for each server*, which is difficult to deploy and has high synchronization overhead in *multi-tenant clusters*.

We wish to design an *active probing system for RoCE clusters without relying on a controller* to solve the above limitations. Fortunately, the use of *rail-optimized networking* [26, 28] has become more common in recent years to minimize cross-ToR (Top-of-Rack switch) traffic and flow interference caused by hash collisions. In a rail-optimized cluster, *each host has multiple NICs, and traffic between different NICs on the same host must go through the top-tier switches in the cluster.* In this topology, the controller is unnecessary.

In this paper, we propose Hostmesh, the first network monitoring and diagnostic system for rail-optimized RoCE clusters *based solely on full-mesh probing between RNICs on the same host.* Hostmesh leverages the topology feature of rail-optimized networks and does not rely on a controller to generate pinglists. By modifying the probe packet's 5-tuple, Hostmesh can *cover all RNICs, switches, and links in a rail-optimized cluster*. Since the target RNIC is on the same host, Hostmesh can measure end-host processing delay, *one-way network latency*, and detect *one-way packet drops* with low overhead. With these metrics, Hostmesh can effectively detect and locate RoCE network failures and performance bottlenecks. In addition, since there is no need to respond with ACKs, there is *no probe timeout caused by high processing delays*.

We deployed Hostmesh for over three months on a multi-tenant, rail-optimized DML RoCE cluster with hundreds of servers. During the deployment, Hostmesh effectively detected and located 8 types of problems caused by hardware failures, misconfigurations,

network congestion, and intra-host bottlenecks. And we share our experience in dealing with them. In summary, this paper makes the following contributions:

- We propose the first network monitoring and diagnostic system for rail-optimized RoCE clusters based solely on full-mesh probing between RNICs on the same host.
- We propose an innovative method to accurately measure one-way network latency and detect one-way packet drops based on commodity RNICs with low overhead.
- We summarize the problems that Hostmesh has detected and located and share our experience in dealing with them.

2 BACKGROUND & MOTIVATION

2.1 Network Problems in RoCE Clusters

2.1.1 Frequent Packet Drops Caused by Network Failures.
In theory, lossless RoCE networks do not drop packets due to congestion. However, we find that packet drops occur frequently due to network failures. These failures include (1) *anomalous RNICs*, such as RNIC flapping[1], and (2) *anomalous network devices*, such as link flapping, damaged or aged fiber, and PFC misconfigurations.
Some services are sensitive to single-point failures. Of all the services served by RoCE networks, distributed machine learning (DML) is the most vulnerable to network problems. In DML, all participating GPUs periodically synchronize their local gradients over the network [5, 17, 34]. The completion time of this process is determined by the slowest GPU. Since RDMA throughput is vulnerable to packet drops, either end-host or network packet drops can cause severe throughput degradation of multiple flows, which can further degrade the average throughput of the DML cluster.

2.1.2 Frequent Performance Bottlenecks.
In addition to network failures, *performance bottlenecks* can also severely 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. As cluster size grows, both the frequency and impact of performance bottlenecks increase. Next, we present two performance bottlenecks that occur in our DML 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 grows dramatically, increasing the probability of hash collisions. When this happens, all flows on the bottleneck link suffer throughput degradation, further slowing the throughput of the entire cluster.
Case 2: CPU overload. Before training begins, each participating host in a training task needs to load training models from the remote storage servers. 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.

[1]The state of a flapping RNIC frequently switches between up and down.

