Hawkeye: Diagnosing RDMA Network Performance Anomalies with PFC Provenance

Shicheng Wang[1], Menghao Zhang[2], Xiao Li[3], Qiyang Peng[2], Haoyuan Yu[3], Zhiliang Wang[1], Mingwei Xu[1], Xiaohe Hu[4], Jiahai Yang[1,5], Xingang Shi[1]
[1]Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University
[2]School of Software, Beihang University
[3]Department of Computer Science and Technology, Tsinghua University
[4]Infrawaves [5]Quan Cheng Laboratory

Abstract

RDMA is becoming increasingly prevalent from private data centers to public multi-tenant clouds, due to its remarkable performance improvement. However, its lossless traffic control, i.e., PFC, introduces new complexities in network performance anomalies (NPAs) due to its cascading congestion spreading property, which usually incurs complaints from customers/applications about certain flows' performance degradation. Existing studies fall short in fine-grained visibility of PFC impact and traceability of PFC causality, and are thus ineffective in diagnosing the root causes for RDMA NPAs. In this paper, we propose Hawkeye, an accurate and efficient RDMA NPA diagnosis system based on PFC provenance. Hawkeye comprises 1) a fine-grained PFC-aware telemetry mechanism to record the PFC impact on flows; 2) an in-network PFC causality analysis and tracing mechanism to quickly and efficiently collect causal telemetry for diagnosis; and 3) a provenance-based diagnosis algorithm to comprehensively present the anomaly breakdown, identifying the anomaly type and root causes accurately. Through extensive evaluations on both NS-3 simulations and a Tofino testbed, Hawkeye can quickly and accurately diagnose multiple RDMA NPAs with over 90% precision and 1-4 orders of magnitude lower overhead than baselines.

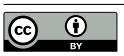
CCS Concepts

• **Networks** → **Programmable networks**; **Network monitoring**; **Data center networks**; **Network performance analysis**.

Keywords

RDMA Networks, Performance diagnosis, Programmable Networks, Network Provenance

ACM Reference Format:
Shicheng Wang, Menghao Zhang, Xiao Li, Qiyang Peng, Haoyuan Yu, Zhiliang Wang, Mingwei Xu, Xiaohe Hu, Jiahai Yang, Xingang Shi. 2025. Hawkeye: Diagnosing RDMA Network Performance Anomalies with PFC Provenance. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3718958.3750490



SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/25/09
https://doi.org/10.1145/3718958.3750490

1 Introduction

Remote Direct Memory Access (RDMA) has become a pivotal technology for network applications to achieve ultra-low latency and ultra-high throughput. With the advent of RoCEv2 (RDMA over Converged Ethernet version 2) [29], RDMA has been increasingly used for many distributed applications in industrial large-scale data centers [6, 17, 20, 56, 80], especially in areas like distributed deep learning training [9, 16, 30, 31, 51, 65, 69] and cloud storage [6, 11, 12, 15, 17, 32, 44, 70]. And currently it is becoming a trend to expand RDMA from private data centers to public multi-tenant clouds [1, 5, 6, 56].

However, RDMA also introduces new complexity for network performance anomalies (NPAs), especially under the trend of deployment in multi-tenant public clouds. In RDMA networks, Priority Flow Control (PFC) [28] is deployed to enable a loss-free network and guarantee high performance. Nevertheless, PFC introduces several performance issues. Once one port becomes congested, PFC spreads the congestion across multiple hops and imposes performance degradation on other flows that may even not traverse the initial congested point. Therefore, various performance anomalies can be raised, such as head-of-line blocking, unfairness, PFC storm and even PFC deadlock [20, 26]. Even with fine-grained congestion control [36, 45, 80] and advanced load balance [52, 59], PFC cannot be fully eliminated and still occurs frequently. Worse yet, these mechanisms can in turn be misled by PFC to make wrong decisions [25, 63]. As a result, when customers/applications complain about certain flows' performance degradation, it is difficult to diagnose the root causes for the RDMA NPAs, especially under increasingly complex traffic in multi-tenant public clouds.

Current network performance anomaly diagnosis mechanisms cannot diagnose RDMA NPAs and the root causes effectively and efficiently. First, fine-grained PFC visibility is lacking in current monitoring and diagnosis systems, making inaccurate congestion identification. Existing NPA diagnosis studies, either on the host side [19, 21, 41, 46] or the switch side [6, 17, 22, 24, 39, 47, 58, 66, 79], lack the monitoring capability for flow-level PFC impact, and thus cannot distinguish PFC-related anomalies from direct flow contention. Second, current monitoring and diagnosis systems fall short in quickly and efficiently tracing PFC causality. Due to the cascading property of PFC, the congestion may not occur within a single queue independently. Therefore, tracing the PFC causality requires collecting the related telemetry on the PFC spreading path efficiently. However, existing approaches either fail to capture the complete set of causal switches [22, 37, 47, 58, 66, 79] or suffer from high overhead [24, 55]. Third, current diagnosis algorithms, which

are mainly based on the flow-interaction paradigm [10, 37, 46, 66], cannot support accurate root cause analysis on PFC anomalies. They attribute the root cause to the flows contending with the victim flow in the shared queue, instead of flows or hosts causing PFC originally. Therefore, both *why* (the root cause flows or hosts beyond the victim path) and *how* (the PFC spreading path) about the anomaly are largely ignored by current diagnosis paradigms.

In this paper, we present Hawkeye, an accurate and efficient RDMA NPA diagnosis system with PFC provenance. Hawkeye detects the victim flow encountering performance anomalies, constructs the complete anomaly provenance and locate the root causes: flows (e.g., problematic bursts) or hosts (e.g., host-side PFC injection), hence providing network operators with comprehensive understanding of the anomalies. The accuracy and efficiency of Hawkeye are derived from the following three key techniques. First, Hawkeye augments telemetry systems by incorporating flow-level PFC impact, delivers fine-grained PFC visibility and enables analysis of PFC-induced effects beyond conventional intra-queue flow interaction. Second, Hawkeye supports accurate and efficient telemetry information collection across the network. By line-rate analyzing the victim flow path and the port-level PFC causality in the data plane, Hawkeye quickly identifies the switches causally relevant to the anomaly, and efficiently collects the telemetry data through the switch CPU asynchronously. Third, Hawkeye constructs a heterogeneous provenance graph describing causality relationship between flows and ports. Hawkeye also proposes a signature-based anomaly diagnosis algorithm to match the behaviors and locate the root cause. We implement a prototype of Hawkeye at both Intel Tofino hardware and NS-3, and make the code publicly available at Github [62]. Extensive evaluations on simulations and testbed demonstrate that Hawkeye is able to accurately and robustly diagnose multiple representative NPAs in RDMA networks, with > 90% average precision and ~100% recall. Hawkeye also presents high efficiency in telemetry collection and diagnosis, whose bandwidth and processing overhead are orders of magnitude lower than baselines. Hawkeye also fits well in current switch hardware with efficient resource usage.

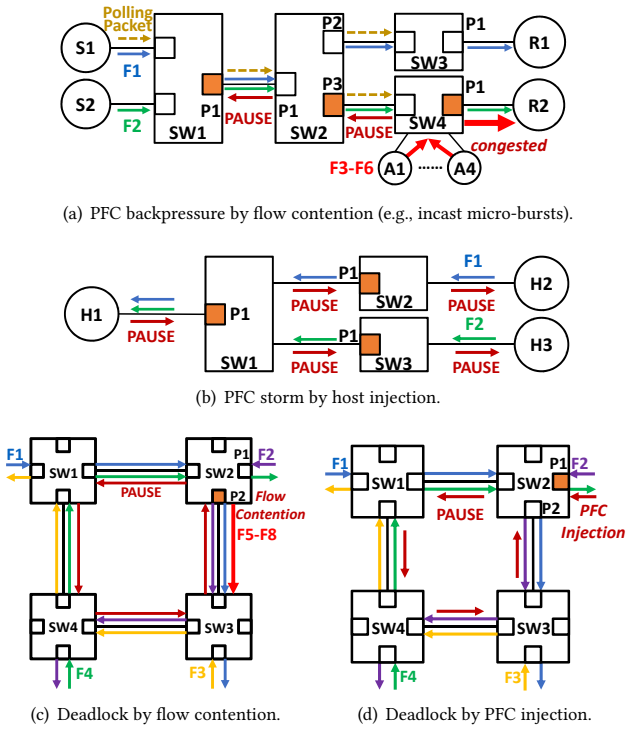
Contributions. We analyze the complexity of diagnosing RDMA NPAs (§2) and propose Hawkeye, an accurate and efficient RDMA NPA diagnosis system with PFC provenance (§3). We implement the prototype at both NS-3 simulation and Tofino hardware (§3.6). Our evaluations demonstrate the effectiveness and efficiency of Hawkeye (§4).

Ethics: This work does not raise any ethical issues.

2 Background and Motivation

RDMA requires a lossless network to achieve high transport performance[1]. In particular, RoCEv2 deploys PFC [28], the hop-by-hop flow control, to achieve the loss-free property. With PFC, when a switch's ingress queue exceeds a threshold (X_{off}) due to congestion, it sends a "PAUSE" frame to stop upstream packet transmission, and sends a "RESUME" frame when the queue drops below another threshold (X_{on}). While PFC prevents packet loss, it can potentially spread congestion in a cascading manner, causing issues like head-of-line blocking, unfairness, PFC storms, and even deadlock [20, 26]. To alleviate the side effect of PFC, conventional wisdom introduces the end-to-end congestion control [36, 40, 45, 80] to mitigate the congestion. However, PFC cannot be fully eliminated and still occurs frequently.

[1]Although lossy networks are becoming popular currently, whether lossless or lossy networks are better is still an open question and lossless networks are still widely deployed.



(a) PFC backpressure by flow contention (e.g., incast micro-bursts).

(b) PFC storm by host injection.

(c) Deadlock by flow contention.

(d) Deadlock by PFC injection.

Figure 1: PFC introduces complexity in congestion.

2.1 RDMA NPAs

Hawkeye focuses on the performance anomalies caused by switch queue congestion in networks. However, these problem are challenging to diagnose in the context of RDMA. We here present several typical cases of RDMA NPAs to illustrate their complexity.

