MARS: Fault Localization in Programmable Networking Systems with Low-cost In-Band Network Telemetry

Benran Wang, Hongyang Chen, Pengfei Chen*, Zilong He, Guangba Yu
{wangbr5,chenhy95,hezlong,yugb5}@mail2.sysu.edu.cn,chenpf7@mail.sysu.edu.cn*
School of Computer Science and Engineering, Sun Yat-sen University
China

ABSTRACT

Recently, the adoption of Software Defined Networking (SDN) as a network infrastructure has gained significant popularity. Although the openness and programmability of SDN ease the construction of large complex networks, it is still challenging to diagnose faults in a complex datacenter-scale network, which is crucial to guarantee rigorous service level agreement (SLA) of upper-layer applications. Previous network diagnosis tools incur significant overhead in fine-grained telemetry, and usually lack the ability to automatically diagnose fine-grained faults. Although on-demand monitoring methods is proposed to reduce telemetry overhead, they struggle to effectively set static thresholds, which requires expert experience. In this paper, we present MARS, a lightweight system for anomaly detection with dynamic threshold and automatic root cause localization in programmable networking systems. MARS collects aggregated packet-level telemetry on demand and generates a ranked list of fine-grained fault culprits at multiple levels, including port-level, switch-level, and flow-level. Experimental evaluations show the cost-effectiveness of MARS, both in terms of network bandwidth and switch memory usage. Moreover, MARS achieves a 0.97 F1 score in anomaly detection, and 0.95 Recall at *Top-2* and an overall 0.3 Exam Score in root cause localization.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network performance analysis**; • **Computer systems organization** → **Maintainability and maintenance**.

KEYWORDS

P4, In-band Network Telemetry, Fault Localization, Software Defined Network

ACM Reference Format:
Benran Wang, Hongyang Chen, Pengfei Chen*, Zilong He, Guangba Yu. 2023. MARS: Fault Localization in Programmable Networking Systems with Low-cost In-Band Network Telemetry. In *52nd International Conference on Parallel Processing (ICPP 2023), August 7–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605622

1 INTRODUCTION

With the demand for applications with low latency and high throughput, a growing number of IT enterprises choose to deploy their applications, particularly latency-critical applications, in datacenters. Since applications always dynamically scale to deliver a high quality of experience (i.e., QoE), it is of great importance to efficiently manage the datacenter network to ensure that the network can scale to meet the evolving demands of applications while reducing the complexity in network management.

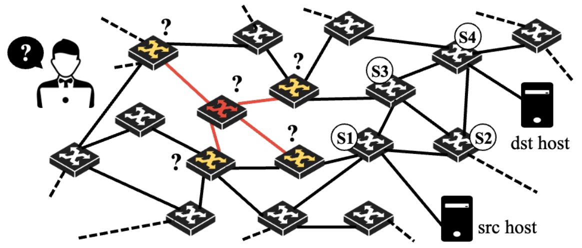

Figure 1: A network fault happens in a large-scale network.

Recently, the adoption of SDN and programmable switch technology has facilitated the management of the underlying network infrastructure in datacenters, leading to a significant increase in the scale and complexity of the network. The booming scale of networks renders network failures as norm cases rather than exceptions. It is critical to diagnose these failures in time, as even a minor degradation in network performance can have a significant impact on the quality of upper-layer applications [15, 34, 53]. Fig. 1 shows a complex datacenter network where a network fault occurred in a switch. The network fault causes multiple neighbour switches to behave abnormally simultaneously. Therefore, diagnosing network failures manually is time-consuming and error-prone. A system designed for network monitoring and automatic failure diagnosis is necessary.

This paper focuses on the problem of automatically localizing the root cause of network faults in a complex network with a low monitoring overhead. Although programmable switches have inspired extensive researches [9, 14, 22, 30, 44, 54] on this problem, the effectiveness of these work remains inadequate due to the following limitations.

- **The overhead of monitoring is high, downgrading the performance of datacenter network.** Existing monitoring methods either use packet mirroring at switches to monitor the network [21, 40, 46, 54], or actively injecting probe packets into the network [4, 7, 51] to estimate the network status. While these methods provide valuable information about the network, they also result in additional network traffic. The additional overhead has a negative impact on network performance.
- **Existing anomaly detection methods are inefficient and unscalable.** To conserve network bandwidth and reduce network overhead, trigger-based methods [25, 26, 47, 53] have been proposed. Existing anomaly detection methods rely on the trigger-based methods to conduct anomaly detection. These methods send monitoring data collected in the data plane to the control plane only when anomalies (e.g., high end-to-end latency and queuing delay) are detected

with static thresholds. However, since the number of network flows is always huge, setting thresholds for each flow is time-consuming and requires substantial expert knowledge, hindering the adoption of trigger-based methods in large-scale datacenter networks.

- **Existing network failure diagnosis methods cannot well cover multiple causes of network fault automatically, since they struggle to extract clues from massive network flows.** Jia et.al. [22] rely on the set intersection of time-out paths to locate the position of packet loss, which is not robust and may fail if an anomaly lasts shorter than the time window. IntSight [31] delegates diagnosis to the data plane but has limitation in analyzing network faults that may spread to adjacent switches. Though query-based debugging [26] collects extensive data, it requires massive expert experience, which is time-consuming and error-prone.

To address these limitations, we propose MARS, a holistic system comprising of **M**onitoring, **A**nomaly detection, **R**oot cause analysis in **S**DN. MARS is a system based on P4 [11] that integrates low-cost telemetry and posterior fault localization with high accuracy. It collects telemetry metadata (e.g., path sequence from path id, so-called "path-aware") carried by packets and temporarily stores them in edge switches. The telemetry data for diagnosis is sent to the control plane only when the data plane detects an anomaly. The on-demand collection decreases additional network bandwidth consumption (**Limitation 1**). The system processes streaming latency data for each flow with a modified reservoir module and updates thresholds dynamically for detecting network delay (**Limitation 2**). The Frequent Sequence Mining (FSM) [23] utilizes path information from the path-aware telemetry data to identify suspicious positions within the network. With scores calculated by Spectrum-Based Fault Localization (SBFL) [5] and telemetry data such as throughput and queuing conditions, MARS can accurately determine the port/switch/flow-level root cause with a 0.95 recall at *Top-2* and an overall 0.3 Exam Score (§5). This ranked list of culprits serves to expedite problem resolution for operators (**Limitation 3**).

In summary, we make the following contributions.

- **On-demand Monitoring.** We propose a novel path-aware method to monitor network traffic and report data on demand. This method is independent of the length of the path and does not raise extra costs as the network becomes larger.
- **Self-adaptive Anomaly Detection.** We propose self-adaptive in-network anomaly detection, where the data plane accurately detects anomalies in a versatile network. This is achieved by updating the threshold using a reservoir model in the control plane.
- **Automatic Root Cause Analysis.** We combine Frequent Sequence Mining and Spectrum-Based Fault Localization to precisely locate the faulty switch. The root cause can be automatically localized at different levels in a rank list.
- In the consideration of five different fault scenarios, we prove the effectiveness and low overhead of MARS.

2 BACKGROUND

SDN & PDP: Software Defined Network (SDN) and PDP (Programming Data Plane) are widely adopted in modern large-scale web systems such as Google and Facebook recently due to their openness and programmability. SDN separates the control plane (decision-making) from the data plane (forwarding of packets) in a network to enable centralization of network management and programmability. PDP refers to the programmable aspect of the data plane in SDN, allowing network administrators to program switch with customized processing logic.