2.2 Limitations of Existing Active Probing Mechanisms

We need *an efficient network monitoring and diagnostic system* to quickly detect and locate RoCE network failures and performance bottlenecks. To ensure that these problems can be detected independent of service traffic, this system should be based on *active probing*. Existing active probing schemes typically *rely on a controller to design the probing plan for each server*. However, in *multi-tenant clusters*, the reliance on controllers has the following limitations.
Deploying a controller in multi-tenant clusters is difficult. In multi-tenant clusters, the network between different tenants is isolated. In this scenario, a centralized controller that can communicate with all tenant hosts might not be accepted by some tenants due to security concerns. Another option is to select a server within each tenant cluster as the controller. However, this consumes additional resources from the tenant host, such as CPU and memory. In addition, the number of hosts in a tenant cluster can grow or shrink at any time, making it difficult to ensure stable operation of the controller. As a result, *reliance on controllers increases the difficulty of deploying the system in multi-tenant clusters.*
Controllers in multi-tenant clusters have high synchronization overhead. In multi-tenant clusters, a server might be removed from or added to a tenant cluster frequently. If a server is removed from a tenant cluster and other servers in that cluster are still probing it, those probe packets will be dropped. This appears to be a network problem, and using these dropped probe packets to troubleshoot networks can lead to incorrect conclusions. Therefore, the controller deployed in multi-tenant clusters must maintain the latest server information of each tenant cluster and frequently update the pinglist for each server to avoid probing noise, which introduces high synchronization overhead.

2.3 Opportunity: Rail-optimized Network

We wish to design an *active probing system for RoCE clusters without relying on a controller* to solve the above limitations. However, it is a fact that in some topologies, such as clusters with only one RNIC on each server, we need to perform inter-server probing to cover all network devices and links in a cluster. For these clusters, a controller is essential for designing probing plans and providing the communication address of the target servers. Fortunately, *the use of rail-optimized topologies in RoCE clusters has become more common in recent years to achieve better network performance*, and these clusters do not necessarily require a controller.

2.3.1 Rail-optimized Topology.
As host computing power grows, both the number and line rate of NICs on a host increase in order to achieve sufficient communication capability. For hosts with multiple NICs, *rail-optimized networking* provides better network performance. In traditional Clos networks, all NICs on a host are connected to the same ToR switch. In contrast, in rail-optimized networks, NICs on a host are connected to different ToR switches (rail switches), and NICs with the same index on different hosts form a rail. Figure 1 shows a small-scale two-tire rail-optimized cluster where all rail switches are connected to all spine switches in a full-bisection fashion. In this topology, intra-rail communication is below the rail switch, while

inter-rail communication must pass through the spine switches. For hosts with multiple NICs, the rail-optimized topology allows more hosts to be under the same ToR switch compared to the Clos topology, minimizing cross-ToR traffic and hash collisions.

A key insight is that in a cluster with the network topology shown in Figure 1, ***each host has multiple NICs, and traffic between different NICs of the same host must go through top-tier switches in the cluster***. Therefore, in such a cluster, we can use different NICs on each host to probe each other, and keep changing the probe's 5-tuple[2]. Because ECMP hashes flows to parallel paths based on the flow 5-tuple, these probe packets can cover all NICs, switches, and network links in the cluster. Because the target NIC is on the local host, there is no need to rely on a controller to generate pinglists and provide target addresses.

2.3.2 *Scalability of Rail-optimized Clusters.*

The two-tier rail-optimized cluster shown in Figure 1 has only one group of rail switches and servers connected to them, and the number of servers this cluster can accommodate is limited by *the number of downstream ports on the rail switch*. To accommodate more servers, a cluster can have multiple parallel groups of rail switches and servers, and connect all rail switches and spine switches in a full-bisection fashion. However, such a two-tier rail-optimized cluster cannot scale infinitely because *the spine switch has a limited number of ports* and can only connect to a limited number of rail switches.

Further scaling requires more tires of switches, and there are two ways to do this. The first is to *add more tires of rail switches below the spine switches to form a larger rail-optimized cluster*. In this cluster, the lower-tier rail switches connect to the higher-tier rail switches with the same index to aggregate the network, and the highest-tier rail switches connect to the spine switches. As a result, intra-rail traffic is still below the rail switches (up to the highest rail switch at most), while inter-rail traffic goes up to the top-tier switches in the cluster. Therefore, probe packets between different NICs on the same host can still effectively cover all networks in this cluster. The other option is to *add switches above the spine switches to interconnect multiple two-tier rail-optimized clusters*. In this topology, probe packets between different NICs on a host only go up to the spine switches and cannot reach the network above the spine switches. As a result, these probe packets cannot cover inter-cluster networks. However, these packets can still effectively probe networks within each cluster.

