Microscope: Queue-based Performance Diagnosis for Network Functions

Junzhi Gong
Harvard University

Yuliang Li
Harvard University

Bilal Anwer
AT&T

Aman Shaikh
AT&T

Minlan Yu
Harvard University

ABSTRACT

By moving monolithic network appliances to software running on commodity hardware, network function virtualization allows flexible resource sharing among network functions and achieves scalability with low cost. However, due to resource contention, network functions can suffer from performance problems that are hard to diagnose. In particular, when many flows traverse a complex topology of NF instances, it is hard to pinpoint root causes for a flow experiencing performance issues such as low throughput or high latency. Simply maintaining resource counters at individual NFs is not sufficient since the effect of resource contention can propagate across NFs and over time. In this paper, we introduce Microscope, a performance diagnosis tool, for network functions that leverages queuing information at NFs to identify the root causes (i.e., resources, NFs, traffic patterns of flows etc.). Our evaluation on realistic NF chains and traffic shows that we can correctly capture root causes behind 89.7% of performance impairments, up to 2.5 times more than the state-of-the-art tools, with very low overhead.

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; **Network performance analysis**; *Network performance modeling*.

KEYWORDS

NFV, performance, diagnosis

ACM Reference Format:
Junzhi Gong, Yuliang Li, Bilal Anwer, Aman Shaikh, and Minlan Yu. 2020. Microscope: Queue-based Performance Diagnosis for Network Functions. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3387514.3405876

1 INTRODUCTION

Network function virtualization (NFV) transforms hardware middleboxes to software running on commodity hardware – called Virtual Network Functions (VNFs), thereby bringing flexibility and agility to network operations. As a result, NFV has become popular in both industry and research [1, 9, 38, 48, 49]. For example, Internet Service Providers purchase network function solutions (e.g., NATs, Firewalls, VPNs) developed by different vendors, then run them in chains or DAGs (Directed Acyclic Graph) to serve traffic from various users.

Since VNFs process packets in software, there are inevitably more performance variations (e.g., throughput variations, high tail latency, jitters) than hardware platforms. These performance problems have a significant impact on service-level agreements and user experiences [35]. A survey we carried out with network operators has revealed various types of performance problems encountered in real-world NFV deployments (see § 2).

When performance problems emerge, the first question is who (users, ISP operators, NF vendors) is responsible for the problems. The process of finding answers often leads to blame games amongst these parties because many performance problems are intermittent and thus not easily reproducible. Furthermore, each party lacks full access to the system for debugging (e.g., ISPs cannot access NF vendor codes, while vendors may not know user traffic).

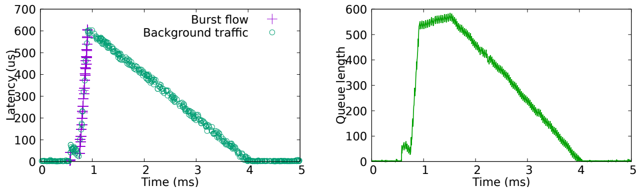
Let us use an example to highlight these challenges. Suppose we have an NF chain consisting of a Firewall followed by a VPN. Assume that some packets experience long latency at the VPN. Intuitively, operators start by blaming the VPN vendor for the problem. However, no problems are observed when the VPN is run without the Firewall. As a result, operators start to suspect that it is user traffic that is the root cause of the problem. Sometimes bursty traffic may lead to queue buildups at the VPN which results in a long latency. However, this turns out not to be the case this time. Rather, as we find in the end, this problem is caused by a bug in the Firewall that inflates the time it takes to process some flows, resulting in intermittent traffic bursts towards the VPN.

In practice, this problem is much more complex as user traffic goes through a variety of different NF chains. Many fine time-scale events occur in each NF (e.g., interrupts, cache misses, context switching) that could cause intermittent performance problems, whose effects may propagate across NF instances. Although there has been significant work dealing with NF performance optimization [18, 29, 32, 34, 39, 43, 51, 55], load balancing [23, 27], and auto-scaling [20, 44, 52], performance problems in NF chains still abound since it is hard to carefully engineer all the components to build a fully-optimized chain. And even if we succeed in doing so, we still need to optimize performance every time software, hardware, or NF chain configuration changes [13].

In this paper, we propose Microscope, a performance diagnosis tool that identifies the causal relations in a DAG (Directed Acyclic

SIGCOMM '20, August 10–14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3405876



(a) Packet Latency

(b) Queue length

Figure 1: We send CAIDA traffic [2] to a Firewall. At 570 μs, we inject a bursty flow which lasts 340 μs. (a) All the other flows arriving in the next 3 *ms* experience long latency. (b) The input queue quickly builds up but then takes around 3 *ms* to drain.

Graph) of NFs without any knowledge of their implementation. With the causal relations, we can identify the root cause(s) of performance problems such as misconfiguration and bugs, traffic anomalies, system interrupts, load imbalances, etc. There have been many research papers dealing with the identification of causal relations for large-scale distributed systems [16, 19, 36, 41, 44]. For example, NetMedic [36] infers causal relations by computing the probabilities of monitored behaviors (e.g., resource usage, processing rate) and problems (e.g., long latency) falling in the same *time window*. However, this work is not a good fit for NFs because NFs process packets in tens to thousands of microseconds and thus can easily be affected by the fine-timescale network (e.g., traffic bursts, queuing) and system behaviors (e.g., interrupts, context switching, data copies [37]). Correlating fine time-scale abnormal behaviors with problems is challenging because there can be many episodes of such behaviors and each episode can have a lasting impact. So it is hard to place the related behaviors and problems in the same time window. For example, Figure 1 shows that a bursty flow of 300 μs can impact flows that arrive in the next three milliseconds because of the long time the queues take to drain. While this happens, the impact of queuing may propagate to other NFs and flows (see more examples in § 2).

To address this challenge, we observe that it is through queues that adjacent NFs in a chain interact with one another. An upstream NF affects its downstream NF by changing the traffic rates to the queue, while the processing rate of the downstream NF affects the rate at which the queue drains. Moreover, as shown in Figure 1, a queue essentially "propagates" the impact of a previous event (e.g., traffic bursts) to future flows. Therefore, Microscope directly collects queuing information with low overhead. It then performs a queue-based causal analysis that quantifies the impact of flows and NFs on packet delay. Microscope also aggregates packet-level diagnosis into relational patterns which allow operators to automatically focus on the right flows and NFs.

Our evaluation demonstrates that Microscope can correctly capture 89.7% of all performance problems emanating from a variety of reasons such as traffic bursts, interrupts, NF bugs, etc., which is up to 2.5 times higher than the state-of-the-art tools. Microscope achieves this while keeping the runtime overhead quite low.

2 MOTIVATION

In this section, we show a few examples to highlight the challenges of diagnosing performance problems in NF chains. We next discuss our survey with network operators which corroborates these challenges.

2.1 Challenges of NF Diagnosis

We consider a DAG (Directed Acyclic Graph) of NFs where NFs can be provided by one or more vendors. We assume no access to NF implementation. There are NICs (SR-IOV), hardware switches, and/or software switches [10, 11, 30] that direct user traffic to NFs and forward traffic between NFs[1]. Note that one NF type may run multiple NF instances on multiple cores on the same or different servers to scale to traffic growth. For the rest of the paper, we use the term NF to refer to an NF instance unless noted otherwise. Our goal is to find the causal relations between the NFs/flows with intermittent[2] performance problems (e.g., low throughput and long tail latency).