PFC backpressure by flow contention. Flow contention, such as incast traffic, micro-bursts or ECMP load imbalance, is frequent in data center networks [6, 18, 20, 27, 59], which rarely occurs within a single switch in isolation. It typically causes PFC and spread congestion across multiple hops in RDMA networks. Therefore, victim flows even without any queue sharing with the congestion culprit may get blocked, such as F1 in Figure 1(a). Besides, the queue buildup caused by PFC falsifies the congestion signals such as queue length [80] and queuing delay [36], which may mislead congestion control to make long-term rate degradation on these victim flows [63]. PFC backpressure by flow contention can also be potentially exploited by attackers, such as LoRDMA attacks [63], thereby leading to performance degradation of innocent flows covertly.

PFC storm. We define PFC storms as the cascading PFC backpressure that caused by continuous PFC injection by hosts (as shown in Figure 1(b)), potentially leading to significant degradation of overall network performance [6, 17, 20]. Host PFC injection may originate

from diverse causes such as malfunctioning or buggy NICs, slow receiver issues caused by buffer exhaustion on the NIC, or even PCIe bottlenecks [20]. PFC storms thus present different durations and the number of paused links. In such scenarios, even without any flow contention, PFC injection can still lead to extensive or even network-wide traffic blocking.

PFC deadlock. PFC deadlock occurs when the paused links form into a cycle [20]. No packets in the deadlock links can be transmitted, and the flows will be stopped entirely until the packets in the buffers are dropped manually. PFC deadlock requires cyclic buffer dependency (CBD) [26, 27, 50, 55, 67] in the network, where each buffer waits for the next to drain out. Even in tree-based topology with loop-free routing normally, CBD can also be caused by problematic routing on single/multiple flows such as link failure, port flaps and transient/persistent routing misconfigurations or routing loops [20, 27, 67]. Once one of the ports in the CBD becomes congested, PFC pause frames will propagate and form into a cycle, i.e., deadlock. PFC deadlock can be categorized into two types based on the initial congestion. The *initiator-in-loop* deadlock is caused by the flow contention within the CBD. As shown in Figure 1(c), SW2.P2 encounters micro-bursts and triggers PFC backpressure along the CBD. The short-duration flow contention (<1ms) then leads to a persistent deadlock. The *initiator-out-of-loop* deadlock is caused by the PFC injection outside the CBD. SW2.P1 in Figure 1(d) gets PFC frames injected and spreads congestion along the CBD, resulting in a deadlock.

2.2 Complexity of RDMA NPAs

There are two key complexities in RDMA NPA diagnosis.

Frequent and transient queue congestion. The traffic in RDMA networks exhibits a wide and scattered pattern, especially the significant occurrence of bursty mice flows. Exacerbated by the prevalence of incast traffic patterns [18] and line-rate start of RDMA congestion control [13, 57, 80], mice flows can easily overwhelm the shallow buffers of switches quickly [20], resulting in frequent and transient congestion. Worse yet, even with expensive emerging switching hardware such as NVIDIA Spectrum-X [49], fine-grained load balance is still challenging in large RDMA networks [59], which further increases the possibility of congestion.

PFC complexity. PFC introduces a unique congestion pattern in RDMA networks. Unlike traditional TCP networks where switch queue congestion stems solely from local flow contention within queues, PFC-induced congestion spreads from downstream nodes independently of local queue conditions. This creates novel performance anomalies analyzed in §2.1. Worse yet, PFC invalidates conventional diagnostic approaches based on local flow interaction analysis [66], and such methods cannot identify root causes located multiple hops away. As Figure 1(a) demonstrates, F1's degradation at SW1.P1 actually originates from the bursts at SW4.P1 (propagated via PFC). Only analyzing local contention between F1 and F2 would miss the true root cause at SW4.P1, which is hops away and even out of F1's path.

2.3 Problems of Current Work

In this subsection, we summarize why existing methods fall short in accurately and efficiently diagnosing RDMA NPAs.

Current solutions fall short in fine-grained PFC visibility. Current solutions in traditional TCP networks, either host-based [19, 21, 46] or network-based [22, 24, 39, 47, 58, 66, 79], lack the visibility of PFC-related information, such as port status and the PFC impact on each flow. As a result, it is challenging for host-based solutions (even RDMA-specific solutions [20, 41]) to accurately determine whether a flow has encountered PFC-related anomalies based solely on end-to-end metrics (e.g., latency). Many traditional in-network systems also have no capability of effective PFC telemetry monitoring, and thus are unable to detect PFC frames or PFC-related anomalies. Although some emerging studies in RDMA networks monitor the events such as PFC status [20], pause frames [79] and NACK messages [17], fine-grained PFC visibility, including the PFC status at per-packet level and PFC impact on each flow, is still lacking. For example, PFC watchdog [6, 17, 20], a typical industrial monitoring tool, checks the PFC status of switch ports periodically. However, the polling period is hundreds of milliseconds or even seconds, which may miss massive transient PFC congestion. Besides, the port-level monitoring lacks fine-grained records of the performance impact on each flow, and thus cannot help identify the root causes for the victim flows.

Current solutions fall short in fast and efficient PFC causality tracing. Current diagnosis mechanisms fall short in identifying the PFC-relevant switch and collecting its telemetry. Some fine-grained monitoring systems [22, 37, 47, 58, 79] mainly focus on the flows within a single switch, which thus cannot find the PFC root cause located hops away. A straw-man causality tracing method is to collect the complete telemetry information from all switches in the network [24] and analyze the PFC causality relationship for the victim flow. However, full telemetry collection usually incurs high overhead for both collection and diagnosis. Another method is to collect the telemetry of the victim flow path's switches, such as SpiderMon [66]. Nevertheless, it may miss a part of the PFC spreading path, especially the initiator node, since the PFC path does not always overlap completely with the victim flow path. Additionally, some solutions [20, 67] support loop detection on PFC spreading paths. For example, PFC watchdog can only check the port PFC status on a single switch, and the operator should analyze the PFC spreading path by manually checking multiple switches. Besides, ITSY [67] sends probing packets to the upstream PFC causal ports to detect PFC deadlock, but it ignores non-loop PFC backpressure and cannot diagnose the root cause of PFC anomalies.

Current solutions fall short in accurate anomaly diagnosis. Currently, traditional diagnosis studies usually analyze the flow contention within the switch queues on the victim flow path. They find the main contributor by flow statistics [4, 22, 39, 46, 58, 60], queue measurement [10, 37], or provenance [66, 68]. However, these diagnosis solutions become less accurate in RDMA networks and fail to completely describe the whole anomaly causality. First, the diagnosis analyzer cannot identify whether congestion is caused by local flow contention or remote PFC, and how badly the PFC is pausing the victim flow due to the lack of PFC consideration in diagnosis algorithms. Second, the diagnosis analyzer cannot identify the true anomaly type due to the lack of PFC causality understanding. For example, PFC deadlock will be misdiagnosed if the loop is not recognized. Third, the diagnosis analyzer cannot attribute the anomaly to the root-cause flow events, even with

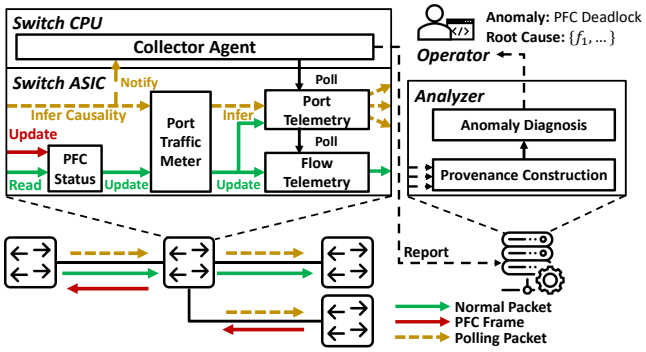
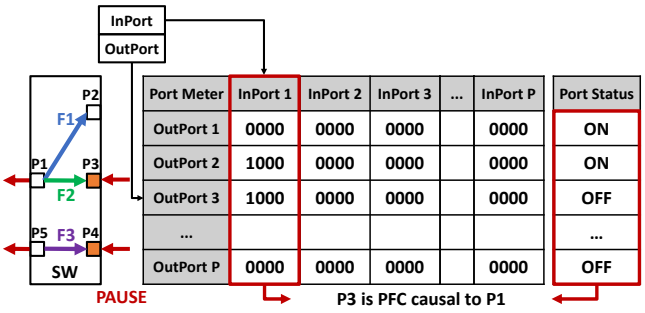


Figure 2: HAWKEYE framework.



Port Meter	InPort 1	InPort 2	InPort 3	...	InPort P	Port Status
OutPort 1	0000	0000	0000		0000	ON
OutPort 2	1000	0000	0000		0000	ON
OutPort 3	1000	0000	0000		0000	OFF
...						...
OutPort P	0000	0000	0000		0000	OFF

Figure 3: HAWKEYE PFC causality structure.

extensive flow interaction analysis in current switches. For the root cause flows (e.g., bursts in SW4.P1 in Figure 1(a)) outside the victim flow path (e.g., F1 in Figure 1(a)), the diagnoser can only attribute the anomalies to flows sharing queues with the victim flow, and cannot catch the real root causes correctly.

3 HAWKEYE Design

3.1 Design Goal

Based on the above analysis, diagnosing RDMA network performance anomalies should fulfill the following goals: 1) It should provide fine-grained PFC visibility. The monitor should capture flow victimization characteristics, distinguishing between PFC-induced congestion and conventional flow contention while quantifying respective victimization extents. 2) It should efficiently collect the telemetry relevant to the victim flows. The system must quickly and exactly identify all causally relevant switches on both the victim flow path and PFC spreading path, to comprehensively cover the anomaly provenance while minimizing the size of collected telemetry. 3) It should accurately identify the anomaly and the corresponding root cause. After collecting the telemetry, the analyzer should comprehensively understand the anomaly, including the detailed anomaly scenario (e.g., PFC-related anomalies or normal flow contention), how the PFC spreads (i.e., the PFC spreading path), and where the anomaly originates (e.g., micro-bursts, load imbalance or host PFC injection).