2.3.3 *Benefits of Probing Between RNICs on the Same Host.*

In addition to not relying on a controller, probing between RNICs on the same host has the following two advantages over probing between servers.

Get one-way latency and detect one-way packet drops with low overhead. For mechanisms that probe between servers, the probing method is typically that (1) the prober sends a probe to the responder; (2) after receiving the probe, the responder replies with an ACK; and (3) based on the probe send time and the ACK receive time, the prober calculates the network RTT. In this way, if the network RTT increases abnormally, or the ACK packet does not return, the problem may be in either the path traversed by the probe packet or the path traversed by the ACK packet. *If we could measure one-way network latency or detect one-way packet drops, we could locate network problems more accurately.* When probing between servers, this has a high overhead because it requires accurate synchronization of clocks and information between servers. In contrast, when probing between RNICs on the same host, this can easily be done without synchronization overhead.

No probing noise caused by high processing delay in the responder server. In inter-server probing, if the responder server has a high CPU load, it will spend more processing time responding with ACKs. In some severe cases, the responder's processing delay can exceed the timeout threshold, resulting in a probe timeout in the prober. This timeout is misleading because *it has the same phenomenon as packet drops, and using it to troubleshoot network problems can lead to incorrect conclusions.* In contrast, when probing between RNICs on the same host, there is no need to send ACKs, so there is no noise caused by high processing delay.

[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.

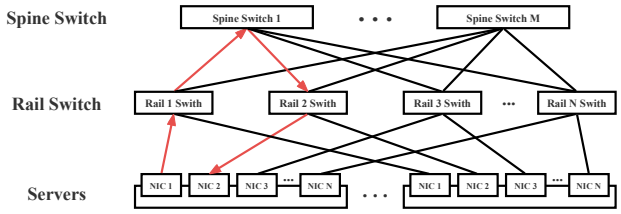

Figure 1: A two-tier rail-optimized cluster. Each server has multiple RNICs connected to different rail switches. Traffic between RNICs on the same host traverses the top-tier switches in the cluster. The red lines with arrows show such an example.

3 SYSTEM DESIGN

3.1 Design Challenges

To accurately locate RoCE network failures and performance bottlenecks, we should address the following challenges.

Accurately measure processing delay and one-way network latency with commodity RNICs. Performance bottlenecks can be detected by latency metrics such as *network latency* for network congestion and *end-host processing delay* for CPU overload. However, it is not enough to measure network latency at the application layer. If that latency increases, it's difficult to tell whether the cause is CPU overload or network congestion. We need to accurately measure processing delay and network latency to accurately locate bottlenecks in the end hosts and networks, respectively. In addition, the measurement should be based on commodity RNICs to ensure deployability. We address this challenge in Section 3.3.1.

Distinguish between host and network drops. Packet drops can be caused by end hosts or switches. Identifying the source of packet drops is a prerequisite to accurately locating anomalous devices. For example, a failed switch link drops some probes. At the same time, several anomalous RNICs are also dropping probes. Using all of these packet drops to locate the anomalous switches may lead to incorrect conclusions. Therefore, we must first decide whether the end hosts or the switches are the source of the packet drops. We address this challenge in Section 3.4.1.

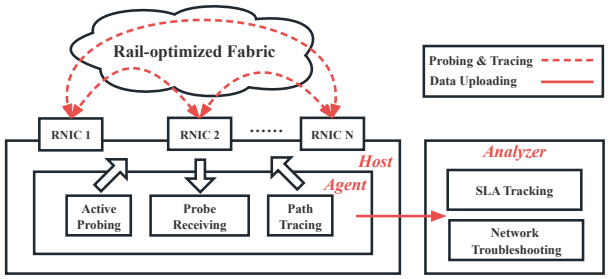

Figure 2: Hostmesh framework. The core idea of Hostmesh is to perform full-mesh probing between RNICs on the same host.