The state-of-the-art approach for diagnosing causal relations is time-based correlation [16, 19, 36, 41, 54]. They are based on the assumption that abnormal behaviors happening in the same time window as the problem are more likely to be the root causes. This approach, unfortunately, does not work for NFs that exhibit abnormal behavior at fine time-scales (e.g., traffic bursts, CPU interruption, context switching), and where such behavior can have lasting impacts. Next, we provide a few examples to demonstrate the challenges with such scenarios.

1. Lasting impacts of microsecond-level behaviors. A flow may experience performance problems when it faces an abnormal behavior such as a traffic burst. However, the impact may last long after the flow finishes. As shown in Figure 1a, although the bursty flow finishes at time 1 *ms*, all the new flows that arrive up to 3 *ms* later still experience a long latency. The reason is shown in Figure 1b. The queue takes about 3 *ms* to drain because the Firewall is busy keeping up with the incoming flow rates.

In practice, in addition to traffic bursts, there are many other fine time-scale abnormal behaviors (such as interrupts, context switching etc.) that happen all the time at different NFs. The impact of these behaviors lasts for different time periods depending on the severity of resource contention, incoming traffic rates, and processing rates of NFs. This makes it challenging to define the right size for time window based correlation: a small window misses the correlations with behaviors whose impacts last longer than the window size, while a large window ends up including lots of unrelated behaviors.

2. Lasting impact propagates across NFs. A few fine time-scale behaviors at one NF may affect packets at another NF which has no spatial or temporal correlation with the first NF. To illustrate this, we send CAIDA [2] traffic through a chain consisting of a NAT followed by a VPN. We send another flow *A* directly through the VPN (see Figure 2a). Figure 2b shows that flow *A* at the VPN experiences low throughput during $[1.5ms, 2.3ms]$ time interval. If

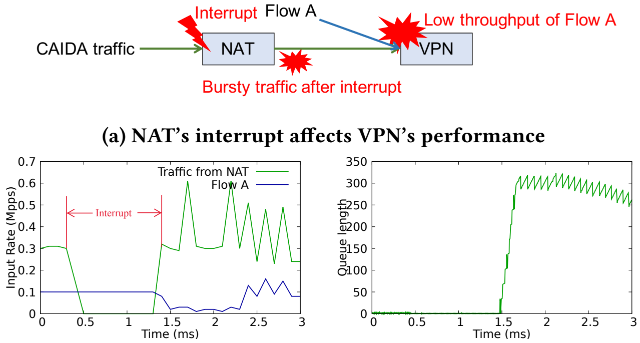
[1] For simplicity, we assume the switch is not the cause. We can easily treat the switches as another NF in the system for diagnosis if needed.

[2] Our solution also works for persistent problems. But persistent problems are much easier to diagnose and can use existing tools (e.g., PerfSight [53]).

we do time-based correlation, we may think the low throughput of flow *A* is caused by the surge in traffic from the NAT. But we send the CAIDA traffic at a constant rate throughout the experiment. In fact, the root cause is that the NAT experiences a CPU interrupt between $[0.5ms, 1.3ms]$, and hence is unable to send any traffic to the VPN during this time. After the interrupt, the NAT resumes processing and sends a burst of packets to the VPN, which causes the throughput drop for flow *A*.

The queue at the VPN illustrates this phenomenon (Figure 2c): The traffic burst from the NAT builds up the queue at the VPN starting around 1.5 *ms*, affecting flow *A* packets arriving from then on, although these packets only traverse the VPN, and do not overlap temporally with the interrupt at the NAT.

In practice, the performance depends on not only the abnormal behaviors at different NFs or traffic sources, but also the queue occupancy at these NFs when such behavior occurs. Therefore, it is not enough to identify causal relations based on temporal or spatial correlations.



(a) NAT's interrupt affects VPN's performance

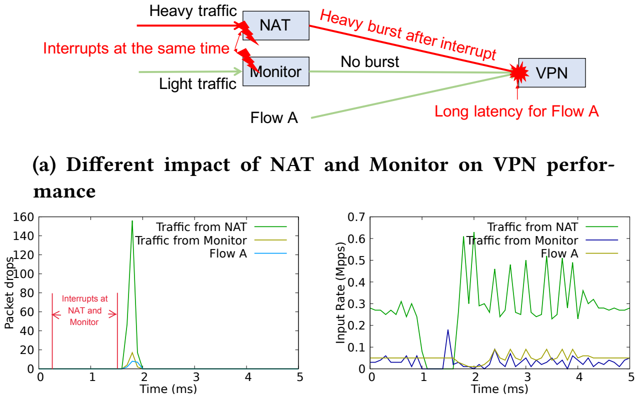
(b) Throughput at the VPN

(c) Queue length at the VPN

Figure 2: Impact propagation across NFs

3. Different impacts from similar behaviors. The same abnormal behavior may have a different impact on performance. Figure 3a shows a NAT and a Monitor both sending traffic to a VPN. The NAT sends traffic at 0.25 *Mpps* while the Monitor sends traffic at 0.05 *Mpps*, both with 64-byte packets. We also send flow *A* to the VPN directly. Figure 3b shows that all the flows experience different levels of packet losses during $[1.6\ ms, 2\ ms]$ interval at the VPN. Similar to the previous example, this is caused by interrupts occurring at upstream NFs (the NAT and the Monitor). However, it is hard to identify which upstream NF contributes more to the problem, because both interrupts happen before the packet loss period. The input rate changes in Figure 3c helps identify the causal relations. The input rate from the NAT increases more than that from the Monitor. This means the NAT's interrupt is the dominant contributor to packet losses.

The upshot is that it is not enough to simply correlate behaviors of components together, especially when there are many concurrent microsecond-level abnormal behaviors. Rather, we need to quantify the impact of these behaviors. In the example, in addition to identifying those packet losses are correlated with interrupts at both the NAT and the Monitor, we need to quantify each interrupt's contribution to change in input rate, so that we can focus on the most important problems.



(a) Different impact of NAT and Monitor on VPN performance

(b) Packet drops on the VPN

(c) Input rates changes

Figure 3: Different impact of similar behaviors

2.2 Survey on Performance Diagnosis

To understand the reality of performance problems and diagnosis of network functions today, we conducted a survey with 19 network operators (from ISPs, data centers, and enterprises) in January 2020. Our complete survey form and results are published at [7]. Among the survey respondents, four belonged to small networks (< 1K hosts), six to medium networks (1k-10K hosts), four to large networks (10K-100K hosts), and the remaining five to extra-large networks (>100K hosts). Below we describe the main findings.

These operators often face performance problems. In particular, five operators said they have to diagnose 10-100 performance problems monthly, while four of them spend more than 12 hours on performance diagnosis per month.

These problems are hard to diagnose because of diverse symptoms and root causes. Typical symptoms include: multiple NFs experience problems (e.g., low throughput) at the same time (seven operators experienced this), when the problems are intermittent (nine operators), and when the problems only happen for one user but not others (seven operators). Typical root causes include resource contention (7 operators saw this), traffic bursts (12 operators), interrupts (5 operators), and other NF bugs (15 operators). One type of tricky problems is caused by interactions between NFs. That is, the problem manifests only when multiple NFs are running together, not while debugging individual NFs in isolation. There are many different causes, such as upstream NFs' output traffic affecting downstream NFs (6 operators saw this), misconfiguration on one NF affecting another NF (8 operators), or resource contention (3 operators).

The top requirements for performance diagnosis tools are high accuracy (9 operators) and low overhead (12 operators). Moreover, many operators would like a *ranked* list of root causes (12 operators) where each cause indicates aggregated flows (7 operators) or network functions (9 operators).