3.2 Overview and Workflow

Figure 2 shows the overall framework of HAWKEYE. First, HAWKEYE passively records the telemetry with PFC visibility and causality awareness (§3.3). It enables packet-level port PFC status visibility to record PFC impact, and logs PFC causality to trace PFC spreading. It then aggregates the detailed flow and port telemetry information augmented with PFC impact for comprehensive diagnosis. Second, HAWKEYE deploys fast in-data-plane PFC causality analysis and CPU-based telemetry collection (§3.4). To start diagnosis and trigger the PFC causality analysis, we set a host-based agent monitoring the flow performance. Once performance degradation is detected, the detection agent will send a "polling" packet with the victim flow information encoded into the network. When a HAWKEYE switch receives the polling packet, it infers the causal neighboring switches relevant to this anomaly at line rate and distributes the polling packets to them. Simultaneously, the switch notifies its CPU to perform asynchronous telemetry collection without compromising packet forwarding. Third, HAWKEYE runs a provenance-based offline diagnosis algorithm (§3.5). The analyzer constructs a heterogeneous provenance graph describing the comprehensive anomaly breakdown, including the flow victimization extents, PFC causality, and each flow's contribution to this anomaly. Then it conducts a diagnosis procedure to trace the congestion causality and pinpoint the root causes.

3.3 Telemetry Logging

Awareness of PFC visibility and causality. To support PFC-related telemetry, HAWKEYE sets up a PFC status register at the egress pipeline for each port, as shown in the "Port Status" column in Figure 3. When receiving a PFC frame at the ingress pipeline of a specific port, HAWKEYE passes the frame into the egress pipeline, extracts the pause time encoded in the frame and updates the port status and remaining pause time (refer to the red line in Figure 6)[2]. Accordingly, when a data packet enqueues during the paused status of the port, it is regarded as a paused packet, and the corresponding telemetry is updated correspondingly. Furthermore, to analyze PFC causality, HAWKEYE records the port-level traffic causality at the egress pipeline, as shown in Figure 3. Specially, to monitor the PFC spreading contribution between ports, HAWKEYE sets up a traffic meter for each port pair to record the traffic volume. This is much finer grained than existing approaches like ITSY [67] that uses only a single bit to note the presence of traffic between ports. For example, as shown in Figure 3, although both P3 and P4 are PFC congested, P4 is irrelevant to the PFC backpressure at P1 because there is no traffic sending from P1 to P4, and thus P4 does not contribute to the packet accumulation at P1. Therefore, once the upstream switch connected with P1 complains about its PFC congestion, this switch can identify the causal ports relevant to the PFC backpressure as P3.

Epoch-based multi-granularity telemetry. HAWKEYE logs flow-level and port-level telemetry in an epoch-based manner for fine-grained monitoring, as shown in Figure 4. The switch maintains a set of epochs in the egress pipeline as a ring buffer. To index the epoch to which a packet belongs at line rate, we utilize the timestamp in the packet metadata. Specifically, programmable switches

[2]To enable PFC awareness of HAWKEYE on Tofino, we have to pass PFC frames into P4 pipelines and record PFC status using extra registers, since Tofino does not support the monitoring of certain PFC features natively, such as the real-time PFC status of each port.

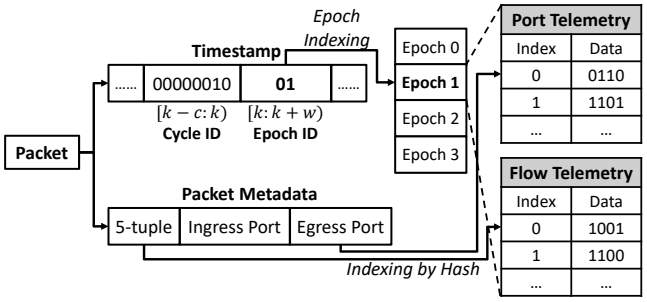


Figure 4: Telemetry layout.

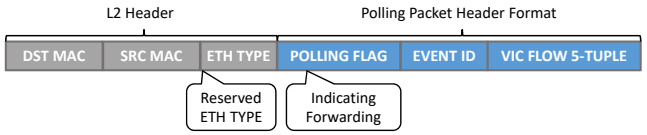


Figure 5: Hawkeye polling packet format.

assign each enqueued packet a unique 48-bit timestamp with nanosecond granularity, where the least significant bit corresponds to 1 ns. Consequently, we can select several bits from this 48-bit timestamp to demarcate the epoch. For example, assuming that the epoch size is 1 ms, which is approximately 2^{20} ns, we can utilize `timestamp[21:20]` to identify indices of different epochs, as shown in Figure 4. In addition, to distinguish the sequential order of different epochs in the ring buffer, Hawkeye also records the 8 bits preceding the epoch index as the epoch ID, e.g., `timestamp[29:22]`. Once the epoch ID in the incoming packet timestamp is newer than that stored in the current epoch, which means a wrap-around, the register value will be reset and start to count from zero.

For flow-level telemetry, Hawkeye records the flow 5-tuple, packet count, and egress queuing depth, similar to existing work [22, 47]. Additionally, to enable fine-grained PFC visibility, Hawkeye also records the number of packets paused by PFC for each flow. For each incoming packet, Hawkeye first locates the right flow telemetry table based on its epoch, and the slot in this table is indexed by the hash value of the packet's 5-tuple. Then a bit-wise XOR between the packet's 5-tuple and the 5-tuple in the slot is executed. The result 0 indicates this packet belongs to an existing flow and the telemetry will be updated accordingly; the result 1 indicates this packet belongs to a new flow, and the existing entry will be evicted and stored at the controller. Moreover, since PFC spreads hop by hop, diagnosing the anomaly usually requires aggregating flow-level data into port level, requiring considerable computation overhead on the data plane. We therefore record the port-level telemetry in the egress pipeline, to avoid this expensive computation. For each port, Hawkeye records the number of packets paused by PFC and egress queue depth. The port-level data is indexed by the port number in each epoch, and is also updated by each incoming packet, similar to the flow-level data.

3.4 Telemetry Collection

Anomaly-driven detection agent. Rather than monitoring anomalies on switches [66, 79], Hawkeye detects the NPA by deploying a host-based detection agent. The detection agent monitors the

Table 1: Polling flag specifications.

Polling Flag	Meaning
00	Useless tracing
01	(**Default**) Only trace along victim flow path
10	Only trace along PFC causality
11	Trace both of them

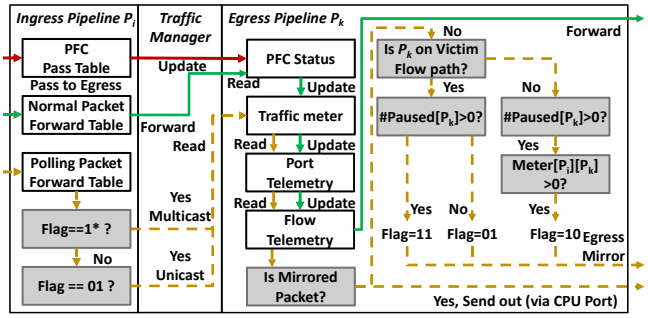


Figure 6: Hawkeye PFC causality analysis.

end-to-end performance of flows (e.g., RTT), and triggers an anomaly diagnosis event when the performance degradation exceeds a threshold. This host-side triggering mechanism provides two key advantages over switch-triggering approaches. First, host-side triggering eliminates duplicated efforts of PFC tracing. Due to the cascading property of PFC congestion, there are usually multiple switches on a PFC path detecting the anomaly simultaneously and initiating the procedure of PFC causality analysis. In contrast, by sending polling packets from the source host and piggybacking PFC tracing instructions, each switch forwards the extra control packets only once to completely cover PFC causality. Second, in-network monitoring cannot obtain some high-level performance metrics, such as RTT [36, 45], flow completion time [72] or job completion time [53], which usually require massive information from host-side applications.

Once performance degradation is identified, the host-end detection agent generates a polling packet into the network, which is assigned to a separate queue that is not paused by PFC, to further trigger the telemetry collection and diagnosis. The polling packet format is illustrated in Figure 5. By default, the polling flag is set to `01` (see Table 1), instructing switches to forward the packet along the path of the victim flow, whose 5-tuple is encoded in the polling header. Once PFC is detected on the victim flow path, it will also be forwarded to the PFC spreading path to trace the causality (flag `1*`), so that the complete anomaly causality is covered. To avoid duplicate detection for a single flow, Hawkeye drops polling packets with the same 5-tuple within a certain time interval. All polling packets are set to the same priority as control packets (e.g., CNP) to avoid potential queuing delay. The forwarding of polling packets is conducted exclusively by the switch data plane, as shown in the gold dotted line Figure 6.

In-data-plane causality analysis. Hawkeye presents a line-rate causality analysis mechanism that can accurately locate all causally relevant switches on the paths of the victim flow and PFC spreading, without covering unnecessary ones or introducing additional overhead. Specifically, each Hawkeye switch analyzes its PFC causality

and further forwards the polling packet to the causally relevant switches. The causality analysis workflow of polling packets inside a switch is demonstrated in Figure 6. First, when a switch on the victim flow path receives a polling packet marked as `01` (e.g., SW1 in Figure 1(a)), the polling packet is unicasted to the same port as the victim flow by extracting the IP address (in the 5-tuple) encoded in the packet. The switch then checks the number of paused packets on the egress pipeline. If the victim flow is PFC paused, the higher bit of the polling flag is set, notifying the next switch (from which PFC frames spread) to analyze its PFC causality. The ingress pipeline of the downstream switch (e.g., SW2), multicasts the `1*` polling packet to each egress port, so that every port analyze wether it is related to the anomaly. They check the PFC causality structure (maintained as Figure 3) and only emits the polling packets on the ports causally relevant to this PFC congestion (e.g., SW2.P3). Such PFC causality analysis procedure is conducted recursively hop by hop, until the last switch whose egress port is connected to a host or has no PFC paused packets, which respectively means the PFC comes from host injection or the flow contention within this queue. Note that, once receiving the polling packet, each switch will also notify its controller through mirroring the polling packet to the CPU port, triggering the asynchronous telemetry collection. Consequently, all the telemetry data on the PFC spreading path and victim flow path will be collected, and the polling packet is transmitted at line rate without disrupting the collection.