OpenFlow & P4: OpenFlow [32], as a communication protocol used in SDN, abstracts traditional network lookup tables and forwarding into general flow tables and match-action list. P4 (Programming Protocol-independent Packet Processors) extends OpenFlow a step further by achieving protocol independence. P4 is a high-level programming language and provides an abstract representation of packet processing pipelines, allowing network engineers to define how packets should be processed within a network without having to deal with low-level hardware details.

In-band Network Telemetry (INT): Proposed by the P4 organization, INT [38] is a technique for monitoring and collecting network data within the data plane. By inserting or modifying fields of the packet header, INT can gather telemetry data in real-time, without the need for external probes or monitoring agents. INT enables network administrators to gather fine-grained and up-to-date information about network performance and behavior, and helps to troubleshoot and diagnose network issues more efficiently.

3 MOTIVATION

Motivation #1: Offloading the Burden to Edge Switches. To further reduce the overhead caused by monitoring, SpiderMon [47] saves network bandwidth by collecting telemetry to the control plane on demand. It requires all switches to store data and report diagnosis data to the control plane. However, regardless of the type of network, core switches are always busier than edge switches, as the possibility of low link utilization in the core layer is lower than in the edge layer [10], which is demonstrated by the Cumulative Distribution Function (CDF) in Fig. 2. To relieve the strain on core switches, MARS is expected to gather telemetry data on edge switches and maintain the coverage of telemetry. As a result, we are inspired to collect aggregated data at the packet level and record only in edge switches. In this case, the fault localization is expected to perform well without end-to-end data, which is realized with FSM and SBFL.

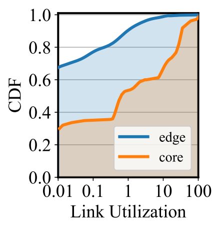

Figure 2: The link utilization of the core layer versus the edge layer.

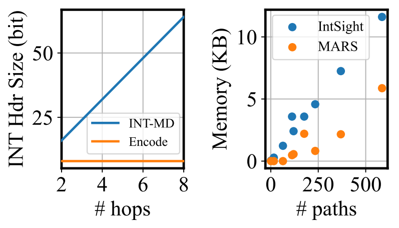

Figure 3: Encoding path with ID can reduce INT header size. Among the methods that encode path, MARS spares more switch memory.

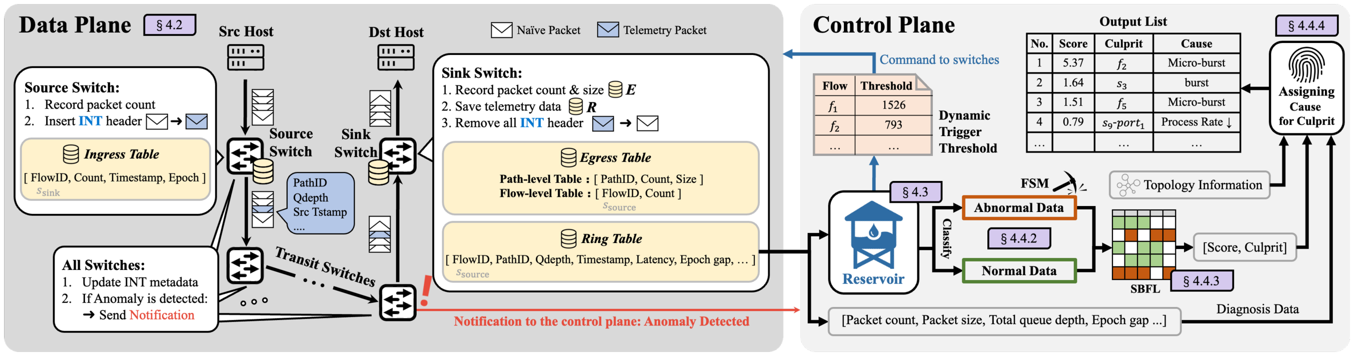

Figure 4: Overview of MARS. The left part of figure depicts how MARS collects and stores telemetry data. The right part shows the workflow of MARS to update dynamic threshold for anomaly detection in the data plane and to locate the root causes, resulting in a ranked list with culprits and causes.

Motivation #2: Reducing Header Size for Path Recording. The packet transmission path plays a critical role in the root cause diagnosis. To record the switch sequence, the INT-MD (eMbed Data) mode, as described in [38], is used by methods such as [22, 38] to inject telemetry data into the packet header at each hop. However, this method leads to an increase in packet size as the length of the transmission path grows, as demonstrated in the left graph of Fig. 3. IntSight [31] encodes the path (sequences of switches) to an ID with a fixed width, reducing the header size. However, IntSight needs to assign many Match-Action Table (MAT) entries of the path id for each path, which consumes switches' memory a lot, as shown in the right graph of Fig. 3. As a result, we are inspired to propose a novel path-aware method, relieving the monitoring overhead in a larger network without consuming too much memory of switches.

4 DESIGN AND IMPLEMENTATION

In this section, we present the design of MARS to monitor, identify, and diagnose abnormal events. The overview is shown in Fig. 4. For **monitoring (§4.2)**, MARS periodically samples network packets as telemetry packets and inserts critical information into telemetry headers. Traffic telemetry data is stored in edge switches' memory and reported on demand. For **anomaly detection (§4.3)**, reservoirs are maintained to update a dynamic threshold for each flow. On each switch, once the current packet's latency is greater than the corresponding flow's latency threshold or a packet loss event occurs, the switch will send a notification packet to the control plane. Triggered by the notification, the control plane will collect the telemetry data stored in edge switches for diagnosis. For **root cause analysis (§4.4)**, the culprits are located by FSM and scored by SBFL. Other telemetry data (e.g., queue depth, packet size) help find out causes for diverse faults at different corresponding levels, e.g., port-level for packet loss, switch-level for Equal Cost Multi-Path routing (ECMP) load imbalance, and flow-level for micro-burst flows.

4.1 Definition

Definition (Source/Transit/Sink Switch). *As shown in Fig. 4, for a packet traversing in the network, the switch that packets first enter is referred to as the* **source switch**. *The switch that the packet last exits from, before reaching its destination host, is called* **sink switch**. *Other switches between source switch and sink switch are named as* **transit switches**. *Note that one switch may play multiple roles (source, transit, sink) for different flows.*

Definition (FlowID). FlowID *is defined as* $\langle s_{\text{source}}, s_{\text{sink}} \rangle$ *without host information, as MARS focuses on the problems that happen in the network, i.e., the anomaly between/in switches. Furthermore,* FlowID *spares more bits than 5-tuple.*

Definition (PathID). PathID *is updated per hop as the packet traverses across switches. At each hop, the updated* PathID *is hashed by* **{PathID, switchID, ingress port, egress port, control}**. *Here the* control *field is set to zero by default unless the hashed value has conflicts with another flow. As a result, MARS needs to install MAT entries only when a hash conflict happens, which decreases the memory usage of switches.*

For instance, in Fig. 1, $\langle s_1, s_4 \rangle$ is a FlowID, with two paths, i.e., $\langle s_1, s_3, s_4 \rangle$ and $\langle s_1, s_2, s_4 \rangle$. Due to the complexity and variability of the network situation, like some load balance strategies, the exact path of a packet is decided on the fly only in the network. Therefore, the final PathID can only be known at the sink switch. The switch memory used for PathID calculation is discussed in §5.5. The control plane calculates each PathID with the same hash algorithm in advance, and saves the map from a PathID to the corresponding switch sequence, i.e., path. Based on the consensus between the control plane and the data plane about the PathID, the control plane can decompress fixed size bits field to path information from the data plane's report in later fault localization analysis.

4.2 Telemetry Collection

As network behavior would not shift dramatically in a short time under normal circumstances [44], MARS applies a sample strategy to collect necessary information without causing large bandwidth overhead. The source switch periodically sets a sample packet to gather telemetry data, including statistics data such as packet counts in the sample period.