3 MICROSCOPE KEY IDEAS

Microscope is a performance diagnosis tool that identifies causal relations for performance problems in a DAG of NFs. We make the following key design decisions in Microscope:

Leverage queuing periods to understand long-lasting impacts of problems at each NF. Since NFs are developed by different vendors and we do not have access to NF internal codes, we propose to focus on the queues between NFs to observe causal relations between NFs and with traffic sources. Our examples in § 2 show that queues can indicate the lasting impact of anomalous behaviors, their propagation across NFs, and quantify the impact from multiple behaviors.

Our key insight is that when a packet experiences a long queue, it is not only because of the current packets in the queue, but also because of all the previous packets that contribute to the queue buildup but already get processed. This is to say that if we had fewer packets, the current queue length would be shorter. Therefore, we introduce a *queuing period* which defines the time period from the time when a queue starts building (from zero packets) to the current time. As an example, consider Figure 1b, where for each victim packet p arriving at time t, the queuing period of p starts from 570 μs to t. By considering the entire queuing period, we can determine the root causes that may not temporally overlap with the observed problem.

Quantify causal relations based on packets received during the queuing period: Our next step is to understand the causal relations between anomalous behaviors at NFs and packets in the queue. Generally speaking, packets are stuck in a queue for two reasons: high input rate from upstream NFs or slow processing rate at the current NF. We tell whether it is upstream NFs or the current NF that contribute to the queuing and by how much, by comparing the input rate or processing rate of an NF *during the queuing period* to the peak processing rate of the NF. For example, in Figure 2c, we attribute the queue buildup at the VPN to the NAT because of the high input rate from the NAT. In Figure 3c, we tell the relative contribution of the NAT and the Monitor by checking their respective input rate changes.

Furthermore, the impact of abnormal behaviors is propagated across NFs through packets. Therefore, we propose to trace back the journey of all the packets in the queue and analyze how quickly these packets are processed at each NF.

Aggregate causal patterns: Given many fine timescale anomalous behaviors and several performance problems (e.g., tail latency packets), it is important for operators to focus on the most important problems and root causes. We propose a causal relation aggregation algorithm that automatically generates a ranked list of causal patterns with scores: *<culprit flow aggregates, culprit NF> ⟶ <victim flow aggregate, victim NF>: score*. This is based on AutoFocus algorithm [25], but we modify it to aggregate causal patterns instead of traffic clusters.

4 MICROSCOPE DESIGN

Microscope collects packet's timestamps, queuing, and flow information between NFs without accessing internal NF codes in the runtime (Table 1). Based on the collected information, Microscope selects victim packets which experience high latency, low throughput, or losses, and diagnose their root causes.

Name	Explanation
Timestamps	timestamps when an NF reads or writes a batch of packets to each queue
Batch size	the batch size when the NF reads a batch of packets from the queue
Flow information	e.g., source, destination IP addresses and port numbers
Packet IDs	e.g., IPID

Table 1: Information collected by Microscope during runtime

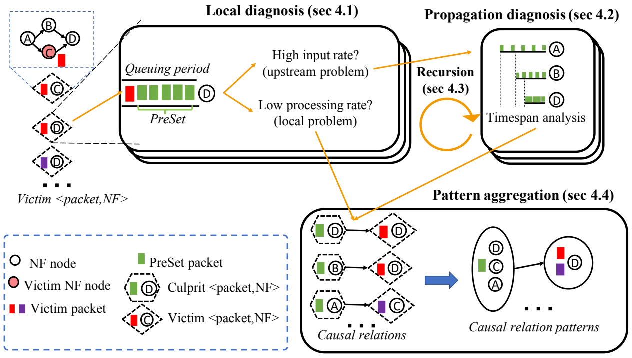


Figure 4: Microscope Architecture

Figure 4 shows that Microscope performs offline diagnosis in four steps: (1) For each victim packet at an NF where the packet experiences local abnormal performance, Microscope performs **local diagnosis** to understand whether the root cause is at the local NF or upstream NFs. The key idea is to leverage queuing periods to tell if the packet is delayed by low processing rates at the local NF or high input rates from upstream NFs (§4.1). (2) If the problem of the victim packet is caused by high input rates, Microscope performs **propagation analysis** to identify the culprit upstream NFs. Microscope inspects the packets in the queuing period (i.e., PreSet packets) and analyze how the *timespan* of these packets change at each NF (§4.2). (3) When an NF contributes to the high input rate of the *Preset* packets, this NF could also experience performance problems. Therefore, Microscope **recursively diagnoses** this NF using steps (1) and (2) (§4.3). (4) Microscope **aggregates causal relations** between culprit <packet,NF> to victim <packet,NF> into a small list of causal relation patterns using AutoFocus algorithms (§4.4). We now describe each step in detail.

4.1 Local Diagnosis

We focus on *victim packets* which experience bad performance (e.g., high latency or low throughput at the 99th percentile) or simply get lost (i.e., when we do not have records for them at some NFs). For each victim packet, we look at all the NFs on its path where its local performance is abnormal. Similar to NetMedic [36], we

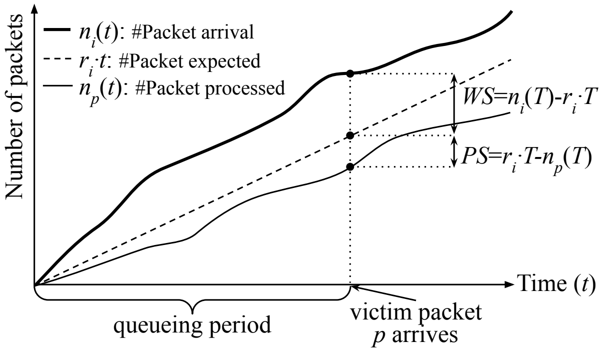


Figure 5: Diagnosing lasting impact at an NF (Section 4.1).

claim abnormality if the NF's performance is beyond one standard deviation computed over recent history.

Suppose a victim packet p has abnormal performance (e.g., a long latency) at NF f. Our goal is to identify all the abnormal behaviors (at f or upstream NFs) which impact the packet p. These behaviors do not have to overlap with packet p in time.

The direct cause of p's long latency is the queue with pending packets when p arrives at f. Thus we consider the queuing period from when the first packet is enqueued up to the time at which p arrives at the NF (Figure 5). By considering the entire queuing period, Microscope learns the whole history of how the queue is built up. Therefore, even if the culprits for the problem do not overlap with the victim packets in time, Microscope can still detect the cause of the problem.

Let T be the length of the queuing period at NF f. Let $n_i(T)$ and $n_p(T)$ be the number of packets arriving and getting processed at the NF during time T. Figure 5 shows that the queue builds up because the input rate is higher than the processing rate. This can happen due to two reasons: (1) *High input rate*: the input rate is higher than the peak processing rate; (2) *Low processing rate*: the processing rate of the NF f is lower than its peak processing rate (e.g., due to cache misses, CPU interrupt). We define r_i as the peak processing rate of an NF with the same hardware/software settings in the NFV topology.[3]

By comparing $n_i(T)$ and $n_p(T)$ with the expected number of packets ($r_i \cdot T$), we can quantify the two reasons. We use an input workload score S_i^f to represent the number of extra input packets, compared to the number of packets that can be processed at the peak rate during a period of T.

$$S_i^f = \begin{cases} n_i(T) - r_i \cdot T & \text{if } n_i(T) \geqslant r_i \cdot T \\ 0 & \text{if } n_i(T) < r_i \cdot T \end{cases} \tag{1}$$

We use a processing score (S_p) to represent the number of fewer packets being processed, compared to the number of expected packets processed during a period of T.

$$S_p^f = \begin{cases} r_i \cdot T - n_p(T) & \text{if } n_i(T) \geqslant r_i \cdot T \\ n_i(T) - n_p(T) & \text{if } n_i(T) < r_i \cdot T \end{cases} \tag{2}$$

Note that we define S_i^f and S_p^f to make sure they together cover all the packets that contribute to the queue buildup in T. That is, $S_i^f + S_p^f = n_i - n_p$ and $n_i - n_p$ is equal to the queue length.