Controller-assisted data collection. Upon receiving the polling packet, the switch reports the telemetry it stored to the analyzer server. Although programmable switches are able to generate packets and take telemetry data as the payload, it is cumbersome to dump all the data fully using data plane packet generation. Since each stage only allows one memory access, dumping the telemetry for all flows requires numerous recirculations. Besides, due to the limited PHV field size of programmable switches, a large number of packets need to be generated to dump all the data. Consequently, in-data-plane packet generation will impose a non-negligible overhead on traffic forwarding. To alleviate these problems, we design a controller-assisted data collection mechanism that enables the asynchronous telemetry collection without affecting the data plane forwarding. Once receiving the mirrored polling packet, the controller then starts to read the telemetry register, filters out useless telemetry such as zero-value data slots, and aggregates the data into a large packet reporting to the analyzer. The packet size can be as large as the Maximum Transmission Unit (MTU), thereby reducing the number of telemetry packets. To prevent duplicate data collection on one switch caused by multiple flows' polling packets (e.g., F1-F4 in Figure 1(c)), Hawkeye set a time interval between two telemetry collections. Note that Hawkeye can easily support multiple NPAs concurrently. If two NPAs do not have the path overlap, their telemetry data can be collected and diagnosed independently.

3.5 Provenance-based Diagnosis

Hawkeye proposes a diagnosis paradigm that can completely analyze the anomaly causality. Due to the PFC complexity (§2.2), diagnosing RDMA NPAs should answer the following questions: 1) whether the victim flow encounters normal flow contention or PFC at each hop and their corresponding impact extents; 2) how PFC spreads to the victim flow path and which flow contributes to this spreading; 3) where is the initial congestion point and what is the root cause. Hawkeye addresses these challenges through: 1) constructing a heterogeneous wait-for provenance graph to comprehensively describe the congestion causality(§3.5.1); 2) identifying the anomaly case and locating the root causes by analyzing the provenance and matching anomaly signatures (§3.5.2).

3.5.1 Provenance Graph Construction. The heterogeneous wait-for provenance graph encodes the congestion causality between flows and ports. Specifically, flows and ports are represented as the nodes in the graph, and directed weighted edges between the nodes are defined to present their wait-for relationship [76]. Hawkeye first adds port-level edges presenting the PFC causality, then adds flow-port edges demonstrating the PFC pausing impact on flows, and finally adds port-flow edges indicating flow contention. The formal procedure is described in Algorithm 1.

Port-level edges. We define the port-level provenance graph to describe PFC causality including the spreading path and the initial congestion point. For a port P_i paused by PFC, it actually *waits for* the downstream congested ports to drain out the queues. Therefore, we define a wait-for directed edge from a congested egress port P_i (e.g., SW2.P3 in Figure 1(a)) to each downstream congested egress port P_j (e.g., SW4.P1 in Figure 1(a)). For P_j, its contribution to the PFC congestion at P_i is determined by the traffic rate between P_i and P_j, $meter[P_i][P_j]$, and the queue buildup of P_j, $qdepth[P_j]$. Therefore, we define the edge weight from P_i to P_j as $w_{ij} = paused\_num[P_i] * \frac{meter[P_i][P_j]}{\sum_k meter[P_i][P_k]} * qdepth[P_j]$. For P_j with N packets enqueuing over a period, $qdepth[P_j]$ can be calculated as the average $\frac{1}{N} * \Sigma_{k\in[1,N]} qdepth(pkt_k)$.

Flow-port edges. We define flow-port edges to represent which ports pause a certain flow and the corresponding severity. Specifically, for a flow f_i which traverses through a PFC-paused port P_j, such as F1 in SW1.P1 in Figure 1(a), f_i is waiting for P_j to restart packet forwarding. We hence add an edge from f_i to P_j, and set the weight as the number of paused packets for f_i at P_j, denoted as $paused\_num(f_i, P_j)$.

Port-flow edges. Conversely, for some ports encountering flow contention, they are actually waiting for the flows to be drained from their queues. We characterize the wait-for relationship from a port to a flow using the flow's contribution to the overall flow contention in the port. When a packet pkt of flow f_i enqueues with $qdepth(pkt)$, a flow-level wait-for edge from f_i to f_j can be defined with the weight as the number of f_j's packets in $qdepth(pkt)$, denoted by $x_j(pkt)$. Therefore, the average wait-for edge from f_i to f_j over a period can be calculated as $w(f_i, f_j) = \Sigma_{pkt_k \in f_i} x_j(pkt_k)/N_i$, where N_i is the number of f_i's packets in this period. This wait-for weight is exactly the congestion contribution of flow f_j to f_i. However, f_j's contribution to the overall flow contention can not be described by directly aggregating the wait-for weights to f_j from every f_i, since f_j may also be blocked by other flows. For example, a large flow blocked by short micro-bursts may have a high wait-for weight from other flows, although it is actually a victim [37]. Therefore, the wait-for weight from f_j to other flows should be subtracted. We hence finally denote the *port-flow* wait-for weight to f_j as $\Sigma_i w(f_i, f_j) - \Sigma_k w(f_j, f_k)$. Flows with *positive* and *negative*

Algorithm 1: Provenance Construction Procedure.

Input: P - Port list in reported telemetry; F - Flow list in reported telemetry; T - Epoch size; N - Network topology
Output: G - Provenance graph

 # Construct port-level provenance
for $p_i \in P$ **do**
  G.AddVertex$(p_i, PORTLEVEL)$
  $qdepth[p_i] = total\_qdepth[p_i]/pkt\_num[p_i]$
  $sum\_meter[p_i] = \Sigma_{p_j \in p_i.peer\_switch} meter[p_i][p_j]$
for $p_i \in P$ **do**
  for $p_j \in p_i.peer\_switch : meter[p_i][p_j] > 0$ **do**
    $weight = paused\_num[p_i] * meter[p_i][p_j] * qdepth[p_j]/sum\_meter[p_i]$
    G.AddEdge$((p_i \rightarrow p_j), weight)$
for $f_i \in F$ **do**
  G.AddVertex$(f_i, FLOWLEVEL)$
 # Construct flow-port provenance
for $(f_i, p_j) \in F \times P : paused\_num(f_i, p_j) > 0$ **do**
  G.AddEdge$((f_i \rightarrow p_j), paused_num(f_i, p_j))$
 # Construct port-flow provenance
for $p_i \in P$ **do**
  Seq = ReplayQueue(p_i)
  $Contrb$ =Contribution(Seq)
  for $f_j \in F$ **do**
    G.AddEdge$((p_i \rightarrow f_j), Contrb[f_j])$
Function ReplayQueue(p):
  for $f_i \in F$ **do**
    for $j \in pkt\_num(f_i, p)$ **do**
      $time = j * T/pkt\_num(f_i, p)$
      $enqueue\_list+ = (pkt_j, time)$
  Sort $enqueue\_list$ by ascending order of $time$
  return $[pkt \in enqueue\_list]$
Function Contribution(Seq):
  $Contrb = \{\}$ $W = \{\{0\}\}$ $w = \{\{0\}\}$
  for $pkt_i \in Seq$ **do**
    for $pkt_j \in qdepth(pkt_i)$ **do**
      $W[f(pkt_i)][f(pkt_j)]+ = 1$
  for $(f_i, f_j) \in F \times F$ **do**
    $w[f_i][f_j] = W[f_i][f_j]/pkt\_num[f_i]$
  for $f_j \in F$ **do**
    $Contrb[f_i] = \Sigma w(f_i, f_j) - \Sigma w(f_j, f_i)$
  return $Contrb$
return G

weights are contention *contributors* and *victims*, respectively. Note that in real networks, the queue may experience PFC pause and flow contention simultaneously. The packets in the queue may also includes paused packets, which is not congested by local flow contention. In this case, the port-flow edge construction excludes the paused packets in queues.

3.5.2 Anomaly Breakdown and Diagnosis. The provenance graph provides a comprehensive breakdown for the anomaly causality, including the victimization extent from flow contention and PFC at each hop, the PFC spreading causality, and each flow's contribution. We present the signatures of representative RDMA NPAs mentioned in §2.1, describing the corresponding graph feature of each anomaly. Their detailed formal definitions are shown in Table 2, and their example provenance graphs are demonstrated in Figure 12. Hawkeye traverses the graph and matches the signatures to identify the anomaly and locate the anomaly-contributing flows, including the root causes and PFC-spreading ones. The formal diagnosis procedure is shown in Algorithm 2. The supported signatures can be extended if more anomalies are identified.

Table 2: Representative signatures.

Anomaly	Root Cause	Signatures
Micro-bursts incast	Flow contention (Micro-bursts)	$\exists path \subseteq G_P, \exists p \in path :$ $(\text{out-deg}_P(p) = 0$ $\wedge (\exists F_c \subseteq F, \forall f \in F_c :$ $(p, f) \in E_{PF}$ $\wedge \text{weight}(p, f) > 0 \wedge \text{burst-flow}(f)))$
In-loop deadlock	Flow contention	$\exists loop \subseteq G_P : (\forall p \in loop :$ $(\text{out-deg}_P(p) = 1$ $\wedge \forall p' : (p, p') \in E_P \rightarrow p' \in \text{loop}))$ $\wedge (\exists p'' \in loop, \exists F_c \subseteq F, \forall f \in F_c :$ $(p'', f) \in E_{PF} \wedge \text{weight}(p, f) > 0)$
Out-of-loop deadlock	Flow contention	$\exists loop \subseteq G_P, \exists p \in loop, \exists p' :$ $(\text{out-deg}_P(p) > 1$ $\wedge \exists path(p, p') \subseteq G_P : \text{out-deg}_P(p') = 0$ $\wedge (\exists F_c \subseteq F, \forall f \in F_c :$ $(p', f) \in E_{PF} \wedge \text{weight}(p', f) > 0))$
Out-of-loop deadlock	Host PFC injection	$\exists loop \subseteq G_P, \exists p \in loop, \exists p' :$ $(\text{out-deg}_P(p) > 1$ $\wedge \exists path(p, p') \subseteq G_P : \text{out-deg}_P(p') = 0$ $\wedge \forall f : (p', f) \in E_{PF} \rightarrow \text{weight}(p', f) \leq 0)$
PFC storm	Host PFC injection	$\exists path \subseteq G_P, \exists p \in path :$ $(\text{out-deg}_P(p) = 0$ $\wedge \forall f : (p, f) \in E_{PF} \rightarrow \text{weight}(p, f) \leq 0)$
Normal flow contention	Flow contention	$(\forall u, v \in G_P : u \neq v \rightarrow (u, v) \notin E_P)$ $\wedge (\exists p \in G_P, \exists F_c \subseteq F, \forall f \in F_c :$ $((p, f) \in E_{PF} \wedge \text{weight}(p, f) > 0))$

PFC backpressure by flow contention. This anomaly case is characterized by 1) a high number of paused packets of the victim flow, 2) a PFC path on the port-level sub-graph, and 3) flow contention at the initially congested port. As shown in Figure 12(a), F1 is paused at SW1.P1 by PFC instead of normal flow contention (the dotted flow-port lines). And the initial node at the PFC path has a large number of outgoing port-flow edges, remarking that the root cause is flow contention. The major contributing flows are also remarked by the positive weights (marked in red) directed to them. The flow contention cause can be further analyzed. For example, incast bursts can be identified by analyzing the contributing flows' paths and throughput [46, 79], and load imbalance can be located by calculating ECMP imbalance ratio. We omit these details orthogonal to our work which can be further integrated into Hawkeye. Besides the root-cause flows, other anomaly-contributing flows can be identified. For example, F2 is responsible for the PFC spreading from SW2.P3 to SW1.P1, since it is paused at both ports.