4.2.1 ***Packet Category***. MARS uses the reserved field in the IP layer (e.g. "option" field in IPv4 Option) to distinguish different types of a packet (naïve packet and telemetry packet).

Naïve Packet. Naïve packet is a packet without a telemetry header inserted by the switch. For all naïve packets, only a field of small size (e.g., 8 bits) will be inserted into the header to record

PathID, which will be carried and updated through the network. For all packets, the total packet number and total packet size in each epoch are counted at both source switches and sink switches.

Telemetry Packet. In each epoch, the source switch adds a telemetry header into naïve packet for each flow, without modifying other header fields and payload. The epoch period can be set by the controller at runtime. The resulting packet with the telemetry header is referred to as a telemetry packet. Note that telemetry packets of different flows are not marked simultaneously, which can mitigate the transient burden of telemetry data on the network. The telemetry packet carries 11 bytes of telemetry data, including the timestamp that the packet enters the source switch (source timestamp), packet count of the packet's FlowID in the last epoch, total queue depth, and telemetry epoch ID. Here the timestamp can be compressed into a smaller size [47]. The total queue depth will be updated at each hop by in-network computing, i.e., adding up each switch's queue depth.

4.2.2 ***Switch Actions.***

The job of switches involves not only packet monitoring, but also anomaly detection. All switches update PathID for both naïve packets and telemetry packets. Specifically for telemetry packets, switches update the telemetry data, such as the total queue depth in telemetry headers. Anomaly detection is deployed at every switch, so that MARS can respond to anomalies in a more timely manner. Switches send a notification packet to the control plane when a telemetry packet's latency is higher than its corresponding dynamic threshold or a drop event is detected (§4.3). Once a packet triggered anomaly detection, the switch would update a flag in the header to suppress anomaly detection in subsequent hops and to prevent redundant notification to the control plane. On the other hand, switches are only allowed to inform the control plane one time in a time window. This can save the switch resource and network bandwidth.

To prepare and record telemetry data for diagnosis, edge switches (source switches and sink switches) have more works to do.

Source Switch inserts and records telemetry data in following two steps. **(1)** The source switch counts the number of incoming packets of each flow per epoch and records them in the Ingress Table (IT). **(2)** For all incoming packets, the source switch inserts a field to record the PathID. Besides, the source switch modifies the reserved field to distinguish naïve packet and telemetry packet. To determine which packet category a packet belongs to, IT records the timestamp and epoch ID of the latest telemetry packet per-flow, enabling only one telemetry packet to be set per flow in each epoch.

Sink Switches stores the telemetry data in following steps. **(1)** For all kinds of incoming packets, the sink switch records the packet count and packet size corresponding to PathID and FlowID in the Egress Table (ET). **(2)** For telemetry packets, the sink switch extracts the telemetry data to the Ring Table (RT). In addition, RT records the latency, packet counts, and packet size at path-level and flow-level in this epoch. When RT is full, the oldest data will be covered by the newest data, that is why the table is called as "ring". As a result, RT keeps the most recently telemetry. **(3)** All INT headers will be removed at the end of the sink switch, ensuring the monitoring is transparent to end hosts.

Note that node ID of source switch and sink switch already covers half information of FlowID ($\langle s_{\text{source}}, s_{\text{sink}}\rangle$). Therefore, the FlowID can be simplify as s_{sink} in source switch and s_{source} in sink switch, saving the memory of switch.

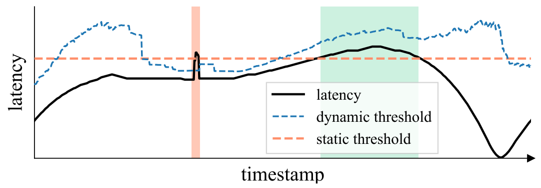

Figure 5: Example of anomaly detection with dynamic and static thresholds in the face of dynamic loads.

4.3 Anomaly Detection

4.3.1 ***High Latency Detection.***

Setting latency thresholds for flows requires expert knowledge, which is not suitable for complex and rapidly changing SDN. As shown in Fig. 5, the network traffic volume varies throughout the day. To successfully detect anomalies (i.e., spike in orange shadow), a static threshold (i.e., orange dashed horizontal line) may cause false positive (i.e., in green shadow). Whereas the dynamic threshold can effectively detect the spike without false alarms. To address this problem, MARS applies a robust unsupervised classifier to distinguish abnormal packets by their end-to-end latency.

Reservoir. To balance the accuracy and memory space, MARS applies a flexible reservoir to maintain the latency of each flow (details are shown in Alg. 1). MARS periodically gathers the value of "latency" field in RT from the data plane with P4 Runtime API [3] to get recent latency data, and feeds the data in the reservoir to update the dynamic thresholds. Classically, each input data will overwrite one randomly selected item in reservoir with the probability p when the reservoir is full [20]. A small number of latency outliers can impact the average, but the median value keeps more stable, the threshold θ to detect outliers is calculated by the function of median m and standard deviation σ, such as $\theta = m + 3\sigma$. However, if many outliers (high latency) are appended into the reservoir with the static probability, the standard deviation will vary a lot even though the median can stay firm. As a result, the threshold may increase due to outliers. To prevent this situation, MARS introduces a penalty factor α. When latency data are fed into the reservoir, the data will be judged whether it is an outlier. The number of consecutively detecting as outliers is set to c_o. The penalty factor is defined as a non-linear function, like $\alpha = \exp(-c_o)$. Therefore, as more continuous outliers are detected, the possibility that incoming data gets into the reservoir decreases severely. When encountering a new unknown flow, switches temporarily use a default threshold to detect anomalies. The default threshold is set at a relatively high level (e.g., 10 seconds) to minimize false positives, and will later be replaced by a dynamic threshold from the reservoir.

4.3.2 ***Drop Event Detection.***

If a packet loss event happens within a telemetry epoch, the packet count in the source switch (c_s) and in the sink switch (c_d) will differ, namely $c_s - c_d > 0$. The difference between the two counts indicates the number of dropped packets. If the difference is greater than a threshold, the switch would inform the control plane. On the other hand, if the drop

Algorithm 1 Reservoir Anomaly Detection

```
Input: Reservoir volume $v \in \mathbb{N}$, static probability $p_s$, constant $C$
Output: Outlier flag $flag$
1: Reservoir $R \leftarrow \{\}$; outlier count $c_o \leftarrow 0$;
2: function INPUT($l$: latency data)
3:     if $l > m(R) + C \cdot \sigma(\mathrm{R})$ then        ▷ whether it is an outlier
4:         $c_o \leftarrow 0$
5:         $flag \leftarrow$ TRUE
6:     else
7:         $c_o \leftarrow c_o + 1$
8:         $flag \leftarrow$ FALSE
9:     end if
10:    $\alpha \leftarrow \exp(-c_o)$        ▷ penalty factor
11:    if $|R| < v$ then        ▷ update Reservoir
12:        $R \leftarrow R \cup \{l\}$
13:    else with probability $\alpha \cdot p_s$
14:        overwrite a randomly selected item from $R$ with $l$
15:    end if
16: end function
```

event lasts several epochs, the telemetry packet of these epochs would be dropped as well. Like the sequence number of TCP, the source switch compares whether the epoch id carried by telemetry is neighbored with the last epoch id of the received telemetry packet of this flow. If the epoch ids are not adjacent, a drop event is detected and the control plane would be noticed by the switch. The difference between two epoch ids indicates how long the drop event lasted, which is revealed from the field "epoch gap" in RT.