4.2 Propagation Diagnosis

Suppose when we diagnose a victim packet p at an NF f, the S_i^f is positive (e.g., $S_i^{VPN} > 0$ in Figure 8). It means that the input workload contributed to the queue build up at f. The reason behind higher input workload could be any of the upstream NFs (which could have ramped up their processing rates) or the traffic sources themselves. In this section, we describe the propagation analysis algorithm we run to identify the causal relations amongst NFs.

Assume when p arrives at NF f, the queuing period has lasted for T. During T, there are $n_i(T)$ packets coming from upstream. We call this set of packets $PreSet(p)$. Our goal is to understand the history of $PreSet(p)$ and why these packets take T time at NF f. We trace back to the upstream NFs that $PreSet(p)$ traverses. To diagnose, we define *timespan of $PreSet(p)$ at an NF as the time between the first and last packet that leaves the NF.* Let T_{source}, T_A, T_B, and T_C be the timespan of $PreSet(p)$ at traffic source, NF A, NF B, and NF C respectively. We first discuss the case where $PreSet(p)$ traverses a chain of NFs, and then generalize to a DAG of NFs.

$PreSet(p)$ **traverses a chain of NFs.** Suppose $PreSet(p)$ traverses a chain of NFs (A, B, C, f). There could be many fine-timescale abnormal behaviors happening at each NFs. For example in Figure 6, there is an interrupt at A, and cross traffic at C. When $PreSet(p)$ arrives at A, A has an interrupt so these packets have to wait, and are processed back-to-back after the interrupt finishes. This squeezes out the inter-packet gaps, so the timespan reduces from T_{source} to T_A. When they arrive at B, B is slower than A, so it takes a longer time to process these packets, increasing the timespan to T_B. When they arrive at C, there is a queue, so packets in $PreSet(p)$ have to wait, and their timespan is squeezed to T_C. These packets cause a bursty input to f, because f cannot process them in T_C time interval. Overall, T_C is smaller than the expected timespan of the $n_i(T)$ packets ($T_{exp} = n_i(T)/r_i^f$). A smaller timespan causes a traffic burst at f, which affects p. So we need to account for the reduction from T_{exp} to T_C.

Note that Microscope only cares about the overall timespan, not the distribution of the n packets within the timespan. This is because, our goal is to diagnose victim packet p at f, no matter how the packets in $PreSet(p)$ distribute within the timespan at each upstream NF, they cause the same effect at f. Moreover, the timespan is easier to measure and compare across NFs than packet distributions.

The next step is to attribute this timespan reduction ($T_{exp} - T_C$) to the source, NF A, B, and C. We split score S_i^f proportionally based on their relative timespan reduction from previous hops. For example, C's score $S^{f \leftarrow C}$ gets a fraction $\frac{T_B - T_C}{T_{exp} - T_C}$ of S_i^f, because it reduces the timespan by $T_B - T_C$ out of the total reduction $T_{exp} - T_C$. Similarly, the source's score $S^{f \leftarrow source}$ gets a fraction of $\frac{T_{exp} - T_{source}}{T_{exp} - T_C}$. If $S^{f \leftarrow source}$ is above zero, we define $PreSet(p)$ as the culprit packets at the source.

The timespan is not always decreasing. For example, B increases the timespan from T_A to T_B. In this case, we treat the timespan

[3] We can measure the peak processing rate r_i by stress testing the NF offline with the same hardware and software settings or collecting the peak processing rate in history when running the NF in production.

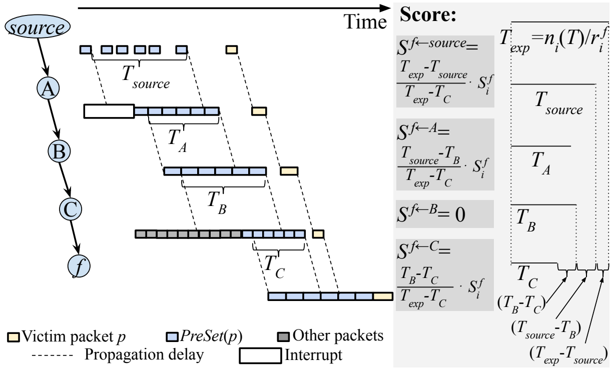


Figure 6: Timespan analysis (Section 4.2). The width of a packet represents its packet processing time; i.e., the wider width means a slower NF.

reduction by B's previous NF (i.e., A) to be $T_{source} - T_B$, because this is the effective reduction from f's perspective. So A's score $S^{f\leftarrow A}$ gets a fraction $\frac{T_{source}-T_B}{T_{exp}-T_C}$ of S_i^f, and B's score $S^{f\leftarrow B}$ is set to zero.

PreSet(p) **traverses a DAG of NFs.** If *PreSet*(p) goes through a DAG from the source to f, these packets go through different paths. So there is a set of paths (*PreSetPath*(p)) that *PreSet*(p) goes through. For each path $path_k$ in *PreSetPath*(p), we only consider the subset of *PreSet*(p) that goes through $path_k$, and assign scores to the source and NFs on $path_k$ in a similar way as the chain case.

The key question to generalize the chain algorithm to each path in the DAG is how to set the expected timespan T_{exp} for each path. Although each path has fewer packets than *PreSet*(p), it does not mean that the expected timespan is smaller. This is because packets on different paths usually interleave. When they fully interleave, they are expected to have the same timespan as *PreSet*(p), which is $n_i(T)/r_i^f$ (the same as the T_{exp} of the chain case). On the other hand, if packets on different paths do not fully interleave, the timespan is smaller than T_{exp}. This means one or more paths are more bursty than they should be and thus may be the root causes the performance of p (which is aligned with our timespan analysis). In conclusion, T_{exp} of each path should equal to $n_i(T)/r_i^f$.

Each NF (and the source) may get multiple scores, one from each path that it belongs to. The question is how to merge the scores of these NFs across paths. Simply summarizing these scores do not work because the sum may exceed S_i^f. The reason is that when the packets from multiple paths merge at NF f, the timespan after the merging may be larger than the timespans before the merging. So the timespan reduction on a single path may be larger than the timespan reduction for the whole *PreSet*(p). Therefore the summation of scores across all paths may be higher than S_i^f. In this case, we just proportionally scale down all the scores to match S_i^f.

4.3 Recursive diagnosis of PreSet packets

If some upstream NF reduces the timespan of *PreSet*(p), the reason could also be local processing at the upstream NF of the input traffic. For example, in Figure 6, A reduces the timespan because of a local interrupt, while C reduces the timespan because of the queue built up by other packets (the grey packets). The reason could also be a mixture of both local processing and input traffic in practice.