PFC storm. Similarly, Hawkeye locates the initially congested port node along the port-level edges in the provenance graph. However, since PFC storm is usually caused by host PFC injection, the initial port node has no flow contention, i.e., no positive outgoing port-flow edges. Therefore, the root cause can be located as host issues, as shown in Figure 12(b).

Initiator-in/out-of-loop deadlock. Hawkeye detects PFC deadlocks by searching for loops starting from the port pausing the victim flow. Hawkeye further identifies the deadlock types and

root causes by analyzing the port-port and port-flow edges. Specifically, on one hand, the initially congested queue may be located in the port within the deadlock loop (i.e., initiator-in-loop deadlock), which is usually caused by the accidental flow contention. In this case, one of the port nodes in the loop will have multiple outgoing positive edges to a set of flows, as shown in Figure 12(c). On the other hand, PFC can be injected by a host or caused by flow contention in a port out of the loop (i.e., initiator-out-of-loop deadlock). In this case, there will be a port node in the loop with multiple neighboring port nodes, which leads to a PFC path ending at the initial congestion node, as shown in Figure 12(d). Analyzing the port-flow edges on it can identify whether the root cause is flow contention or PFC injection. The PFC spreading causality of Hawkeye also enables analysis on circular buffer dependency (CBD) for deadlock prevention and resolution [20, 27, 67]. As shown in Figure 12(c), F1-F4 causes the PFC spreading loop. Further troubleshooting, such as routing configuration checking [7] can be conducted.

Traditional congestion by flow contention. Although flow contention congestion in RDMA networks is almost impossible without generating PFC, it can also be diagnosed by Hawkeye, which degenerates into a traditional congestion diagnosis. In this case, there is no port-level edges because of no PFC spreading path. But there is a set of port nodes containing port-flow edges with positive weights which point to the contributors of flow contention.

3.6 Implementation

We implement a prototype of Hawkeye at Intel Tofino hardware using ~2500 lines of P4 code and ~3000 lines of C code. We also implement an NS-3 version based on HPCC simulator [2]. The root cause analyzer is implemented in Python. We implement a detection agent prototype on NVIDIA BlueField-3 DPU. The source code is publicly available at Github [62].

Host detection agent. The detection agent prototype is implemented on NVIDIA BlueField-3 DPU, based on PCC (Programmable Congestion Control) in DOCA [48]. The agent monitors each flow's RTT on the data-path accelerator via PCC's API. Once the RTT of a specific flow exceeds the threshold, the agent sends a polling packet encoding the victim flow 5-tuple and start a diagnosis. Note that, we use RTT as an example performance metric, and more metrics such as throughput, flow-completion time and even job-completion time can be integrated to support flexible detection.

Enable PFC awareness for P4. In Tofino, PFC port control is orthogonal to P4 application logic, and PFC frames are filtered out by MAC without sending to the P4 pipeline by default, leading to limited PFC visibility. To enable awareness of PFC status, we set the `filterpf` bit to 0 in `rxconfig` register on MAC so that PFC frames can be passed to the application logic. Once receiving a PFC frame at the ingress P4 pipeline, the PFC frame is then passed to the egress pipeline so that the corresponding port status register can be updated. Then the PFC-related telemetry can be maintained accordingly. Fortunately, emerging RDMA-native switches support significantly enhanced monitoring capabilities and expose interfaces for monitoring PFC status at switch ports, eliminating the need for additional PFC frame processing, thereby easing Hawkeye's implementation.

Algorithm 2: Provenance Analysis Procedure.

Data: $Neibr_F(p)$ - Neighboring flow nodes of port p; $Neibr_P(p)$ - Neighboring port nodes of port p; $\text{outdeg}_P(p)$ - Port-level outdegree for port p
Input: G - Provenance graph; N - Network topology; f_v - Victim flow

```
Function AnalyzeFlowContention(p):
    # Flow contention analysis
    if max_{f∈Neibr_F(p)} weight(p, f) ≤ 0 then
        # No flow contention
        check PFC injection from port p's peer device
    else
        for f ∈ Neibr_F(p) : weight(p, f) > 0 do
            # check flow contention causes
            check flow throughput
            check flow priority
            check ECMP imbalance ratio
Function DeadlockDiagnose(loop_list):
    # Root cause
    if ∃p ∈ loop_list : outdeg_P(p) > 1 then
        # Initial node out of loop
        for p_i ∈ Neibr_P(p) ∧ ¬p_i ∈ loop_list do
            CheckPortNode(p_i, [])
    else
        # Initial node in loop
        for p_i ∈ port_list do
            AnalyzeFlowContention(p_i)
    # Buffer dependency analysis
    print port-nodes in loops
Function CheckPortNode(p, port_list):
    # DFS-based check port
    check whether loop exists
    if port_list contains loop then
        # Deadlock
        DeadlockDiagnose(port_list)
    if outdeg_P(p) == 0 then
        # Initial nodes of PFC spreading
        AnalyzeFlowContention(p)
    for p_i ∈ Neibr_P(p) do
        CheckPortNode(p_i, port_list + p)
for p_i ∈ f_v path do
    if weight(f_v, p_i) > 0 then
        # check PFC causality
        CheckPortNode(p_i, [])
    else
        # normal flow contention
        AnalyzeFlowContention(p_i)
```

Hawkeye in Tofino hardware. The polling packet forwarding logic is implemented in the ingress pipeline to determine which port(s) to forward the polling packet. The port status, the PFC causality meter, and the telemetry are collectively implemented in the egress pipeline. To accelerate the bulk register read performance on CPU, we utilize the `REGISTER_SYNC` table operation provided by BF_Runtime, the runtime library on the controller, which utilizes DMA transfer to quickly synchronize the whole register array to the control plane. Then the register value can be quickly read inside the software.

4 Evaluation

We evaluate Hawkeye in both large-scale simulation and real testbed experiments to answer the following questions:

- Can Hawkeye accurately diagnose RDMA NPAs (§4.2)?
- How efficiently does Hawkeye collect the telemetry and diagnose the root cause (§4.3)?
- Can Hawkeye cover common RDMA NPAs (§4.4)?
- Can Hawkeye be efficiently implemented on current network hardware (§4.5)?

4.1 Experimental Setup

Topology. We set up a Fat-Tree (K=4) topology [14] with 20 switches on NS-3 simulation. The link bandwidth is 100 Gbps and the link delay is 2 μs. Our hardware testbed partitions 2 pipelines of a Tofino switch into 2 logical switches. The logical links between them are emulated by directly attached cables. Two Dell PowerEdge R7525 servers are connected to each logical switch.

Workload. We set up an empirical RoCEv2 workload to evaluate Hawkeye, which comes from an industrial operating data center and displays a long-tailed flow size distribution [54]. Specifically, < 80% of flows are smaller than 10 MB, < 90% of flows are smaller than 100 MB, and about 10% flows are 100 MB ∼ 300 MB. The arrival time of flows is based on a Poisson process, the arrival rate of flows is varied based on the link load of the network, and the source and destination of flows are selected randomly. For each anomaly scenario, we craft 100 traffic traces following the described traffic patterns with different link load. We inject a group of synchronized short-lived bursts passing through the same link to craft PFC backpressure by flow contention, configure a host to continuously inject PFC frames and cause PFC storm. We also simulate routing misconfigurations to trigger the initiator-in/out-of-loop deadlocks.

4.2 Overall Effectiveness

In this subsection, we evaluate the effectiveness of Hawkeye. We first evaluate Hawkeye's diagnosis accuracy with different parameters, and compare the accuracy with other baselines using NS-3 simulation.

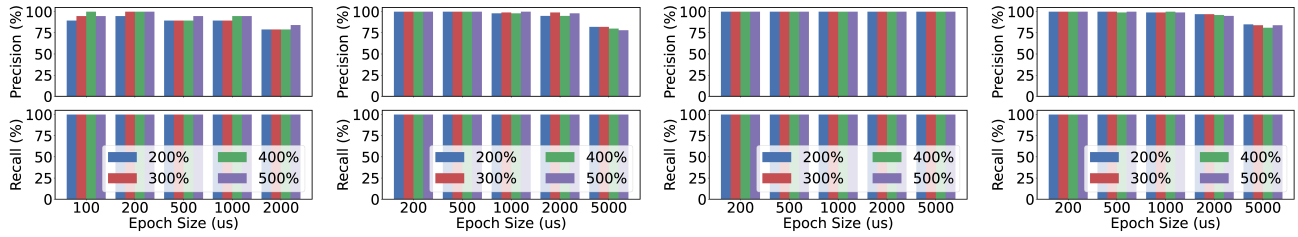
Precision & recall with different parameters. We define a true positive result *iff* it identifies both the exact anomaly case (e.g., a deadlock) and the corresponding root causes (e.g., the burst flows). False positives mean incorrect anomaly cases or root causes are reported, while false negatives occur when an existing anomaly is not reported. We evaluate the precision and recall over different PFC anomalies, as mentioned in §3.5.2, with different combinations of parameters: detection threshold and epoch size. The threshold is normalized as percentage of RTT (i.e., 200% ∼ 500% RTT), and the epoch size varies from 100us to 2ms. As shown in Figure 7, Hawkeye can achieve 100% precision and recall with correctly configured parameters, and the precision is mainly affected by the epoch size of the telemetry. Specifically, as the epoch increases, the accuracy of flow contention identification decreases, thereby reducing the locating precision of transient burst flows and the diagnosis precision. Besides, a too long epoch may mistakenly correlate two events and make wrong causality. For example, a slight flow contention ending before the host PFC injection within the same epoch maybe mistakenly regarded as one NPA. Then the diagnosis may be misled into attributing the PFC causes to flow contention, reducing the precision consequently. Considering the recall, since Hawkeye triggers diagnosis once RTT exceeds the threshold, nearly all the anomalies can be reported and thus false negativeness is rare.