4.4 Root Cause Analysis

The control plane, triggered by a notification from the data plane, collects recent telemetry data as *diagnosis data* from the memory (Ring Table) of sink switches. To avoid massive notifications, each switch is limited to sending only one notification in a time window. Similarly, the control plane is also limited to responding to notifications from different switches in a time window. MARS uses this self-contained telemetry data to identify the root cause. The reservoir categorizes packets into two groups, namely the abnormal set and the normal set. The path of a packet is a sequence of switches it travels through, and if a switch or link frequently appears in the abnormal set but rarely appears in the normal set, it is considered to be a likely cause of failure. This empirical observation is utilized in the fault localization approach of MARS. For instance, sub-sequences of a packet path $\langle s_1, s_2, s_3, s_4\rangle$, such as $\langle s_2\rangle$ or $\langle s_3, s_4\rangle$, is regarded as single switches or links respectively. Sub-sequences consisting of more than two elements are not considered meaningful.

The root cause analysis is composed of four parts. **(1)** MARS estimates actual traffic from sample data (§4.4.1) and classifies them into an abnormal set and a normal set. **(2)** From the abnormal set, MARS mines frequent sequence patterns with FSM as suspect locations where the anomaly happened, called culprits (§4.4.2). **(3)** According to both the abnormal set and normal set, MARS integrates the risk ratio into SBFL to calculate a score for each culprit, which is high if the culprit (pattern) frequently appears in the abnormal set but rarely appears in the normal set (§4.4.3). **(4)** With diagnosis data, MARS assigns a cause for each culprit according to signature matching and calculates a score of causes based on the culprits' score. At last, MARS merges repeated causes and sends an ordered list of culprits with causes to network operators (§4.4.4).

4.4.1 *Actual Traffic Estimation.*

The number of packets in different sample epochs may vary. MARS uses the gap-based sampling strategy [28] to restore the arrive time distribution of real flow. With packet counts *p.count* of each PathID in each epoch, MARS estimates the packets' arrive time $\hat{p}.t$ (details are shown in Alg. 2).

Algorithm 2 Actual Traffic Estimation

```
Input: Sample packets $S$, Time gap between sample pkts $T$
Output: Estimated packets $E$
1: for PathID in $S$ do
2:     for $p \in S_{\mathrm{PathID}}$ do        ▷ per pkt from samples of PathID
3:         for $i \in$ range($p.count$) do
4:             $\hat{p} \leftarrow$ COPY($p$)        ▷ copy sample as an estimation
5:             $\hat{p}.t \leftarrow p.t + \frac{i \times T}{p.count}$        ▷ estimate pkt's timestamp
6:             $E \leftarrow E \cup \{\hat{p}\}$
7:         end for
8:     end for
9: end for
```

4.4.2 *Frequent Sequence Mining (FSM).*

In order to localize the culprit, MARS needs to find out the most frequent sub-sequences in the abnormal set. However, a path that has L switches can have $\frac{1}{2}L(L+1)$ sub-sequences. Naïvely calculating all sub-sequences of all paths is inefficient and time-wasting, even though there are duplicate sub-sequences of different paths. To efficiently find out the most frequently occurring sub-sequence in the abnormal set, Frequent Sequence Mining (FSM) is a prominent solution. With the method of depth-first search or pattern-growth, FSM prunes sequences that are not frequent as early as possible to speed up the algorithm and return frequent sequence patterns.In our evaluation (§5.5), PrefixSpan[23] performs the best, using the minimum running time and relative less memory. FSM algorithms process a list of sequences and output frequent sequence patterns with *support*. The *support* of sequence s_a is defined as the number of sequences that contain s_a. In MARS, the frequent sequence pattern is switch or link (two switches).

For example, suppose control plane receives data with four $path_1 = \langle s_3, s_2, s_4\rangle$ and two $path_2 = \langle s_6, s_2, s_7\rangle$. If the configuration of FSM sets the max pattern length to be 2 and the min relative support to be 50%, i.e., min $support = (4+2)/2 = 3$ in this case. The frequent sequence patterns result in ($\langle s_2\rangle$, $\langle s_2, s_4\rangle$, $\langle s_3\rangle$, $\langle s_3, s_2\rangle$, $\langle s_4\rangle$). Here the pattern with the highest *support* is $\langle s_2\rangle$, whose $support = 6$, and other patterns' *support* are 4. Other patterns like $\langle s_6\rangle$ are pruned because their *support* is lower than the min support.

4.4.3 *Spectrum-based Fault Localization (SBFL).*

After finding out frequent sequences as culprits, MARS needs to sort them in order to filter out *Top-N* culprits. As an approach towards program fault localization, SBFL utilizes various program spectra from software tests at code level or feature level, and corresponding test results to calculate each test case's suspicious score [5]. MARS extends SBFL from the software domain to the network domain, ranking each culprit sequences pattern by comparing the proportion of normal/abnormal data sets with and without the pattern.

Relative risk is a statistical analysis technique in medical studies [35]. MARS integrates relative risk into SBFL to calculate the suspicious score of each pattern as

$$Score(pattern) = \frac{N_{pf}/(N_{pf}+N_{ps})}{N_{nf}/(N_{nf}+N_{ns})}, \tag{1}$$

where N_{pf} is the number of packets in abnormal set (**f**ailing test) whose path contains the specific **p**attern, N_{ps} is the number of packets in normal set (**s**uccessful test) whose path contains the specific **p**attern, N_{nf} and N_{ns} is the number of packets whose path does **n**ot contain the specific *pattern* in **f**ailing test and **s**uccessful test. To avoid the erro of division by zero, N_{nf} can be considered as adding a constant, normally equal to 1, when the processing data set is too small and all abnormal data share a same pattern. In this case, the equation variation can be written as $(N_{pf}/(N_{pf}+N_{ps}))/((N_{nf}+1)/(N_{nf}+N_{ns}))$. The numerator of (1) is the abnormal proportion of this *pattern*, and the denominator stands for the abnormal proportion of other patterns. A higher score value indicates a higher possibility that the pattern is the source of failure, as compared to other patterns.

4.4.4 *Culprit localization.*

The root cause of why a node/link behaves abnormally can vary. To give more comprehensive help to network operators, MARS relies on signature matching to assign a cause for each culprit pattern with telemetry data. By querying the paths that passed through the frequent patterns in the diagnosis data, MARS assigns causes for culprits according to packet counts and packet size at edge switches (to calculate throughput), total queue depth in the whole path, and the topology information. Each culprit will earn a score that is the pattern score multiplied by the corresponding proportion of the path, as shown in Alg. 3.

We give five signatures for five common root causes at flow level, switch level, and port level. The signatures can be extended if more root causes are considered.

- **Micro-burst** is a flow-level cause. It is a short-lived spike of flow that exceeds average traffic, leading to queue buildup and resulting in a high latency or packet loss [10, 12]. Its signature is whether a flow's pps (packets per seconds) raises sharply in the problematic period.
- **ECMP Load Imbalance** is a switch-level cause. The paths between edge switches are usually not single but multiple. For instance, there are four equal paths from s_{11} to s_{15} in (Fig. 6). Ideally, each hop can assign the traffic equally for load balance (e.g., s_9 sends equally traffic to s_1 and s_2). However, imperfect hash algorithm and uneven flow distribution under time and space may bring about ECMP load imbalance [37]. If s_9 fails to balance the traffic between two paths, say, more flows are set towards to s_1. Though high delay happens at s_1 with queue building up, the root cause is at another switch s_9. The signature of the root cause is whether the throughput of each path in an ECMP group is evenly distributed when a switch's queue experiences sudden congestion.