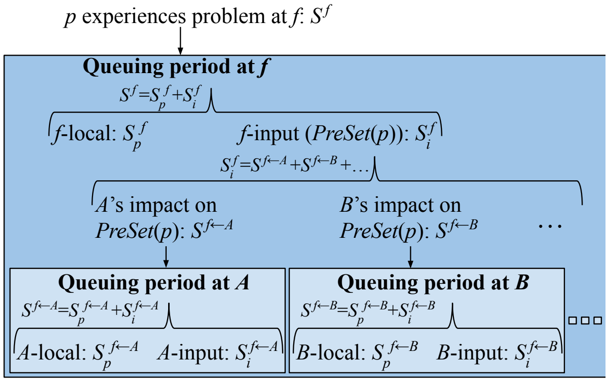

p experiences problem at f: S^f
Queuing period at f
$S^f=S_p^f+S_i^f$
f-local: S_p^f
f-input ($PreSet(p)$): S_i^f
$S_i^f=S^{f\leftarrow A}+S^{f\leftarrow B}+\ldots$
A's impact on $PreSet(p)$: $S^{f\leftarrow A}$
B's impact on $PreSet(p)$: $S^{f\leftarrow B}$
…
Queuing period at A
$S^{f\leftarrow A}=S_p^{f\leftarrow A}+S_i^{f\leftarrow A}$
A-local: $S_p^{f\leftarrow A}$
A-input: $S_i^{f\leftarrow A}$
Queuing period at B
$S^{f\leftarrow B}=S_p^{f\leftarrow B}+S_i^{f\leftarrow B}$
B-local: $S_p^{f\leftarrow B}$
B-input: $S_i^{f\leftarrow B}$
□□□

Figure 7: Recursive diagnosis (Section 4.3). Each box is one level of recursion.

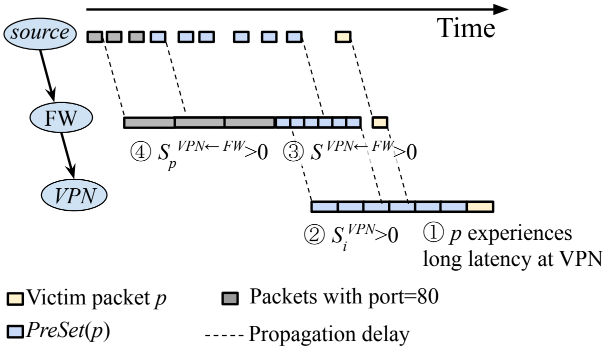


Figure 8: Diagnosing the example problem in §1.

To understand them quantitatively, we recursively apply techniques in § 4.1 and § 4.2 on the queuing period when the first packet of *PreSet*(p) arrive at each NF (e.g., the period of the grey packets at C in Figure 6). This recursive process is illustrated in Figure 7. The recursion terminates when no NF with positive S_i remains or when we reach the source.

We use the example in § 1 again to illustrate the need of recursion (see Figure 8). To understand why the Firewall reduces the timespan, we recursively analyze the queuing period after the arrival of the first packet of *PreSet*(p) at the Firewall. We find that the queuing is due to the slow processing of packets ($S_p^{VPN\leftarrow FW} > 0$), which we call *culprit packets*. Later we use pattern aggregation (§ 4.4) to determine which flows lead to such slow processing.

4.4 Pattern Aggregation

Given many packet-level causal relations, our next step is to aggregate them into causal relation patterns operators can act on. For example, if we can narrow down that certain flow aggregates always face problems at a particular NF, the vendor of this NF can investigate the configuration and the processing of these flows.

Our pattern aggregation takes the packet-level causal relations *<culprit packets, culprit NF>⟶<victim packet, victim NF>: score>*

as input, and generates aggregate patterns in the form of *<culprit flow aggregates, culprit NF set>→<victim flow aggregates, victim NF set>: score*. Here a flow aggregate is defined by the five-tuple derived from source IP prefix, source port range, destination IP prefix, destination port range, and protocol set; one can also extend flow aggregates to cover other packet fields. An NF set includes not only NFs but also input traffic sources.

Our pattern aggregation problem is similar to AutoFocus [25] which automatically aggregates multi-dimensional flows into traffic clusters that best represent the current traffic (i.e., hierarchical heavy hitter). Our goal is to find pattern aggregates along many dimensions which include culprit flow aggregates, culprit NF set, victim flow aggregates, victim NF set, where the flow aggregates further include five tuples or more and the NF set aggregates NF instances of the same type. We determine *significant pattern aggregates* which contributes to a large portion of the score (e.g., above a threshold *th*), after excluding descendants of each of these aggregates (similar to hierarchical heavy hitters [25]). Note that a higher threshold *th* leads to fewer details in the report. In practice, operators can adjust the aggregation threshold *th*, and thereby trade-off succinctness of the report with the amount of detail in it.

A key challenge is that we have far more dimensions than traffic aggregates. To speed up the aggregation process, we leverage the causal relation between culprit packets/NFs and victim packets/NFs. Most of the time, in a significant pattern aggregate, its culprit flow aggregate is also a significant flow aggregate if we just run AutoFocus on the culprit flow fields. The same observation applies to victim flow aggregates. This is because a victim packet is affected by a limited set of packets and a culprit packet set affects a limited number of victim packets.

Our pattern aggregation algorithm works in three steps: First, we group packet-level causal relations by culprit packets and culprit NFs. For each <culprit packet, culprit NF> we run AutoFocus on <victim packet, victim NF> dimensions and generate a few intermediate pattern aggregates with aggregated victim packet/NF fields. Next, we run AutoFocus again on the intermediate pattern aggregates to generate the final significant pattern aggregates. Our evaluation shows that this decoupling significantly reduces the aggregation time without losing any significant patterns.

Some problems may be intermittent but happen repeatedly (for example in Section 6.4, big-triggering flows arrive intermittently to the Firewall, which cause significant performance problems). Our aggregation algorithm can effectively find out these repeating problems over the timeframe when the operator runs Microscope, since they usually share some patterns.

5 IMPLEMENTATION

Microscope includes a data collector in the runtime and an offline diagnosis module.

Runtime information collection. We implement the data collector in the DPDK library with about 200 lines of code. Thus we can support any NF using DPDK as the packet I/O library. DPDK has a receive function and a transmission function, which handle the input and output queues of an NF. We instrument these functions to collect the required runtime data (see Table 1). It is feasible to extend our collector to other packet I/O libraries (based on VNF vendor's choice) like netmap [50] or VPP [11].

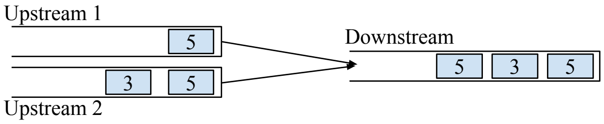


Figure 9: Resolving IPID ambiguity using order of packets.

To minimize the performance impact of Microscope on the NF, we keep the overhead on the critical path of the execution to a minimum. Instead of dumping to the hard disk directly, Microscope's collector writes the data to shared memory where it is picked up by a standalone dumper for storing on the disk.

For each packet, we record the five-tuple and the IPID, so a packet is uniquely identified across different NFs. Since DPDK fetches packets in batches (the maximum batch size is typically 32 packets), we also record a per-batch timestamp as well as the size of the batch. These data are sufficient for identifying the queuing period, because if the batch size is smaller than the maximum size, the queue must have been cleared, which is an indication of the start of a new queuing period (we will discuss cases where the queue is mostly non-empty in Section 7).

Directly collecting the data incurs a high overhead because we need more than 15 bytes per packet. We compress the data down to around two bytes per packet. The intuition is that the same packet traverses multiple NFs, so we just need to keep the five tuples of each packet at the end of the NF graph. For all other NFs, we only need to record the IPID of the packet.

However, this leads to challenges in reconstructing the trace of each packet (mapping the records of the same packet across different NFs), because different packets may have the same IPID in different NFs and we get confused about the traces of these packets. We resolve this ambiguity by using three pieces of "side-channel" information: the paths of packets, the timing of packets, and the order of packets.