Precision& recall v.s. baselines. We compare the precision and recall between Hawkeye and other baselines in diagnosing PFC-related anomalies and normal flow contention. We deploy SpiderMon [66] and NetSight [24] as the baselines from traditional networks. SpiderMon monitors the queuing delay of every packet and reactively analyzes the flow interaction within the queues on the victim flow path; NetSight collects per-packet postcards at each switch across the network. Since traditional solutions cannot handle RDMA NPAs well, we also propose another 2 baselines derived from Hawkeye. The "full polling" method collects the complete telemetry across all the switches in the network. The "victim-only" method only polls the telemetry on the victim flow path without tracing the switches on PFC spreading paths. As the sub-optimal results of Hawkeye are already presented in Figure 7, we demonstrates the upper bound of precision and recall each method presents over different anomalies with its optimal parameters, as depicted in Figure 8. Hawkeye shows an upper bound of accuracy at the same level as the full polling method, since it also covers the complete causal telemetry. However, only collecting telemetry on the victim flows presents much lower precision, especially on deadlock diagnosis, since the causality coverage is limited. Specifically, when diagnosing burst-induced PFC backpressure, PFC storm, and normal flow contention, the victim flow triggering the diagnosis may pass through the initial congestion point, which means the PFC path is exactly the victim flow path. Therefore, the precision of the victim-only method is closed to Hawkeye. In PFC deadlock scenarios, each flow typically occupies only a part of the loop. As a result, tracing only the victim flow path results in incomplete provenance graph and lower accuracy. Furthermore, since traditional baselines do not have the visibility for PFC, they can hardly diagnose the PFC-related NPAs, despite high effectiveness on normal flow contention.

4.3 Efficiency Breakdown

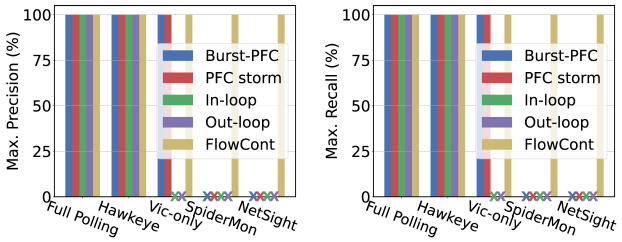
We evaluate the efficiency of each component in Hawkeye.

Telemetry logging effectiveness. Hawkeye logs port-level and flow-level telemetry to construct complete causality and make fine-grained diagnosis. We evaluate the effectiveness of the telemetry at different granularity. We set up 2 baselines: 1) Port-level only telemetry system, which only stores the port-level telemetry including the paused packet count at each port and the port-level traffic meter. The port-only method still supports in-network PFC causality analysis. 2) Flow-level only telemetry system, which only stores the flow-level telemetry including the paused packet count and queue depth for each flow. The flow-only method is thus unable to trace PFC due to the lack of port-level traffic causality. Figure 10 demonstrates the precision and recall of different methods for monitoring traffic containing mixed different anomalies. Two baselines present much lower performance, due to the lack of complete causality. Specifically, the port-level method cannot identify the flow contention and therefore misses the root causes, although it can detect the PFC path. And the flow-level method can



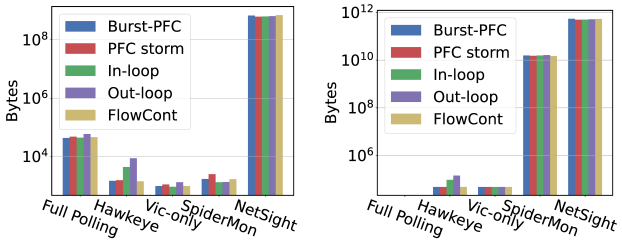
(a) PFC backpressure by flow contention. (b) PFC storm by host injection. (c) Initiator-in-loop deadlock. (d) Initiator-out-of-loop deadlock.

Figure 7: Precision & recall of typical anomaly cases over different epoch sizes and thresholds.



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

Figure 8: Precision & recall upper bound v.s. baselines.



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

Figure 9: Overhead v.s. baselines.

not identify PFC path, leading to wrong diagnosis, as analyzed in §2.3.

Processing overhead. We compare the processing overhead using the size of the telemetry collected for diagnosis. As shown in Figure 9(a), the full polling method collects the telemetry on all switches, which includes a huge amount of irrelevant telemetry data. The victim-only method merely collects the telemetry on the victim flow path, showing much lower processing overhead despite lower accuracy. Hawkeye collects the telemetry on the switches along both victim flow paths and PFC spreading paths, thus exhibiting approximately higher processing overhead than the victim-only method. SpiderMon collects the flow telemetry along the victim flow path with 36 bytes per flow, showing similar telemetry amount to the victim-only method. NetSight collects all the postcards for every packet at each hop, and consequently results in significant overhead.

Monitoring bandwidth overhead. We measure the additional bandwidth overhead during the monitoring introduced by different methods in Figure 9(b). Hawkeye monitors and records the telemetry passively without extra traffic, and it only generates additional polling packets to the switches on victim flow paths and PFC spreading paths when an anomaly is detected. Therefore, it presents

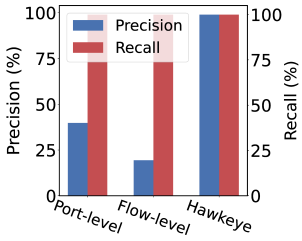


Figure 10: Diagnosis effectiveness of different telemetry systems.

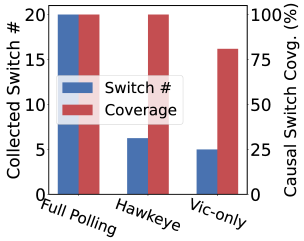


Figure 11: Count of Collected switch & causal switch coverage ratio.

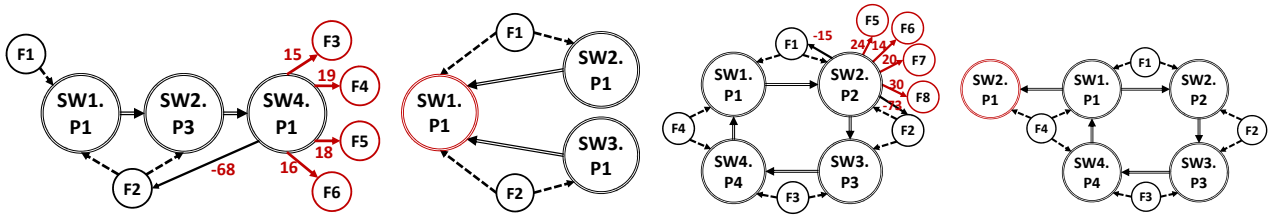
very low bandwidth overhead. The victim-only method presents approximately lower bandwidth overhead because the polling packets are only forwarded to the switches on the victim flow path. The full polling method causes no extra bandwidth since it does not need polling packets to trigger the telemetry collection on switches. SpiderMon adds an extra 16-bit header field in every packet to record the cumulative delay, leading to higher bandwidth overhead. NetSight generates considerable per-packet additional bandwidth overhead, with about 15 bytes per packet and per average hop count due to the postcard for the packets at every switch.

Telemetry-collected switch. Hawkeye collects the telemetry on the switches related to the anomaly via in-network PFC causality analysis. We evaluate the collected switch count and the coverage ratio of the causally relevant switches, as Figure 11 shows. Compared to full polling, Hawkeye collects a much smaller number of switches while ensuring 100% coverage of causal switches. In contrast, the victim-only method exhibits low causality coverage due to not tracing the PFC, despite a lower count of switches collected. Therefore, the complete provenance cannot be constructed, and the accuracy is thus lower. To summarize, through in-network PFC causality analysis, Hawkeye efficiently collects the anomaly causal switches with low collection scale.

4.4 Case Study

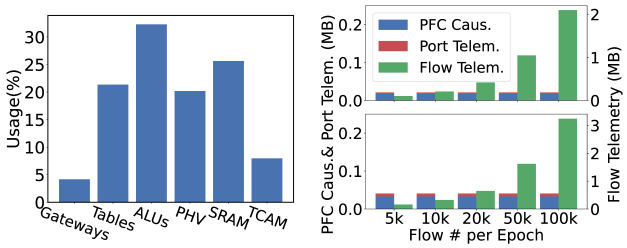
We demonstrate Hawkeye's diagnosis effectiveness by constructing the provenance graphs of the typical NPAs in §2.1.

PFC by incast micro-bursts. As shown in Figure 1(a), multiple line-rate micro-burst flows are injected into SW4.P1 by devices A1~A4 and spread PFC to SW1.P1. As Figure 12(a) shows, the victim flows F2 and F1 are paused at SW1.P1 and SW2.P3 by PFC (dotted edges to port nodes), and the port-level graph indicates that the initial congestion point is SW4.P1. Inside SW4.P1, the main



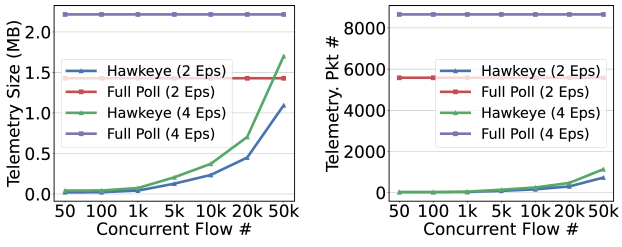
(a) PFC by incast micro-bursts. (b) PFC storm. (c) Initiator-in-loop deadlock. (d) Initiator-out-of-loop deadlock.

Figure 12: Provenance graphs for typical anomalies. Port-flow and port-level edge weights are omitted for simplicity.



(a) Resource usage summary. (b) Resource usage with 2 & 4 epochs.

Figure 13: Hardware resource usage.



(a) Telemetry size reduction. (b) Telemetry packet reduction.

Figure 14: CPU reduces telemetry collected.

contributor flows F3-F6 (connected with red edges) are identified as root causes.