Algorithm 3 Culprit localization

```
Input: Frequent pattern $F$, Diagnosis data $D$
Output: Culprits Set $C$
1: for $pattern \in F$ do
2:     for $flow$ that traverse $pattern$ do
3:         $culprit.cause \leftarrow \textsc{Signature}(D)$
4:         $culprit.score \leftarrow pattern.score \cdot \frac{\#\text{ pkts of } flow}{\#\text{ pkts go through } pattern}$
5:         $C \leftarrow C \cup culprit$
6:     end for
7: end for
8: $C \leftarrow \textsc{merge}(C)$
```

- **Process Rate Decrease** is a port/switch-level cause. Due to limitation of resource (switch CPU, memory, etc.) or imperfect schedule scheme, the processing rate of the switch may decrease. As switch cannot process packet in time, a low process rate will bring about queuing buildup, therefore the latency raises up. Its signature is that pps remains relatively stable when a queue buildup occurs.
- **Delay** is a port/switch-level cause. Besides process rate decrease, switch errors like interrupts, insufficient power supply and configuration errors can result in delays outside the queue. The identifying characteristic of this root cause is that there is not obvious change in both pps and queue depth, yet the culprit pattern still has a high suspicious score.
- **Drop** is a port/switch-level cause. Though transient drop may happen due the high latency related network faults described above, unanticipated packet loss can also result from link failure like poor cable connection, unwell-behaved network updates, and missing forwarding table entries [53]. MARS detects drops by verifying the sequence ID of the sample epoch. Unlike other root causes related to latency, MARS applies another analysis logic to diagnosis drop event. The time range of the drop event can be known with the "epoch gap" field in the diagnosis data (as described in §4.3.2). MARS uses the number of dropped packets, as determined from the packet counter in the source and sink switches, to identify the affected flows and estimate the number of dropped packets. These flows are then categorized into the abnormal set, while unrelated flows are considered the normal set. MARS then runs another instance of SBFL with these two sets, and the link or switch with the highest suspicious score is considered the most likely culprit of the drop event.

While different frequent patterns may attribute to the same culprit, MARS merges the same cause at last. The actual abnormal localization dominates the causes' score eventually, i.e., the cause score should be lower than the SBFL score of the culprit pattern. Therefore, the merged score of a flow-level cause is the maximum score of repeat items, while other kinds of causes' merged score is calculated by summation. In addition, MARS merges port-level causes of the same type into a single switch-level cause when they are assigned to multiple ports within the same switch.

5 EVALUATION

5.1 Experiment Setup

We developed two prototypes of MARS for evaluation. The performance of MARS is evaluated on a Barefoot Tofino switch [8] in P4 with 1429 lines of code (LOC) in the data plane. The switch emulates multiple logical switches port-to-port connected by fiber-optic cables. The port rate is set as 10 Gbps. The Switch is physically connected to four Linux machines as hosts. Moreover, a simulation environment is set up in Mininet with BMv2 P4 software switches[2] on a physical machine with 8-core CPU, 16 GB memory, and Ubuntu 18.04 OS under a fat-tree topology (Fig. 6). The background flows' packet size and inter-packet gap are consistent with the dataset of UW datacenter trace[10] and the ECMP strategy is based on path weight. The control plane is written with 827 LOC in Python, and root cause analysis is implemented with 793 LOC in Python, using the SPMF data-mining library [17].

5.2 Fault Injection

The transmission speed of background flow is about 200 packets per second. Micro-bursts are generated by injecting one transient flow in a great amount, over 1000 pps within a second. The burst flow occupies the queue in switches quickly, leading to a transient high latency event (Fig. 7(a)). ECMP Load Imbalance is generated by setting a randomly picked switch's ECMP strategy from 1:1 to an imbalance ratio (random from 1:4 to 1:10). As a result, the throughput of two paths in an ECMP group varies (Fig. 7(b)). As the number of packets forwarded to one ECMP path increases, the latency of that path increases, while the latency of the other path decreases. Process Rate Decrease scenarios randomly select a port of a randomly picked switch and decrease its packet process rate lower than 100 pps. Delay and Drop event are generated with Chaosblade [1] by injecting the anomaly to the switch's interface(s).

5.3 Anomaly Detection Effectiveness

Fig. 8 presents the anomaly detection results on comparison between dynamic threshold and static threshold. The higher threshold causes more false negatives and lower recall, while the lower threshold causes more false positives. Static thresholds are easier to trigger false alarms, leading to low precision. As the reservoir can adjust to dynamic network traffic, it avoids many false alarms and reaches high precision. Furthermore, the reservoir without penalty factor α is much easier to be affected by anomalies (high latency), leading to numerous false negatives. The penalty factor helps the reservoir avoid this problem at the cost of 0.02 precision according to recently detected anomaly counts. As a result, the dynamic threshold of MARS achieves 0.96 Recall, 0.97 Precision, and 0.97 F1 score.

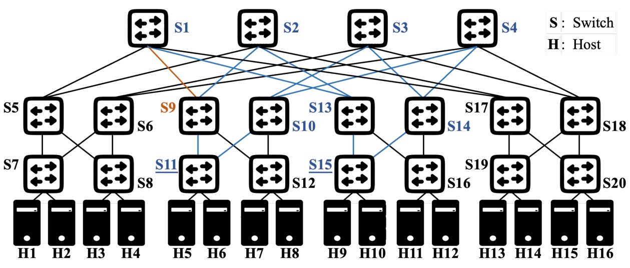

Figure 6: Fat-tree Topology.

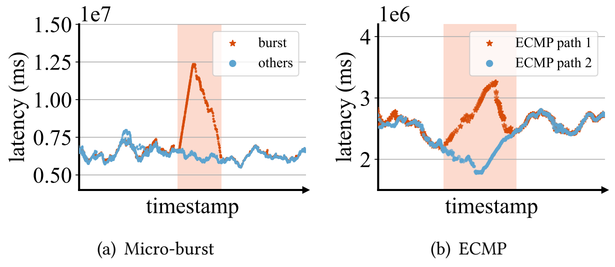

Figure 7: Examples of Anomaly Scenario.

5.4 Fault Localization Effectiveness

We evaluate the effectiveness of MARS in localizing faults in multiple scenarios. To enable the comparison to MARS, we adopted SpiderMon[47], IntSight[31], and SyNDB[26] to output an ordered culprit list. The result of SpiderMon is ordered by the degree in its Wait-For Graph (WFG). The result of IntSight and SyNDB is ordered according to the data from the conditional flow report and p-record, respectively. The result of SpiderMon is based on the level in its Wait-For Graph (WFG), while the ranking of IntSight and SyNDB is based on the query data from the flow report and p-record, respectively. The effectiveness of SyNDB may be overstated (represented by gray color in Table 1) as it is query-based and requires expert knowledge to determine which telemetry data it should query.

To evaluate the effectiveness of MARS, we introduce two metrics that are widely used in root cause localization. **Recall of *Top-k* (*R@k*)** reveals the probability that the root cause can be located within the top k culprits provided by the algorithm [29, 52]. A survey [27] conducted that over 73% developers only consider *Top-5* ranked elements. Thus, this paper splits the results into $R@k(k = 1, 2, 3, 5)$. **Exam Score** is a metric to measure the percentage of the false positives that need to be excluded manually by admin before locating the real root cause [24, 39, 49]. If the root cause is out of *Top-5*, we set a default 10 false positive cause before it. Note that the higher $R@k$ is better, while the lower Exam Score is better. Table 1 shows the result of the evaluation.