(1) The paths of packets. We reconstruct each packet trace from the last NF where we record the five tuples backward to the source. This means at each step, we only need to look into packet records at the immediate upstream NFs, which reduces the chances of overlapping with other packets with the same IPID. Note that this filter does not work for NFs that assign path dynamically such as load balancers.

(2) The timing of packets. Since the delay from an upstream NF to a downstream NF is bounded, we can just consider mapping records on the two NFs that are within the maximum delay (i.e., queuing delay plus propagation delay). Since the propagation delay is small and the maximum number of packets in a queue in DPDK is 1024, out of 65,536 possible IPIDs, the chances that of two packets colliding with the same IPID in the delay bound is small.

(3) The order of packets. The intuition is illustrated in Figure 9 where the IPID 5 has ambiguity. However, if IPID 3 is unambiguous, we know that the left IPID 5 in the downstream queue cannot come from upstream 2 if packet ordering is to be preserved. This allows us to resolve the ambiguity for IPID of 5.

Offline diagnosis. The offline diagnosis module includes 6000 lines of code implemented in C and C++. Operators define the victim packets as those that encountered latency above a threshold, throughput below a threshold, or got lost. For these victim packets, Microscope performs the steps outlined in Figure 4 and outputs a list of causal relation patterns.

For each victim packet, we need to recursively diagnose the first packet in the queuing period at each NF (see § 4.3). In theory, the maximum number of recursions is the sum over the number of upstream NFs for each NF f (i.e., $\sum_{f} N\_upstream_{f}$, where $N\_upstream_{f}$ is the number of upstream NFs for an NF f). In practice, for our 16-NF evaluation topology (§ 6), we need a maximum of five recursions. This is because only a small number of culprit NFs are causally related to a victim.

6 EVALUATION

In this section, we evaluate the accuracy and performance of Microscope. Our evaluation shows that Microscope can correctly capture 89.7% of all performance problems of various types (traffic bursts, interrupts, NF bugs, etc.), up to 2.5 times more than the state-of-the-art tools. We also demonstrate that this can be achieved with a very small overhead during runtime information collection.

6.1 Setting

Network function chain. As shown in Figure 10, we run an NF chain consisting of four types of NFs: NATs, Firewalls, Monitors, and VPNs. This chain is a small-scale replica of commonly used NF chains in practice [4, 8, 48]. Incoming traffic is load balanced at flow level based on the hash of packet header fields. If a flow matches a rule at the Firewall, it is forwarded to the Monitor, otherwise it directly traverses to the VPN.

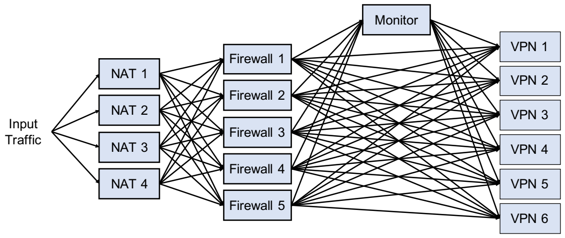


Figure 10: Network function chain used in evaluation

In our evaluation, all NFs are based on DPDK 18.08 [3]. We use NATs, Firewalls and VPNs provided by Click-DPDK [40] while we implemented the Monitor ourselves using the DPDK library. We run the NFs on Linux servers, and each NF instance is a single process bound to a specific physical core, in order to provide the best performance. Each NF uses SR-IOV for network I/O.

Network traffic trace. We use CAIDA [2] traces as traffic in our evaluation; traces are replayed using MoonGen [24] traffic generator. Since the software NF performance is mostly determined by the packet rate, not the byte rate, we set packet-size to 64 bytes to subject our system to high packet rates.

Aggregate threshold. We use 1% as the threshold in the aggregation algorithm, with which we think Microscope reports a reasonable number of causal relation patterns in our evaluation results.

Alternative approach. We compare Microscope with NetMedic. Since NetMedic is a diagnostic tool on general networked systems, we modify NetMedic to adapt it to our NFV system. NetMedic uses a template of graphs to model system components and causal relationships between them. In our context, the nodes are NFs, and the edges exist between NFs that directly exchange traffic (the DAG in Figure 10). NetMedic captures various resource usage and performance metrics for each component in the graph. In our case, we monitor all variables related to NF performance, including CPU usage, memory usage and traffic rates for each NF. NetMedic correlates abnormal behavior occurring in the same time window. In NFV systems, the packet delay is usually very small, so we set the time window to 10 ms, which we find to be the best window size in our evaluation scenarios. We not only compare accuracy between Microscope and NetMedic, but also evaluate accuracy when NetMedic is configured with different window sizes.

Evaluation platform. We run our evaluation on two hosts. Host 1 runs the MoonGen traffic generator, while the entire NF chain (consisting of a total of 16 NFs) runs on host 2. Host 1 is a Dell R730 server, having 10 cores, 32 GB memory, and a two-port 40 Gbps Mellanox ConnextX-3 Pro NIC. Host 2 is a Dell T640 server, having two CPU sockets, each consisting of 10 cores. It has 128 GB memory, and a two-port 40 Gbps Mellanox ConnextX-3 Pro NIC. Both servers run Ubuntu 18.04 Linux OS. As a traffic generator, MoonGen dynamically allocates CPU cores to keep up with the traffic rate in the traffic trace. Each NF instance, on the other hand, is bound to a dedicated core.

6.2 Diagnostic Accuracy

Methodology. We compare the accuracy of Microscope and NetMedic. Ideally, we would like to run evaluation on real problems, but unfortunately ground truth is often hard to come by in such scenarios. So we evaluate the accuracy by injecting problems ourselves. To make sure the ground truth is clear, we generate the CAIDA traffic at a moderate rate (1.2Mpps) so that other problems (i.e., the ones that are not injected) are much less significant and frequent than the injected ones.

Specifically, We inject three kinds of problems: (1) Traffic bursts: We randomly select 5 five-tuple flows and inject traffic bursts at the source with a burst size randomly chosen from 500 to 2500 packets. (2) Interrupts: We randomly select an NF instance and inject an interrupt with a duration randomly chosen between 500 and 1000 μs. (3) NF code bugs. We inject a bug at a random firewall instance that processes specific incoming flows at a low rate (0.05 Mpps). We inject flows that trigger this bug. The flow size is randomly chosen between 50 and 150 packets. Our goal is to identify the culprit flows (with traffic bursts), culprit NF (where we inject interrupts), or culprit NF-flow pairs (in the firewall case). We make sure the injected problems are separate enough in time so we unambiguously know the ground truth. We run our traffic for 5 seconds and collect around 12.5 MB data in the run-time for each evaluation.

Accuracy metric. NetMedic reports a ranked list of possible culprits. For comparison, we also rank different culprits by their scores, and get the rank of the real culprits. Note that lower rank is better here. In fact, the ideal diagnostic result would be the one where the injected problem is flagged as the top culprit.

Overall accuracy. Figure 11 shows that Microscope outperforms NetMedic. The two curves connect the rank of each victim packet for the two algorithms, and they are independently sorted based on the rank. Microscope reports rank of one for 89.7% of cases. For the other 10% or so cases, there are always other culprits that happen concurrently with the injected one. For example, when we inject traffic bursts, sometimes interrupts occur at the same time, and these two culprits both contribute to the performance problem. For such scenarios, Microscope identifies other problems as the top reason rather than the injected ones. On the other hand, NetMedic reports rank of one for only 36% cases, and rank≤5 for only 66% cases. Next let us delve deeper into the three types of injected culprits.