PFC storm. The PFC injection from H1 in Figure 1(b) causes PFC storm in the network. Figure 12(b) demonstrates the PFC path from SW1.P1 to SW2 and SW3. With no flow contention in SW1.P1, the congestion is due to host PFC injection.

Initiator-in-loop deadlock. Figure 1(c) shows the PFC deadlock caused by the flow contention in the loop. The provenance graph in Figure 12(c) identifies the deadlock through a loop of port-level edges. Since each port node has a port-level out-degree < 2, the initial congestion point lies inside the loop. Analyzing the port-to-flow edges for each port reveals the root cause of flow contention at SW2.P2.

Initiator-out-of-loop deadlock. Figure 1(d) shows the PFC deadlock caused by PFC injection out of the loop. The provenance graph in Figure 12(d) first identifies the deadlock through the loop. Among the port nodes in the loop, SW1.P1 has another outgoing path destined to SW2.P1 which has no flow contention, and the PFC injection can be identified.

4.5 Real Testbed Evaluation

We implement HAWKEYE at a real testbed, and validate its deployability and efficiency accordingly.

Switch resource usage. Figure 13(a) shows HAWKEYE's hardware resource usage, which fits well on Tofino. Furthermore, HAWKEYE's memory usage scales well with the epoch count monitored and the maximum flow count per epoch. As shown in Figure 13(b), the memory usage of PFC causality structure (Figure 3) and port-level telemetry is small and constant, bounded by the number of switch ports, while the flow-level telemetry increases with $O(\#flow)$.

CPU poller. We first measure the total time usage of polling the telemetry from switches using the on-switch CPU. Polling full telemetry from one switch, including 2 or 4 epochs, requires approximately 80 and 120 ms, respectively, with each epoch containing the data of 64 ports and 4096 flows. Since CPU polling and line-rate polling packet forwarding are asynchronous, the telemetry collection on multiple individual switches starts in parallel within an end-to-end delay ($\sim \mu s$). Therefore, the total time usage is approximately equal to the time of polling one switch, showing high scalability with network scale.

We then evaluate the efficiency of the CPU poller. Compared to fully dumping the telemetry using packet generation in the data plane, the CPU can reduce the telemetry data size reported to the analyzer significantly by excluding the useless telemetry data such as zero values. As shown in Figure 14(a), in most cases, the concurrent flow count in one epoch is much smaller than the maximum flow count in data plane telemetry, thus the telemetry size can be reduced by >80%. Besides, due to the limited PHV size (around 200 bytes), exporting the complete flow telemetry on the data plane requires a significant number of packets and incurs a heavy communication overhead. However, the CPU can batch the meaningful subset of the telemetry into a much larger packet (e.g., the MTU of 1500 bytes), reducing the number of packets considerably. As shown in Figure 14(b), the telemetry-reporting packet count can be reduced by about 95% in most cases.

5 Discussion

Operating scenarios of HAWKEYE. Network operators can utilize HAWKEYE in different ways by setting different triggering conditions. For example, the monitored performance metrics can be extended to various definitions, such as flow throughput, co-flow completion time, or even application-level metrics such as job completion time or algorithmic bandwidth. Besides on-demand diagnosis, when integrated with pingmesh-like probes [41], HAWKEYE can carry out periodic diagnosis for a more complete understanding of the network. Furthermore, in multi-tenant networks, anomaly diagnosis can be provided as a service to users by sharing client performance information with operators.

Representativeness of covered anomalies. In this paper, we propose a diagnosis method for a range of representative anomalies. Although we do not claim that we cover the complete NPA space in RDMA networks, the anomaly cases addressed in this paper are sufficiently representative. Specifically, the proposed anomalies cases (shown in Table 2) cover the existence of PFC, the different PFC root causes (flow contention or host PFC injection) and the shape of PFC spreading paths (whether a loop exists). We leave more comprehensive NPA coverage and a more general diagnosis algorithm as our future work.

Extensibility of Hawkeye telemetry. Our main focus in telemetry design is to enable PFC visibility and causality tracing, which is orthogonal to flow-level information recording. Therefore, the flow-level telemetry in Hawkeye can be further complemented by numerous existing studies [22, 37, 47, 58] to provide more information such as flow features and queue evolution, enabling a more in-depth understanding of flow interactions. However, integrating fine-grained flow telemetry also incurs higher overhead especially on switch resource usage.

Applicability of Hawkeye. Our Tofino implementation serves as a readily available proof-of-concept. However, Hawkeye is not tightly tied to Tofino, and is inexpensive to implement into next-generation switching ASICs. The real time PFC status of each port is already supported by many RDMA switches. Hawkeye only requires two additional components. First, the PFC causality data, including the causality structure (Figure 3) and port-level telemetry, requires additional registers with constant size, which is bounded by the number of switch ports. Second, Hawkeye enhances the flow-level telemetry with PFC-aware data, such as PFC impact on each flow. Such modification only requires a new data field in the telemetry recording system. As current switches are becoming more plentiful in resource and embracing greater programmability, especially considering the trend of white-box customized switches, we believe that Hawkeye is fully implementable to next-generation commodity switches.

Partial Deployment of Hawkeye. Hawkeye contains two core components: 1) the PFC causality analysis mechanism to trace the PFC spreading path; 2) the PFC-aware flow telemetry to analyze queue contention between flows and PFC impact on each flow, both of which are crucial to locate the root causes. If only a part of switches deploy Hawkeye, such as ToR switches, the PFC tracing will be interrupted at a non-Hawkeye switch, making it difficult to correlate victim flows with root causes even though a certain switch's telemetry may have recorded them. We give a feasible partial deployment option. Assume that all switches support PFC causality analysis, but we only deploy flow telemetry on some *hot-spot* switches, e.g., ToR switches, where a lot of flow contention such as incast frequently occurs. In this case, the PFC spreading causality can be completely traced, and a significant percentage of the root causes located in ToR switches can also be covered. However, diagnosis effectiveness is still inevitably compromised. For instance, some root causes like ECMP imbalance [66] or routing problems [27, 67] occurring on leaf/spine switches, are consequently uncovered.

Parameter setting on different network environment. Hawkeye empirically adjusts the parameters for different network workloads, to achieve a good trade-off between accuracy and overhead. The major parameters in Hawkeye, as mentioned in §4.2, is the epoch size and detection threshold. A shorter epoch size can capture flow interactions more accurately, helping to identify flow contention and major contributors. However, too short epoch size requires much more epochs to be maintained at the switch so that the telemetry will not be flushed quickly, causing higher memory usage and telemetry size. For detection threshold, taking latency as an example metric, it is important to consider the network's scale and application requirements. Typically, for latency-sensitive applications, the detection threshold is usually 2-4 times the maximum RTT (determined by the maximum hop count). A sensitive detection threshold can quickly trigger diagnosis, while causes higher diagnosis frequency.

6 Related Work

Besides the most relevant works we discussed early, our work is also inspired by the following topics.

Performance anomaly in intra-host networks. Many studies observe the intra-host network is becoming an RDMA performance bottleneck. Collie [35] explores the anomaly space to find performance anomalies in intra-host networks. Husky [34] presents a test suite to evaluate performance isolation in RNICs. Hostping [42]diagnoses the performance bottleneck in intra-host networks using loopback traffic. Lumina [71] proposes a test tool to evaluate the performance of RNIC network stacks. However, there are relatively fewer studies effectively diagnosing inter-host RDMA NPAs, as analyzed in §2.3, and we bridges this gap via Hawkeye.

Programmable network. Programmable networks, represented by programmable switches and SmartNICs, have been widely used to accelerate various applications in network monitoring [22, 39, 47, 79], traffic analysis [58, 64, 73, 77], queue measurement [10, 37], traffic control [3, 25, 38, 75], and security [33, 43, 74]. Inspired by them, Hawkeye leverages programmable switches to support PFC-aware telemetry logging, PFC causality tracing and telemetry collection.

Provenance-based diagnosis. There is a body of studies that leverage provenance to trace event causality relationship [8] and diagnose security threats or performance issues. ForenGuard [61] monitors the provenance of runtime activities in SDN. Unicorn [23] leverages the provenance of system executions to detect APT (Advanced Persistent Threat). SpiderMon [66] and PrintQueue [37] construct flow-level and packet-level provenance, respectively, to identify queue contention contributors. DTaP [78] and Zeno [68] construct time-aware event provenance and diagnose performance issues in distributed systems. Hawkeye takes PFC causality into provenance consideration to diagnose RDMA NPAs.

7 Conclusion

RDMA network performance anomalies present new complexities due to the cascading congestion of PFC and make existing solutions less effective. We propose Hawkeye to bridge the gap. It provides fine-grained PFC visibility through a multi-granularity PFC-aware telemetry system, a fast in-network PFC causality tracing mechanism to efficiently collect causal telemetry, and a precise diagnosis algorithm based on the heterogeneous provenance graph. Our evaluations demonstrate Hawkeye can accurately detect and diagnose

RDMA network performance anomalies with low overhead. We hope Hawkeye can inspire new paradigms for anomaly diagnosis in RDMA networks.

Acknowledgment

We thank our shepherd, Costin Raiciu, and the anonymous SIGCOMM reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (No. 62402025 and No.62221003), the Fundamental Research Funds for the Central Universities and gifts from Huawei-BUAA Joint Lab. Menghao Zhang and Zhiliang Wang are the corresponding authors.

References

[1] Alibaba. 2024. Alibaba builds high-speed RDMA network for AI and scientific computing. https://www.alibabacloud.com/blog/alibaba-builds-high-speed-rdma-network-for-ai-and-scientific-computing_594895. (2024).

[2] Alibaba-edu. 2019. HPCC simulation. https://github.com/alibaba-edu/High-Precision-Congestion-Control. (2019).

[3] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt: Sub-RTT Congestion Control for Ultra-Low Latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 219–236. https://www.usenix.org/conference/nsdi23/presentation/arslan

[4] 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)*. USENIX Association, Renton, WA, 419–435. https://www.usenix.org/conference/nsdi18/presentation/arzani

[5] Azure. 2023. High performance computing VM sizes. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-hpc/. (2023).

[6] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 49–67. https://www.usenix.org/conference/nsdi23/presentation/bai

[7] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 155–168. https://doi.org/10.1145/3098822.3098834

[8] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and Where: A Characterization of Data Provenance. In *Database Theory — ICDT 2001*, Jan Van den Bussche and Victor Vianu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 316–330.