SpiderMon focuses on the micro-burst flow that occupies most line-rate. In this case, most flows wait for the culprit flow, thus the degree of the culprit vertex is high, with a large indegree and a small outdegree. However, when a flow bursts in a great amount, most wait-for relation is between burst flow itself, i.e., the indegree and outdegree are similar and SpdierMon would fail to localize the root cause. As SpiderMon only carries a cumulative latency of queuing delta time, which is based on whether to send the "spider" notice packet, it cannot sense the anomaly outside the queue. Therefore, SpiderMon cannot detect the exceptions of delay and drop, thus failing to start a root cause analysis. Though IntSight can sense the drop event at flow-level by comparing the source count and destination, it fails to locate the drop event at a switch/port-level. Similar to SpiderMon, IntSight updates contention points according to queuing delta time. As a result, IntSight falls short to locate the deeper root cause. As IntSight is not good at aggregate reports into a ranked metric, its recall is relatively low until *Top-5*. Since

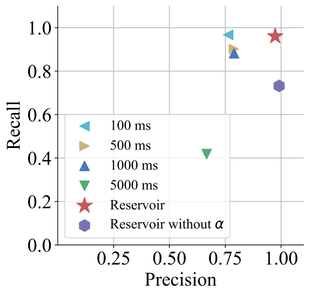

Figure 8: Anomaly detection under different thresholds.

Table 1: Recall and Exam Score of MARS (MS), SpiderMon (SM), IntSight (IS) and SyNDB (SN). Here only SN is aided by expert knowledge to diagnose.

Anomaly Cause	$R@1$ (%)				$R@2$ (%)				$R@3$ (%)				$R@5$ (%)				Exam Score			
	MS	SM	IS	SN	**MS**	SM	IS	SN	**MS**	SM	IS	SN	**MS**	SM	IS	SN	**MS**	SM	IS	SN
Micro-burst	**75**	50	10	44	**85**	62	39	73	**92**	73	60	79	**96**	75	81	94	**0.3**	0.3	2.4	1.5
ECMP	**88**	70	29	50	**100**	96	50	79	**100**	96	54	96	**100**	100	96	100	**0.1**	0.4	2.5	0.8
Process Rate	**94**	100	71	100	**100**	100	100	100	**100**	100	100	100	**100**	100	100	100	**0.1**	0	0.3	0
Delay	**73**	-	-	100	**83**	-	-	100	**87**	-	-	100	**93**	-	-	100	**0.4**	10	10	0
Drop	**67**	-	-	100	**94**	-	-	100	**100**	-	-	100	**100**	-	-	100	**0.4**	10	10	0
Overall	***83***	*44*	*21*	*79*	***95***	*52*	*32*	*90*	***97***	*54*	*40*	*95*	***99***	*55*	*55*	*99*	***0.3***	*4.1*	*5.0*	*0.5*

SyNDB cannot decide which data should be queried and diagnose without expert knowledge, it has to iterate the diagnosis process for all kinds of failure causes to find the root cause. Therefore, we have to assume SyNDB knows the root cause at first to conduct the corresponding diagnosis process, rendering SyNDB with expert knowledge to outperform other approaches in many circumstances. Besides, SyNDB does not have a trigger rule for drop events except for updating a forwarding rule. For packet loss caused by other reasons, SyNDB cannot sense it timely. Moreover, to check whether a drop event happened in history, traversing the entire database is needed, which is time-consuming.

In a nutshell, MARS is efficient to localize root cause without expert experience. It achieves 0.95 $R@2$ and 0.3 Exam Score overall. While SpiderMon and IntSight are skilled in specific scenarios, they fall short to detect delay and drop events. SyNDB has a high coverage of the network data, thus performing well in all scenarios as the same as MARS. However, its overhead is tremendous, which will be illustrated in the next subsection.

5.5 Overhead

Network Bandwidth. We estimated the network bandwidth of baselines to be compared with MARS, as depicted in Fig. 9. “Telemetry” refers to the additional bandwidth required for packet information, such as INT headers. “Diagnosis” refers to the data sent from the data plane to the control plane, including telemetry data for root cause analysis. SyNDB does not consume telemetry bandwidth as it does not require INT headers. However, it requires all switches to send recorded data to control plane, causing a significant amount of diagnosis bandwidth. IntSight requires a large INT header (33B) to perform anomaly detection and root cause analysis in switches, consuming a significant amount of telemetry bandwidth. It sends data to the control plane conditionally, resulting in less diagnosis bandwidth compared to SyNDB. SpiderMon has much lower telemetry bandwidth compared to IntSight as its INT header only contains latency information. Unlike SyNDB and SpiderMon which collect data from all switches, MARS only requires edge switches to send diagnosis data. In addition, MARS collects less data per edge switch compared with IntSight. For example, the bit map of IntSight that indicates a specific switch in packet’s transmission path is usually set at 48 bits per map. Thus, MARS consumes less diagnosis bandwidth. However, MARS requires more telemetry information in the INT header, leading to a slightly greater telemetry bandwidth than SpiderMon, but still much smaller than IntSight. Overall, MARS has the least total bandwidth consumption and the smallest diagnosis overhead. Though the total bandwidth overhead of SpiderMon and MARS is near, MARS collects information in a more comprehensive manner, providing a more dimensional and thorough root cause analysis.

Switch Resource Usage. Fig. 10 shows the usage of switch resource[1] and how MARS scales with the Ring Table size. Ring Table size indicates the number of history packet can be collected to the control plane on each switch once a time. The history data is saved in the SRAM (Static RAM) register of switch. MARS fits in the Tofino pipeline comfortably for now, and can scale to record more data as switch memory size increases over time [33].

Switch Memory Usage for PathID. The switch memory usage of IntSight is $M_{\text{IS}} = \sum_{p \in paths} \#hop_p \times size(\text{MAT}_{\text{IS}})$, and MARS requires $M_{\text{MS}} = \sum_{p \in paths} \sum_{h_p \in hops(p)} p_{hash} \times size(\text{MAT}_{\text{MS}})$. Here $p_{hash} \in (0, 1)$ is the probability of hash conflict. As a result, $M_{\text{IS}} > M_{\text{MS}}$ holds in all cases if $\text{MAT}_{\text{IS}} = \text{MAT}_{\text{MS}}$. In a $K = 4$ fat-tree topology, there are 216 paths between edge switches (8 one-hop paths, 8 three-hop paths, and 96 five-hop paths). As IntSight needs to assign MAT entries for all switches on a path, it needs to assign 512 MAT entries in total, with each MAT entry consuming around 7 bytes of memory. For MARS, though a MAT occupies around 10 bytes of memory, only 48 MAT entries are enough to distinguish all the 112 paths, with CRC16/CRC32 as the hash algorithm. Consequently, MARS saves more 43.6% switch memory than IntSight.

FSM Algorithms. Fig. 11 shows the comparison of different FSM algorithms in our fault scenarios, including PrefixSpan [23], Lapin [48], GSP [42], Spade [50], Spam [6], SM-Spade and SM-Spam [16]. Besides the min *support*, Some algorithms can limit the maximum pattern length. Since MARS only considers patterns whose length is less than two, the algorithms with maximum pattern length equals two perform relatively better. Most algorithms finish in 200 ms and consume less than 30 MB of memory. PrefixSpan performs the best among all algorithms.

5.6 Limitation

Expanding on the types of fault reasons requires expert knowledge intervention. Experts need to analysis and summarize the fault

[1]The PHV (Packet Header Vector) carries packet data throughout the Tofino pipeline. Hash Bits are used in hash generators, e.g., ECMP, and to calculate PathID. TCAM (Ternary Content Addressable Memory) is utilized in the matching part of MAT, such as the longest-prefix match for IP addresses. Action Data are the stage data for PHV ALUs (Arithmetic Logic Units).