3.2 Hostmesh Framework

Figure 2 shows the Hostmesh framework. Hostmesh is a network monitoring and diagnostic system for rail-optimized RoCE clusters based on active probing between RNICs on the same host. It exploits the fact that in a rail-optimized cluster, *each host has multiple NICs, and traffic between different NICs on the same host must go through the top-tier switches in the cluster*. Hostmesh has two modules, *Agent* and *Analyzer*, and does not rely on a controller to generate pinglists.

Agent performs full-mesh probing between different RNICs on the same host to measure end-host processing delay, one-way network latency, and to detect one-way packet drops. As ECMP hashes packets to parallel paths according to their 5-tuples, each Agent continuously changes the probe packet's source port to achieve full coverage of the cluster network, including all RNICs, switches, and network links. Therefore, Hostmesh can effectively detect all possible network problems in a rail-optimized RoCE cluster. In addition, Agent continuously tracks the paths of probes using `Traceroute`.

Analyzer collects probe results and probe paths from all Agents and is responsible for SLA tracking and network troubleshooting. Next, we describe the design and implementation details of Agent and Analyzer, respectively.

3.3 Hostmesh Agent

3.3.1 *Measure Processing Delay and One-way Latency.*

We need first select the QP type to use for probing, which should be able to *accurately measure one-way network latency and end-host processing delay with low overhead*. First, as shown in Figure 3, accurate measurement of network latency requires two timestamps: ② sender RNIC sends the probe and ③ receiver RNIC receives the probe. However, instead of providing timestamps on sending and receiving packets, *commodity RNICs only provide timestamps when generating CQEs.*

For all three QP types (RC, UC, and UD), a CQE with an RNIC timestamp can be generated when an RDMA message is successfully received (③). In RC, however, the sender RNIC only generates a CQE when it receives the ACKs of all the message packets it has sent to the destination. Therefore, we cannot get ② in RC. In UC and UD, the sender RNIC does not guarantee data reliability, and it generates a CQE immediately after the message is sent. With UC or UD QPs, Agent can accurately obtain both the send and receive timestamps of probes.

In terms of connection overhead, if Agent uses connection-based QPs (RC or UC), for each target RNIC, the sender RNIC must create a QP and connect it to the destination QP, which consumes the limited QP context (QPC) cache in the RNIC [18–20], increases the probability of QPC cache misses for service traffic, and will lead to RNIC performance degradation. In contrast, UD is connectionless. With UD, Agent only needs to create one QP on each RNIC. Therefore, we choose UD, which can accurately measure network latency and processing delay with the least connection overhead.

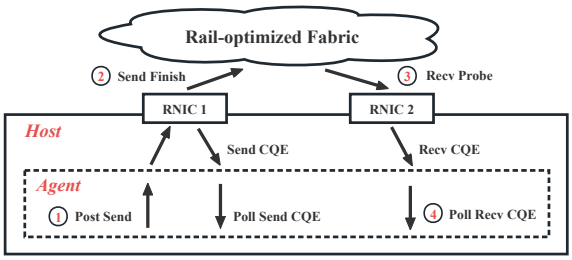

Figure 3: Probing end-host processing delay and one-way network latency with UD QPs. The timestamps to be measured are marked with circles.

The probing process for processing delay and network latency is shown in Figure 3. Each RNIC acts as both a sender and a receiver. For each RNIC, Agent periodically uses it to ping a randomly selected other RNIC on the local host. Each Agent maintains a source port pool, and for each probe, Agent randomly selects a source port for it from this pool to ensure complete coverage of network links. In addition, Agent periodically changes the source port in the pool to detect problems that can only be triggered by certain 5-tuples, such as silent packet drops for certain 5-tuples. Agent uses ③-② to obtain the one-way network latency. And by measuring ① and ④ at the application layer, Agent can obtain the end-host processing delay by (④-①)-(③-②) without clock synchronization between the host CPU and RNICs. If the receiving RNIC does not receive the probe in a certain period of time, the probe may be dropped on this *one-way path*, and this probe is marked as a timeout.