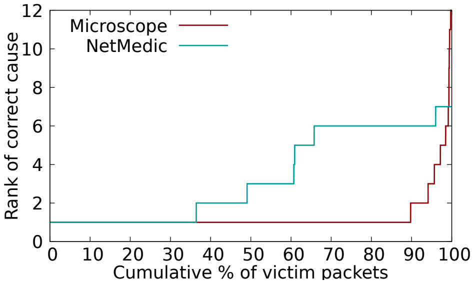


Figure 11: Overall diagnositic accuracy of Microscope and NetMedic. The x-axis represents the cumulative percentage of victim packets, and the corresponding y-axis represents the rank of the correct cause. For example, if the curve goes through point (x, y), it means for x% of victim packets, the rank of the real cause is no larger than y.

Diagnosing injected traffic bursts. Figure 12a shows the result for victim packets affected by traffic bursts. For 99.8% of victim packets, Microscope names the traffic burst as the most likely cause. In contrast, NetMedic's diagnosis is much less accurate. In fact, only for 3.7% of victim packets NetMedic ranks the traffic burst as the top one. For 39.9% of victim packets NetMedic ranks the traffic burst the second-most likely culprit.

To understand why sometimes Microscope and NetMedic fail to report burst as the most likely cause, we analyzed many such cases manually. For Microscope, in most of these cases, there are other culprits such as interrupts happening concurrently with the injected traffic burst which also affect the packet performance significantly. Microscope ranks some of these natural culprits before the injected bursts, which is not totally incorrect. However, NetMedic almost always misdiagnoses the problem, because it is often misled by the local processing rate. When a burst arrives at an NF, the local processing rate is always much higher than the normal time, so NetMedic ends up ranking the local problem as the top cause.

Diagnosing injected interrupts. Figure 12b shows the result for victim packets affected by interrupts. For 85.0% of victim packets, Microscope reports rank one for the interrupt. In contrast, NetMedic only reports rank one for 52.8% of victim packets.

For cases where Microscope is not able to pinpoint injected interrupts as the top reason, it is due to other events such as traffic bursts or other interrupts that affect the outcome. In contrast, NetMedic misdiagnoses a lot of cases. For example, due to the delay of impact propagation (similar to the second example in § 2), NetMedic is unable to correlate the victim with the interrupt (NetMedic still gives it a rank because it gives every possible culprit a rank).

Diagnosing injected NF bugs. Figure 12c shows the result for victim packets affected by the NF bug at Firewall 2. The bug is ranked first for 73.0% cases by Microscope, and has rank no larger than two for 95.5% cases. However, the bug is ranked first for only 63.3% cases by NetMedic, and all other cases have a rank larger than 3. For those cases where Microscope assigns second rank to the bug, Microscope ranks traffic bursts from the source first, because in this experiment, we manually injected some traffic to trigger bugs in the Firewall which increased the traffic rate. On the other hand, NetMedic assigns fourth to sixth place to the real cause for a large fraction of cases. In most of these cases, the problem happens in VPNs which are propagated from the bug in the Firewall, but NetMedic cannot correlate the bug triggering with the victim packets because there is a time gap between the bug and the final problem. Thus, it places the correct culprits lower that the other possible culprits (such as the Monitor and the other four Firewalls).

Runtime overhead. We also test the overhead of our runtime information collector by determining the degradation of the peak throughput at NFs which we find to be between 0.88% and 2.33% for different NFs. Note that this is the worst case overhead (under peak throughput); in reality the overhead is lower since NFs do not constantly run at the peak throughput.

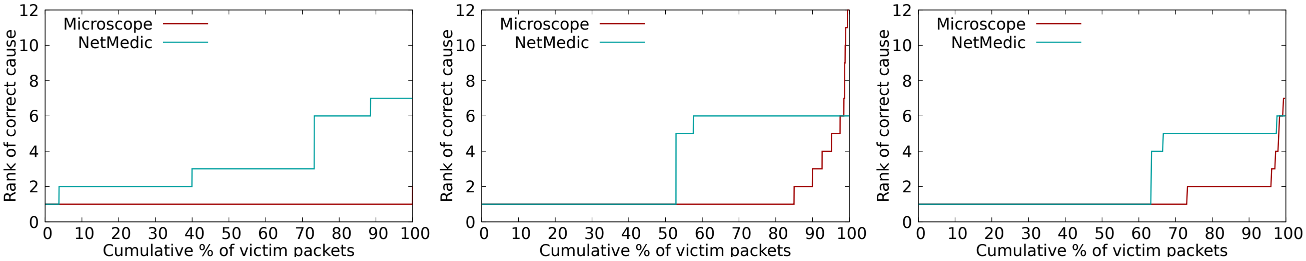
NetMedic accuracy with different time window sizes. Figure 13 shows how the accuracy (the percentages of results that rank the correct answer first) of NetMedic varies with different time window sizes. For all window sizes, NetMedic achieves much lower correct rate than Microscope (see Figure 11). Since NetMedic achieves the best correct rate when the window size is 10ms, we use 10ms as the NetMedic window size in other experiments.

6.3 Detailed Evaluation

Impact of burst sizes: We first inject traffic bursts from the source with different burst sizes, from 200 packets to 5000 packets. We found that when the burst size is 5000 packets, Microscope names the traffic burst as the most likely cause for all victim packets. As the burst size decreases, the accuracy also decreases. The reason for the accuracy decrease is that, when the burst size is small, it will contribute less to the queue relative to other concurrently occurring culprits.

Impact of interrupt lengths: We inject interrupts with different lengths, from 300μs to 1500μs. We found that when the interrupt length is 1500μs, Microscope names the interrupt as the most likely cause for almost all victim packets. However, as the interrupt length decreases, the accuracy also decreases. It is because when the interrupt length decreases, fewer packets are buffered due to the interrupt, and so the contribution of the interrupt is smaller compared to other concurrent culprits.

Impact of propagation hops: We inject different types of problems, and classify victim packets based on how many hops it takes for the effect to propagate to the ultimate victim. We found that as the number of hops increases, the accuracy of Microscope decreases. The reason is that, when problems propagate across hops, concurrent culprits can also propagate to the same victim, and thus the impact of the problem we inject is smaller.



(a) Traffic bursts **(b) Interrupts** **(c) NF bugs**

Figure 12: Diagnostic accuracy of Microscope and NetMedic for each injected culprit.

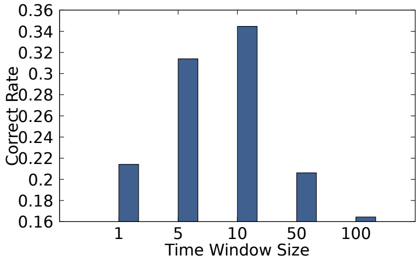


Figure 13: The correct rate of diagnosis when NetMedic is configured with different time window sizes.

6.4 Effectiveness of Pattern Aggregation

To demonstrate the effectiveness of pattern aggregation, we run CAIDA traffic at 1.2 Mpps, and inject flows that trigger the bug at Firewall 2. The bug-triggering flows are TCP flows from 100.0.0.1/32 to 32.0.0.1/32, with source TCP port number in [2000, 2008] and the destination TCP port numbers in [6000, 6008]. The experiment resembles the example in § 1. Note that Microscope has no explicit information about the bug. Neither does it know about the flows that trigger the bug.

Pattern aggregation presents concise results. There are a total of 84K causal relations, and pattern aggregation aggregates them to 80 different patterns. The run-time of the aggregation is around three minutes. The number of patterns could be reduced by further optimizations. For example, currently each port in [2000, 2008] and [6000, 6008] is in separate patterns, because the raw hierarchical heavy hitter algorithm we use [25] only considers either the static port range (1024-65536) or single port numbers. If we provide adaptive port ranges, we expect to report fewer rules (e.g., a single pattern with source port range of 2000–2008 and destination port range of 6000–6008). Furthermore, operators can also tune the threshold in the aggregation algorithm to adjust how many details the report contains.