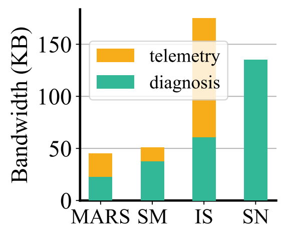

Figure 9: Extra bandwidth.

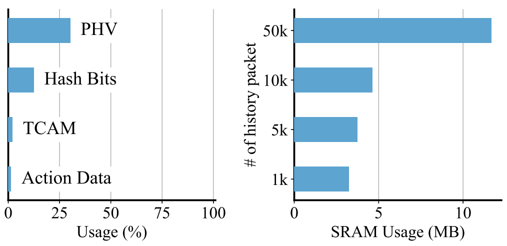

Figure 10: The resource usage in Tofino

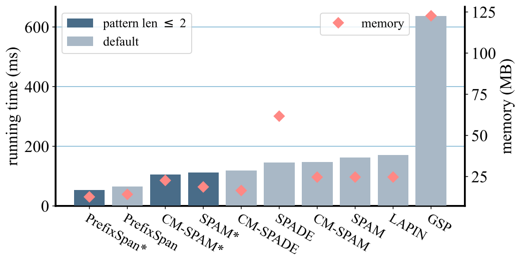

Figure 11: Overhead of FSM.

reasons, so that they can design corresponding signatures for fault matching. Besides, MARS can not handle continuous packet loss that has not been restored and report it promptly, as MARS cannot compare the packet count without receiving new packets.

6 RELATED WORK

Out-of-band network diagnosis. Out-of-band diagnosis methods either use packet mirroring at switches to monitor the network [21, 40, 46, 55], or send probes into the network to estimate the network status [18, 43]. NetSight [21] creates copies for all packets in the forwarding device and sends them to the control plane for further diagnosis. However, the “always on” mirroring for all packets incurs excessive data collection that is unrelated to the diagnosis. While [40, 46, 55] utilize sampling technique to reduce monitoring overheads, they perform poorly in diagnosis because simply sampling misses unexpected events easily. Pingmesh [18] and NetBouncer [43] install agents on each end-host to send probe packets, which would be collected by a processor to detect and diagnose the network failures. However, the probing traffic and the production traffic may travel along various paths, hindering accurate and timely diagnosis of transient failures.

Programmable switch assisted diagnosis. Since out-of-band network diagnosis methods introduce extra overhead to the network, existing works aim to leverage programmable switches to monitor the network and localize the network failure. BurstRadar [25], ConQuest [13], *Flow [41] depict the status of the queues when packets enter in the switch. However, they indiscriminately monitor and collect telemetry data from switches. SpiderMon [47] reduces the report overheads by providing an on-demand collection method. Only when a switch detects an anomaly, the telemetry data in all switches would be collected for fault localization. However, in most data centers, link utilizations are rather low in all layers except the core, and a subset of the core links often experience high utilization [10]. It is challenging to minimize the usage of core switches due to the requirement for monitoring and data collection. Therefore, MARS only needs the telemetry data collected from the edge switches to perform fault localization. Like SpiderMon, IntSight [31] and Marple [36] also detect anomalies by preset thresholds, hindering the adoption of the detecting accuracy in the network where the traffic is versatile. On the contrary, MARS detects anomalies with dynamic thresholds. Query-based diagnosis systems [19, 26, 45, 53] require the operators to understand the network and the potential locations of the failures in advance, hindering the timely and convenient diagnosis in a large-scale network. On the contrary, MARS is capable of automatically detecting and localizing the failures without any static queries.

7 CONCLUSION

This paper proposes MARS, a low-cost system for anomaly detection and precise root cause localization in programmable networks. MARS achieves low-cost monitoring using INT and an on-demand strategy to monitor, detects anomalies including drop and high delay with dynamic thresholds, and localizes the root cause with FSM and SBFL. Our evaluation on a real-world Tofino testbed and a simulated Mininet environment shows that, even with a low overhead in network bandwidth and switch memory, MARS can achieve accurate anomaly detection (0.97 F1 score) and precise root cause localization (0.95 R@2) for diverse network faults.

ACKNOWLEDGMENTS

The research is supported by the National Key Research and Development Program of China (No. 2019YFB1804002), the National Natural Science Foundation of China (No. 62272495), the Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515020054). The corresponding author is Pengfei Chen.

REFERENCES

[1] 2021. Chaosblade. https://github.com/chaosblade-io/chaosblade. Accessed: 2021-12-03.
[2] 2022. Bmv2. http://bmv2.org/. Accessed: 2022-09-13.
[3] 2023. P4Runtime Control Plane API. https://github.com/p4lang/p4runtime. Accessed: 2022-02-12.
[4] Kanak Agarwal, Eric Rozner, Colin Dixon, et al. 2014. SDN Traceroute: Tracing SDN Forwarding without Changing Network Behavior. In *HotSDN '14*. ACM, New York, USA, 145–150. https://doi.org/10.1145/2620728.2620756
[5] Aitor Arrieta, Sergio Segura, Urtzi Markiegi, et al. 2018. Spectrum-Based Fault Localization in Software Product Lines. *Information and Software Technology* 100 (2018), 18–31. https://doi.org/10.1016/j.infsof.2018.03.008
[6] Jay Ayres, Jason Flannick, Johannes Gehrke, et al. 2002. Sequential PAttern Mining Using a Bitmap Representation. In *KDD '02*. ACM, New York, USA, 429–435. https://doi.org/10.1145/775047.775109
[7] Jozef Babiarz, Roman M. Krzanowski, Kaynam Hedayat, et al. 2008. *A Two-Way Active Measurement Protocol (TWAMP)*. Request for Comments RFC 5357. Internet Engineering Task Force. https://doi.org/10.17487/RFC5357
[8] barefoot. 2023. Open Tofino. https://github.com/barefootnetworks/Open-Tofino. Accessed: 2022-02-03.
[9] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, et al. 2020. PINT: Probabilistic In-band Network Telemetry. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3387514.3405894 arXiv:2007.03731
[10] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *IMC '10*. ACM Press, Melbourne, Australia, 267. https://doi.org/10.1145/1879141.1879175
[11] Pat Bosshart, Dan Daly, Glen Gibb, et al. 2014. P4: Programming Protocol-Independent Packet Processors. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. 87–95. https://doi.org/10.1145/2656877.2656890

[12] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, et al. 2018. Catching the Microburst Culprits with Snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*. ACM, Budapest Hungary, 22–28. https://doi.org/10.1145/3229584.3229586

[13] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, et al. 2019. Fine-Grained Queue Measurement in the Data Plane. In *CoNEXT '19*. ACM, New York, USA, 15–29. https://doi.org/10.1145/3359989.3365408

[14] Chongrong Fang, Haoyu Liu, Mao Miao, et al. 2020. VTrace: Automatic Diagnostic System for Persistent Packet Loss in Cloud-Scale Overlay Network. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. ACM, 31–43. https://doi.org/10.1145/3387514.3405851

[15] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, et al. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *NSDI '18*.

[16] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, et al. 2014. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In *Advances in Knowledge Discovery and Data Mining (Lecture Notes in Computer Science)*, Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, et al. (Eds.). Springer International Publishing, Cham, 40–52. https://doi.org/10.1007/978-3-319-06608-0_4

[17] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, et al. 2014. SPMF: A Java Open-Source Pattern Mining Library. (2014), 5.