[9] Jiamin Cao, Yu Guan, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2024. Crux: GPU-Efficient Communication Scheduling for Deep Learning Training. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3651890.3672239

[10] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. 2019. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 15–29. https://doi.org/10.1145/3359989.3365408

[11] Youmin Chen, Youyou Lu, and Jiwu Shu. 2019. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 19, 14 pages. https://doi.org/10.1145/3302424.3303968

[12] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. 2016. Fast and general distributed transactions using RDMA and HTM. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 26, 17 pages. https://doi.org/10.1145/2901318.2901349

[13] Wenxue Cheng, Kun Qian, Wanchun Jiang, Tong Zhang, and Fengyuan Ren. 2020. Re-architecting Congestion Management in Lossless Ethernet. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 19–36. https://www.usenix.org/conference/nsdi20/presentation/cheng

[14] Charles Clos. 1953. A study of non-blocking switching networks. *The Bell System Technical Journal* 32, 2 (1953), 406–424. https://doi.org/10.1002/j.1538-7305.1953.tb01433.x

[15] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 401–414. https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi

[16] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/3651890.3672233

[17] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 519–533. https://www.usenix.org/conference/nsdi21/presentation/gao

[18] Yixiao Gao, Yuchen Yang, Tian Chen, Jiaqi Zheng, Bing Mao, and Guihai Chen. 2018. DCQCN+: Taming Large-Scale Incast Congestion in RDMA over Ethernet Networks. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, Cambridge, United Kingdom, 110–120. https://doi.org/10.1109/ICNP.2018.00021

[19] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 61–74. https://doi.org/10.1145/3050220.3050228

[20] 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 (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 202–215. https://doi.org/10.1145/2934872.2934908

[21] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 139–152. https://doi.org/10.1145/2785956.2787496

[22] 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 (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 357–371. https://doi.org/10.1145/3230543.3230555

[23] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[24] 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)*. USENIX Association, Seattle, WA, 71–85.

[25] Jinbin Hu, Chaoliang Zeng, Zilong Wang, Junxue Zhang, Kun Guo, Hong Xu, Jiawei Huang, and Kai Chen. 2023. Enabling Load Balancing for Lossless Datacenters. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. 1–11. https://doi.org/10.1109/ICNP59255.2023.10355615

[26] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2016. Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 92–98. https://doi.org/10.1145/3005745.3005760

[27] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2019. Tagger: Practical PFC Deadlock Prevention in Data Center Networks. *IEEE/ACM Transactions on Networking* 27, 2 (2019), 889.

[28] IEEE. 2011. IEEE 802.1 Qbb - Priority-based Flow Control. https://1.ieee802.org/dcb/802-1qbb/. (2011).

[29] InfiniBand Trade Association. 2020. InfiniBand Architecture Specification Release 1.4 Annex A17: RoCEv2. (2020).

[30] 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)*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[31] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 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)*. USENIX Association, Santa Clara, CA, 745–760. https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng

[32] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 185–201. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kalia

[33] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. 2020. Programmable In-Network Security for Context-aware BYOD Policies. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 595–612. https://www.usenix.org/conference/usenixsecurity20/presentation/kang

[34] 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)*. USENIX Association, Boston, MA, 31–48. https://www.usenix.org/conference/nsdi23/presentation/kong

[35] 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)*. USENIX Association, Renton, WA, 287–305. https://www.usenix.org/conference/nsdi22/presentation/kong

[36] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[37] Yiran Lei, Liangcheng Yu, Vincent Liu, and Mingwei Xu. 2022. PrintQueue: performance diagnosis via queue measurement in the data plane. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 516–529. https://doi.org/10.1145/3544216.3544257

[38] Wenxue Li, Chaoliang Zeng, Jinbin Hu, and Kai Chen. 2023. Towards Fine-Grained and Practical Flow Control for Datacenter Networks. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. 1–11. https://doi.org/10.1109/ICNP59255.2023.10355582

[39] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 311–324. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-yuliang

[40] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58. https://doi.org/10.1145/3341302.3342085

[41] Kefei Liu, Zhuo Jiang, Jiao Zhang, Shixian Guo, Xuan Zhang, Yangyang Bai, Yongbin Dong, Feng Luo, Zhang Zhang, Lei Wang, Xiang Shi, Haohan Xu, Yang Bai, Dongyang Song, Haoran Wei, Bo Li, Yongchen Pan, Tian Pan, and Tao Huang. 2024. R-Pingmesh: A Service-Aware RoCE Network Monitoring and Diagnostic System. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 554–567. https://doi.org/10.1145/3651890.3672264

[42] 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)*. USENIX Association, Boston, MA, 15–29. https://www.usenix.org/conference/nsdi23/presentation/liu-kefei

[43] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3829–3846. https://www.usenix.org/conference/usenixsecurity21/presentation/liu-zaoxing

[44] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled Distributed Persistent Memory File System. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 773–785. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu

[45] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 537–550.

[46] 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 (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 129–143. https://doi.org/10.1145/2934872.2934879

[47] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/3098822.3098829

[48] NVIDIA. 2024. NVIDIA DOCA PCC Application Guide. https://docs.nvidia.com/doca/sdk/nvidia+doca+pcc+application+guide/index.html. (2024).

[49] NVIDIA. 2025. NVIDIA Spectrum-X Networking Platform. https://www.nvidia.com/en-us/networking/spectrumx/. (2025).

[50] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. 2019. Gentle flow control: avoiding deadlock in lossless networks. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 75–89. https://doi.org/10.1145/3341302.3342065

[51] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 691–706. https://doi.org/10.1145/3651890.3672265

[52] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3544216.3544226

[53] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1403–1420. https://www.usenix.org/conference/nsdi24/presentation/rajasekaran

[54] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/2785956.2787472

[55] Alex Shpiner, Eitan Zahavi, Vladimir Zdornov, Tal Anker, and Matty Kadosh. 2016. Unlocking Credit Loop Deadlocks. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 85–91. https://doi.org/10.1145/3005745.3005768

[56] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 708–721. https://doi.org/10.1145/3387514.3405897

[57] John Snyder, Alvin R Lebeck, and Danyang Zhuo. 2022. RDMA Congestion Control: It Is Only for the Compliant. *IEEE Micro* 43, 1 (2022), 76–82.

[58] John Sonchack, Oliver Michel, Adam J. Aviv, Eric Keller, and Jonathan M. Smith. 2018. Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 823–835. https://www.usenix.org/conference/atc18/presentation/sonchack

[59] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM*

'23). Association for Computing Machinery, New York, NY, USA, 816–831. https://doi.org/10.1145/3603269.3604849

[60] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2018. Distributed Network Monitoring and Debugging with SwitchPointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 453–456. https://www.usenix.org/conference/nsdi18/presentation/tammana

[61] Haopei Wang, Guangliang Yang, Phakpoom Chinprutthiwong, Lei Xu, Yangyong Zhang, and Guofei Gu. 2018. Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 3–16. https://doi.org/10.1145/3243734.3243749

[62] Shicheng Wang. 2025. Hawkeye. https://github.com/wangshicheng1225/Hawkeye. (2025).

[63] Shicheng Wang, Menghao Zhang, Yuying Du, Ziteng Chen, Zhiliang Wang, Mingwei Xu, Renjie Xie, and Jiahai Yang. 2024. LoRDMA: A New Low-Rate DoS Attack in RDMA Networks. In *NDSS 2024*. San Diego, CA.

[64] Shicheng Wang, Menghao Zhang, Guanyu Li, Chang Liu, Ying Liu, Xuya Jia, and Mingwei Xu. 2021. Making Multi-String Pattern Matching Scalable and Cost-Efficient with Programmable Switching ASICs. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. IEEE Press, Virtual Event, 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488796

[65] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 739–767. https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang

[66] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and T. S. Eugene Ng. 2022. Closed-loop Network Performance Monitoring and Diagnosis with SpiderMon. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 267–285. https://www.usenix.org/conference/nsdi22/presentation/wang-weitao-spidermon

[67] Xinyu Crystal Wu and T. S. Eugene Ng. 2022. Detecting and Resolving PFC Deadlocks with ITSY Entirely in the Data Plane. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE Press, Virtual Event, 1928–1937. https://doi.org/10.1109/INFOCOM48880.2022.9796798

[68] Yang Wu, Ang Chen, and Linh Thi Xuan Phan. 2019. Zeno: Diagnosing Performance Problems with Temporal Provenance. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 395–420. https://www.usenix.org/conference/nsdi19/presentation/wu

[69] 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 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 44, 14 pages. https://doi.org/10.1145/3302424.3303975

[70] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 221–234. https://www.usenix.org/conference/fast19/presentation/yang

[71] 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 (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 1074–1087. https://doi.org/10.1145/3603269.3604837

[72] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 139–150. https://doi.org/10.1145/2342356.2342390

[73] Menghao Zhang, Guanyu Li, Cheng Guo, Renyu Yang, Shicheng Wang, Han Bao, Xiao Li, Mingwei Xu, Tianyu Wo, and Chunming Hu. 2025. SuperFE: A Scalable and Flexible Feature Extractor for ML-based Traffic Analysis Applications. In *EuroSys 2025*. ACM, Rotterdam.

[74] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*.

[75] Yiran Zhang, Yifan Liu, Qingkai Meng, and Fengyuan Ren. 2021. Congestion detection in lossless networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 370–383. https://doi.org/10.1145/3452296.3472899

[76] Fang Zhou, Yifan Gan, Sixiang Ma, and Yang Wang. 2018. wPerf: Generic Off-CPU Analysis to Identify Bottleneck Waiting Events. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 527–543. https://www.usenix.org/conference/osdi18/presentation/zhou

[77] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. 2023. An Efficient Design of Intelligent Network Data Plane. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 6203–6220. https://www.usenix.org/conference/usenixsecurity23/presentation/zhou-guangmeng

[78] Wenchao Zhou, Suyog Mapara, Yiqing Ren, Yang Li, Andreas Haeberlen, Zachary Ives, Boon Thau Loo, and Micah Sherr. 2012. Distributed time-aware provenance. *VLDB Endowment* 6, 2 (2012), 49–60.

[79] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, Pengcheng Zhang, Dennis Cai, Ming Zhang, and Mingwei Xu. 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 (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 76–89. https://doi.org/10.1145/3387514.3406214

[80] 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. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484