3.3.2 *Path Tracing for Probe Packets.*

The path tracing module is responsible for tracing the paths of the probes so that Analyzer can locate the network problem. This raises a design discussion: *should we trace the probes continuously or only when they encounter a network problem?* We choose the former, and for each probe 5-tuple, Agent periodically `Traceroute` its latest path. Our reasoning is twofold. First, in the case of a persistent failure, such as a down link, 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 ICMP replies. Therefore, we limit the frequency of `Traceroute` in Agent to avoid overloading the switches.

3.4 Hostmesh Analyzer

3.4.1 *Detect Anomalous RNICs in Real Time.*

Among all the inter-host network probe results uploaded by Agents, anomalies can be categorized as *timeout* and *high network latency*. And their causes can be attributed to *RNIC problems*[3] or *switch*

[3]It is difficult to directly determine whether the problem is in the RNIC, the RNIC link, or the switch port connected to the RNIC by probing alone. Therefore, Analyzer considers all these problems as *RNIC problems* and other network problems as *switch network problems.*

Algorithm 1 Identify the Most Suspicious Switch Links

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

network problems. If we can detect anomalous RNICs, we can further filter out anomalous probes caused by these RNICs and accurately locate switch network problems.

We use a simple but effective method to monitor RNIC status. *If a large number of probes to an RNIC show anomalies at the same time, that RNIC is probably abnormal.* Meanwhile, any anomalous probes 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 over a fine-grained period of time* so that we can monitor the status of each RNIC in real time. Since Hostmesh continuously performs full-mesh probing between different RNICs on the host, this requirement is easily met.

3.4.2 *Locate Switch Network Problems.*

Based on 3.4.1, we can obtain anomalous probes caused only by switch network problems. As shown in Algorithm 1, with the paths of these probes, we can use a simple voting mechanism to determine abnormal links[4]. The core idea is to find common links of anomalous paths [4, 6, 16], 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 one by one and counts the number of times each link is traversed. The links with the highest number of votes are the most suspicious.

4 IMPLEMENTATION

Agent is implemented based on the verbs API [22]. It launches two independent threads on each RNIC, responsible for *sending probes* and *receiving probes*. The probe interval for each sending thread is 100 ms, and the timeout for each probe is 500 ms. Changing the source port in the probe packet is done by setting the *flow label* in the verbs API. The probe packet payload is 50 bytes, including some required fields. Small packets reduce bandwidth overhead and allow for accurate latency measurements.

Analyzer uses 20s granularity to detect and locate problems in real time. In each analysis, it processes all data uploaded by Agents in the last 20s. It first assesses the status of each RNIC. If more than 10% of the probes to an RNIC experience a timeout, that RNIC is considered anomalous. During this analysis and for the next minute, Analyzer considers all timeout probes sent to or from that RNIC

[4]By simply replacing all *link* in Algorithm 1 with *switch*, Hostmesh can identify the most suspicious switch.

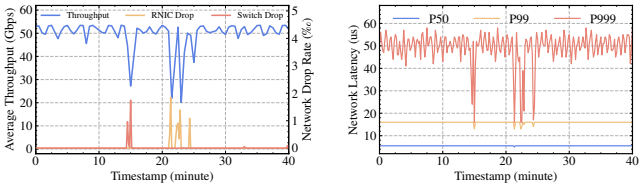

Figure 4: (left) Hostmesh detects and categorizes packet drops in real time. These drops degrade the training throughput. (right) The throughput degradation further leads to degraded network latency due to less congestion, indicating that *Hostmesh can accurately measure network latency and use it to reflect network congestion.*

as *RNIC problems*. After this step, all remaining timeout probes are considered *switch network problems*. Next, Analyzer aggregates all latency and timeout data in this analysis period to determine the (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 latency* for the RoCE cluster. In this way, Analyzer tracks SLAs for cluster networks.