[18] Chuanxiong Guo, Lihua Yuan, Dong Xiang, et al. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. 45, 4 (aug 2015), 139–152. https://doi.org/10.1145/2829988.2787496

[19] Arpit Gupta, Rüdiger Birkner, Marco Canini, et al. 2016. Network Monitoring as a Streaming Analytics Problem. In *HotNets '16*. ACM, New York, USA, 106–112. https://doi.org/10.1145/3005745.3005748

[20] Peter J. Haas. 2016. Data-Stream Sampling: Basic Techniques and Results. In *Data Stream Management: Processing High-Speed Data Streams*, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi (Eds.). Springer, Berlin, Heidelberg, 13–44. https://doi.org/10.1007/978-3-540-28608-0_2

[21] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, et al. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *NSDI 14*. 71–85. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/handigol

[22] Chenhao Jia, Tian Pan, Zizheng Bian, et al. 2020. Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry. In *2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Budapest, Hungary, 1–9. https://doi.org/10.1109/NOMS47738.2020.9110326

[23] Jian Pei, Jiawei Han, B. Mortazavi-Asl, et al. 2001. PrefixSpan,: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In *Proceedings 17th International Conference on Data Engineering*. IEEE Comput. Soc, Heidelberg, Germany, 215–224. https://doi.org/10.1109/ICDE.2001.914830

[24] Jiajun Jiang, Ran Wang, Yingfei Xiong, et al. 2019. Combining Spectrum-Based Fault Localization and Statistical Debugging: An Empirical Study. In *ASE*. IEEE, 502–514. https://doi.org/10.1109/ASE.2019.00054

[25] Raj Joshi, Ting Qu, Mun Choon Chan, et al. 2018. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *APSys '18*. ACM, New York, USA, 1–8. https://doi.org/10.1145/3265723.3265731

[26] Pravein Govindan Kannan, Nishant Budhdev, Raj Joshi, et al. 2021. Debugging Transient Faults in Data Centers Using Synchronized Network-wide Packet Histories. In *NSDI '21*. 253–268. https://www.usenix.org/conference/nsdi21/presentation/kannan

[27] Pavneet Singh Kochhar, Xin Xia, David Lo, et al. 2016. Practitioners' Expectations on Automated Fault Localization. In *ISSTA '16*. ACM, New York, USA, 165–176. https://doi.org/10.1145/2931037.2931051

[28] Yuliang Li, Rui Miao, Mohammad Alizadeh, et al. 2019. DETER: Deterministic TCP Replay for Performance Diagnosis. In *NSDI '19*. 437–452.

[29] Jinjin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In *Service-Oriented Computing*. Springer, Cham, 3–20. https://doi.org/10.1007/978-3-030-03596-9_1

[30] Zhengzheng Liu, Jun Bi, Yu Zhou, et al. 2018. NetVision: Towards Network Telemetry as a Service. In *ICNP*. 247–248. https://doi.org/10.1109/ICNP.2018.00036

[31] Jonatas Marques, Kirill Levchenko, and Luciano Gaspary. 2020. IntSight: Diagnosing SLO Violations with in-Band Network Telemetry. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies* (2020-11-23). ACM, 421–434. https://doi.org/10.1145/3386367.3431306

[32] Nick McKeown, Tom Anderson, Hari Balakrishnan, et al. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review* (2008). https://doi.org/10.1145/1355734.1355746

[33] Rui Miao, Hongyi Zeng, Changhoon Kim, et al. 2017. Silkroad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching Asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 15–28. https://doi.org/10.1145/3098822.3098824

[34] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, et al. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015). https://doi.org/10.1145/2829988.2787510

[35] Julie A Morris and Martin J Gardner. 1988. Statistics in Medicine: Calculating Confidence Intervals for Relative Risks (Odds Ratios) and Standardised Ratios and Rates. *British Medical Journal (Clinical research ed.)* 296, 6632 (1988), 1313–1316. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2545775/

[36] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, et al. 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)*. ACM, New York, USA, 85–98. https://doi.org/10.1145/3098822.3098829

[37] Bronson Nathan. 2014. Solving the Mystery of Link Imbalance: A Metastable Failure State at Scale. https://engineering.fb.com/2014/11/14/production-engineering/solving-the-mystery-of-link-imbalance-a-metastable-failure-state-at-scale/

[38] P4. 2020. In-Band Network Telemetry (INT) Dataplane Specification. https://p4.org/p4-spec/docs/INT_v2_1.pdf

[39] Spencer Pearson, José Campos, René Just, et al. 2017. Evaluating and Improving Fault Localization. In *ICSE*. IEEE, 609–620. https://doi.org/10.1109/ICSE.2017.62

[40] Jeff Rasley, Brent Stephens, Colin Dixon, et al. 2014. Planck: Millisecond-Scale Monitoring and Control for Commodity Networks. In *SIGCOMM '14*. ACM, New York, USA, 407–418. https://doi.org/10.1145/2619239.2626310

[41] John Sonchack, Oliver Michel, Adam J. Aviv, et al. 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

[42] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT '96 (Lecture Notes in Computer Science)*, Peter Apers, Mokrane Bouzeghoub, and Georges Gardarin (Eds.). Springer, Berlin, Heidelberg, 1–17. https://doi.org/10.1007/BFb0014140

[43] Cheng Tan, Ze Jin, Chuanxiong Guo, et al. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *NSDI '19*. USENIX Association, Boston, MA, 599–614. https://www.usenix.org/conference/nsdi19/presentation/tan

[44] Shaofei Tang, Deyun Li, Bin Niu, et al. 2020. Sel-INT: A Runtime-Programmable Selective In-Band Network Telemetry System. 17, 2 (2020), 708–721. https://doi.org/10.1109/TNSM.2019.2953327

[45] Ross Teixeira, Rob Harrison, Arpit Gupta, et al. 2020. PacketScope: Monitoring the Packet Lifecycle Inside a Switch. In *SOSR '20*. ACM, New York, USA, 76–82. https://doi.org/10.1145/3373360.3380838

[46] Olivier Tilmans, Tobias Bühler, Ingmar Poese, et al. 2018. Stroboscope: Declarative Network Monitoring on a Budget. In *NSDI 18*. 467–482. https://www.usenix.org/conference/nsdi18/presentation/tilmans

[47] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, et al. 2022. Closed-Loop Network Performance Monitoring and Diagnosis with SpiderMon. 267–285. https://www.usenix.org/conference/nsdi22/presentation/wang-weitao-spidermon

[48] Zhenglu Yang and M. Kitsuregawa. 2005. LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern. In *ICDEW '05*. 1222–1222. https://doi.org/10.1109/ICDE.2005.235

[49] Guangba Yu, Pengfei Chen, Hongyang Chen, et al. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *Proceedings of the Web Conference 2021*. ACM, Ljubljana Slovenia, 3087–3098. https://doi.org/10.1145/3442381.3449905

[50] Mohammed J. Zaki. 2001. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning* 42, 1 (Jan. 2001), 31–60. https://doi.org/10.1023/A:1007652502315

[51] Matthew J. Zekauskas, Anatoly Karp, Stanislav Shalunov, et al. 2006. *A One-way Active Measurement Protocol (OWAMP)*. Request for Comments RFC 4656. IETF. https://doi.org/10.17487/RFC4656

[52] Xiang Zhou, Xin Peng, Tao Xie, et al. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In *ESEC/FSE 2019*. ACM, New York, USA, 683–694. https://doi.org/10.1145/3338906.3338961

[53] Yu Zhou, Chen Sun, Hongqiang Harry Liu, et al. 2020. Flow Event Telemetry on Programmable Data Plane. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. ACM, New York, USA, 76–89. https://doi.org/10.1145/3387514.3406214

[54] Yibo Zhu, Nanxi Kang, Jiaxin Cao, et al. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 479–491. https://doi.org/10.1145/2785956.2787483

[55] Yibo Zhu, Nanxi Kang, Jiaxin Cao, et al. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, USA, 479–491. https://doi.org/10.1145/2785956.2787483