5 EVALUATION & PROBLEMS FOUND

We deployed Hostmesh for over three months on a multi-tenant rail-optimized DML RoCE cluster with hundreds of servers. During the deployment, Hostmesh effectively detected and located 8 types of problems. In this section, we evaluate Hostmesh's performance, summarize these problems by root cause, and share our experience in dealing with them.

Cause 1: Hardware failures. Hardware failures make up the majority of problems in RoCE networks. Of all the hardware failures, the most common problems are (#1) *RNIC or switch link flapping* and (#2) *packet corruption on RNICs or switches*. These problems cause packet drops and degraded training throughput. In server cases, training tasks will fail. As shown in Figure 4, Hostmesh can detect packet drops in real time and determine whether RNICs or switches are to blame. In addition, Hostmesh can further locate switch network problems accurately by using Algorithm 1.

When flapping occurs (#1), the status of the RNIC or switch port frequently changes between up and down. This used to be a very common problem in our data center. After a long-term analysis with our RNIC vendor, we realized that the root cause was a mismatch between the RNIC/switch and the cable. After replacing the cable, the flapping frequency decreased significantly. Packet corruption (#2) is usually caused by damaged or aged fibers or dust in the optical module. When these failures occur, if an RDMA connection retransmits unsuccessfully more than a certain number of times, the connection will be broken, causing training tasks to fail. To ensure stable operation, our services set the retransmission count to the maximum (7 times for RDMA) and set the retransmission timeout to a higher value to avoid consuming all retransmission counts during hardware failures.

Cause 2: Misconfigurations. Through continuous probing, Hostmesh detected that some RNICs were RDMA unreachable due to (#3) *lack of RNIC routing configurationsr* and (#4) *improper switch Access Control List (ACL) configurations* before running any training tasks, preventing service failures due to the use of these anomalous RNICs. In addition, during training tasks, Hostmesh found some packet drops on switches caused by congestion, resulting in severe training throughput degradation. The root cause is that (#5) *these switches*

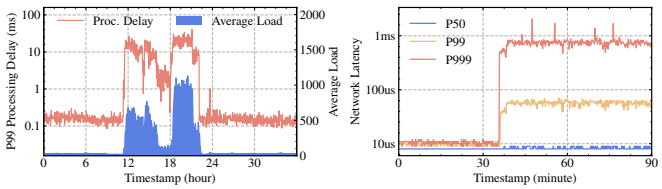

Figure 5: (left) CPU overload results in high processing delay on a host. *The measured end-host processing delay can accurately reflect CPU utilization.* (right) A PFC storm results in high tail network latency in the cluster.

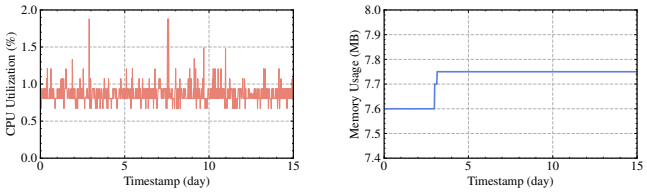

Figure 6: CPU and memory usage of Hostmesh Agent on a host with 8 RNICs.

are not PFC enabled or do not have proper PFC headroom configurations. Hostmesh helps operators effectively monitor and verify the correctness of RoCE network configurations.

Cause 3: Network congestion. As mentioned in Section 2.1.2, (#6) network congestion caused by uneven load balancing is common in large clusters. In rail-optimized networks, *most hash collisions occur on the uplink of rail switches.* In these cases, Hostmesh can detect probes suffering from high network latency and use their paths to further locate the hotspot.

To minimize load imbalance, for DML services that use NCCL [29] for network communication, we first enable PXN (PCI × NVLink) [26, 27], which makes better use of the rail-optimized topology and transfers messages under rail switches as much as possible to reduce the probability of hash collisions in inter-rail communication. We also set the `QPS_PER_CONNECTION` [27] parameter to a larger value, which creates more connections between RNICs. This way, the messages that need to be sent are spread evenly across more connections. Because more connections take more parallel paths, network bandwidth can be better utilized.

Cause 4: Intra-host bottlenecks. During deployment, Hostmesh detected two types of intra-host bottlenecks. The first is the (#7) CPU overload mentioned in Section 2.1.2. As shown in Figure 5 (left), Hostmesh can easily diagnose this problem by measuring the end-host processing delay. In addition, as shown in Figure 5 (right), Hostmesh found several cases where the cluster was suffering from abnormally high tail network latency. Upon further inspection, we found that some RNICs were sending a large number of PFC pause frames, resulting in a *PFC storm* [11]. The root cause is (#8) speed or width downgrades for RNIC or GPU PCIe links, resulting in degraded intra-host bandwidth from the RNIC to the GPU [23]. PCIe downgrades can be caused by loose PCIe interfaces, dust on the PCIe interface, or hardware failures.

Agent deployment overhead. In Agent, CPU is mainly consumed by sending/receiving probes and ACKs, and CPU consumption scales linearly with the number of RNICs on the host. For each RNIC, the probe frequency is about 10 packets per second, which is not CPU intensive. We monitored the Agent process on hosts with 8 RNICs and measured its CPU and memory consumption for half a month. As shown in Figure 6, the average CPU and memory consumption of Agent is about 0.8% and 7.7 MB, respectively, which is low overhead. As for bandwidth overhead, the average bandwidth consumption for each RNIC is less than 20 Kbps, which is negligible for mainstream 100/200 Gb/s RNICs in the data center.

Discussion. First, in addition to RoCE clusters, *the idea of probing between NICs on the same host can be applied to any cluster with a rail-optimized topology*, such as TCP rail-optimized clusters. Second, in this paper, Hostmesh uses CPU memory for RDMA probing. In training clusters, another option is to *use GPU memory for RDMA probing*, which is closer to the RDMA usage of training services and can additionally detect GPU-related problems such as GPU down. However, this approach consumes valuable GPU memory and may degrade service performance. Finally, as mentioned in Section 2.3.2, while Hostmesh can effectively cover networks within rail-optimized clusters, it cannot cover inter-cluster networks. For inter-cluster networks, inter-server probing is still necessary.

6 RELATED WORK

Traditional network monitoring and diagnosis. There is a large body of literature that focuses on monitoring and diagnosis for both physical networks [1, 3, 8, 10, 12–15, 21, 25, 30, 31, 33, 35–37, 39, 40, 42] and virtual networks [7, 32, 41]. However, the active probing schemes among them typically rely on a controller to design the probing plan, which is difficult to deploy and has high synchronization overhead in multi-tenant clusters.

RDMA network monitoring and diagnosis. RDMA Pingmesh [11] is the first diagnostic tool proposed for RDMA networks, but other than using RDMA packets for probing, there is no subsequent exploration and experience. Estat [2] tracks the status of the RDMA network by recording key transmission timestamps in RDMA operations. X-RDMA [24] integrates full-mesh ping at the ToR level to reflect network status. Pangu [9] detects and locates network problems by tracking NACK events in lossless networks. Lumina [38] detects correctness and performance issues in RDMA network stacks. Hostping [23] exploits the feature of RNIC loopback traffic to monitor and diagnose network bottlenecks within RDMA servers. Hostmesh targets rail-optimized RoCE clusters and optimizes system design for network monitoring and diagnosis.

7 CONCLUSION

Rail-optimized networking has become more common to provide better network performance. For rail-optimized RoCE clusters, we rethink the design of the network monitoring and diagnostic system. Hostmesh only performs probing between RNICs on the same host and does not rely on a controller to design pinglists. This makes Hostmesh easier to deploy in any rail-optimized RoCE cluster, especially multi-tenant clusters. In addition, this probing method allows for finer-grained network measurements and reduces probing noise, so that Hostmesh can locate network problems more accurately. *This work does not raise any ethical issues.*

ACKNOWLEDGMENTS

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

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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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*. 185–201.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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/.

[27] NVIDIA. 2023. NCCL Environment Variables. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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*. 991–1010.

[40] 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.

[41] 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.

[42] 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.