Pattern aggregation helps us identify the bug-triggering flows. Figure 14 shows a snippet of the aggregation results. Four of the patterns contain the bug-triggering flows as culprits. Such information can be very helpful in diagnosing the bug.

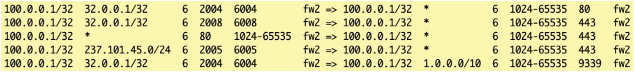

```
100.0.0.1/32  32.0.0.1/32       6  2004  6004        fw2 => 100.0.0.1/32  *          6  1024-65535  80    fw2
100.0.0.1/32  32.0.0.1/32       6  2008  6008        fw2 => 100.0.0.1/32  *          6  1024-65535  443   fw2
100.0.0.1/32  *                 6  80    1024-65535  fw2 => 100.0.0.1/32  *          6  1024-65535  443   fw2
100.0.0.1/32  237.101.45.0/24   6  2005  6005        fw2 => 100.0.0.1/32  *          6  1024-65535  443   fw2
100.0.0.1/32  32.0.0.1/32       6  2004  6004        fw2 => 100.0.0.1/32  1.0.0.0/10 6  1024-65535  9339  fw2
```

Figure 14: A snippet of pattern aggregation result. Each row is one pattern: <culprit 5-tuple> <culprit location> => <victim 5-tuple> <victim location>

Why pattern aggregation helps. Usually it is difficult to identify the bug-triggering flows because they are mixed with other normal flows. But pattern aggregation is effective because it collectively analyzes packets in all queuing period when the processing rate is low. During these queuing periods, packets from the bug-triggering flows appear from frequently than a random flow, so the bug-triggering flows stand out in the pattern aggregation.

6.5 Running in the Wild

We now study how Microscope diagnose performance problems of NFs without any injected bugs. We run a one-minute CAIDA traffic at a high load (1.6Mpps, 64-byte packet size), and use Microscope to diagnose the 99.9-th percentile latency (a total of 80K victim packets). We find that diverse types of problems emerge at the high load. We now present some interesting findings.

The victims caused by propagation is considerable. As shown in Table 2, 21.7% of all victim packets are caused by propagation, and 10.9% are caused by at least two-hop propagation[4]. This emphasizes the importance of locating the right culprit location, which can prevent blame game across operation teams managing different NFs in the real world. Without Microscope, it is difficult to find the right location.

Even though a large fraction of the culprits are local, Microscope also provides very insightful diagnostic information for them, such as the flow patterns (see § 6.4) and the timing.

The time gap between a culprit and its victim is highly variable. Figure 15 shows the CDF of the time gap, which varies from 0 to 91 ms. While half of them are under 1.5 ms, the other half spread almost evenly from 1.5 to 50 ms, with a long tail reaching 91 ms. This means for time-based correlation, it is very hard to find the appropriate time window: a small window may miss the real causal relations, while a large window includes lots of irrelevant signals that mislead the correlation (both are observed when using NetMedic in § 6.2). This highlights the benefit of Microscope's queue-based diagnosis.

[4] 21.7% is the sum of all cells in Table 2 that represent propagation. 10.9% is the sum of cells that represents at least two-hop propagation.

Culprit \ Victim	**NAT**	**Firewall**	**Monitor**	**VPN**
Traffic sources	**5.51%**	**1.43%**	**0.64%**	**3.56%**
NAT	10.46%	**1.84%**	**0.812%**	**0.64%**
Firewall	0%	27.27%	**2.49%**	**3.85%**
Monitor	0%	0%	19.00%	**0.89%**
VPN	0%	0%	0%	21.60%

Table 2: Breakdown of problem frequencies based on culprits and victims. Rows represent culprit NFs and columns represent victim NFs. Numbers show the percentage of problems for each [culprit→victim] pair. Bold numbers represents problems that propagate across different NFs.

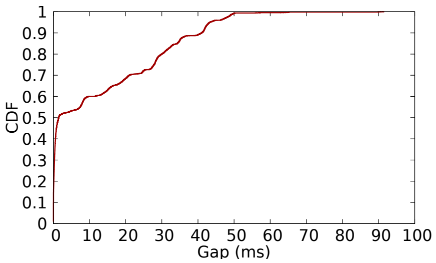


Figure 15: The CDF of the time gap between the culprit and the victim of each causal relation.

Culprit \ Victim	**NAT**	**Firewall**	**Monitor**	**VPN**
NAT1	3.29%	0.68%	0.35%	0.26%
NAT2	2.06%	0.25%	0.12%	0.05%
NAT3	2.92%	0.66%	0.24%	0.29%
NAT4	2.18%	0.25%	0.10%	0.04%

Table 3: Frequency differences for problems caused by different NAT instances

NFs of the same type can cause different levels of impacts. As shown in Table 3, NAT1 and NAT3 cause more problems, at all layers of NFs (we also observe such an uneven impact phenomenon in other types of NFs). However, the traffic is evenly distributed across different NATs. This suggests that many problems stem from factors that exhibit temporal unevenness, such as interrupts and temporal distribution of traffic.

Microscope is very helpful in diagnosing these problems, because it provides and analyzes the queuing, which is the consequence of temporal unevenness.

Some flows are more likely to cause problems. We perform pattern aggregation, and find that the traffic bursts are comprised of certain flow aggregates. Initially we suspected that the skew was due to the skew in traffic: i.e., larger flow aggregates in the traffic were more likely to appear in the bursts. However, when we compared the flow aggregates in the bursts and the flow aggregates over all traffic, we saw a significant difference. This means some flows are more likely to form bursts and lead to problems.

Microscope provides very useful insights for diagnosing such problems, such as the packet information in the queuing period. Without them, it is very hard to diagnose.

7 DISCUSSION

Microscope could fail. In practice, Microscope cannot always get the correct answer. Microscope could fail in the following cases: 1) The expected rate r_i of NFs is not measured correctly. 2) Microscope fails to identify the path of packets by only using the IPID. 3) The queuing period is not measured accurately due to the inaccuracy of timestamps.

Limitation of Microscope. Microscope can diagnose problems from NFs that keep the IPID of the packet, since Microscope uses IPID to identify the packet. Even if the IPID changes, there are other ways to generate a unique ID per packet [22]. For those NFs within which there is no one-to-one mapping for packets before and after the processing of the NF (e.g., compression proxies, TLS terminations), Microscope then cannot diagnose those NFs, but we can still diagnose the NF chain before such NFs and that after such NFs. Those NFs fundamentally require a white-box approach to diagnose, which we cannot help for now.

Non-DPDK NFs and other network components. Note that our implementation can collect data from all DPDK-supported NFs. All other network components, including switches and NICs, are treated as one component in our topology. If we also want to diagnose the problems in switches and NICs, we can treat them as different components in the same way as NFs, and thus we also need the data from queues in switches and NICs. When running NFs in different machines, we need to align the timestamp of data from different machines. This needs clock synchronization (microsecond level), which is already supported in PTP and Huygens [5, 28].

Problems not caused by long queues. Long latencies or packet drops could be caused by the long queue or the misbehaviors of the NF. We only focus on long queues in our paper. For the case of misbehaviors of NFs, the problem could be easily detected by our trace: we can know the delay within the NF by checking the timestamp difference of the packet in the input queue and the output queue, and report that those packets with large in-NF delay are caused by misbehaviors of NFs.

What if the queue is not empty in most cases? In our description of the algorithm, the start of a queuing period is when the queue length exceeds zero, but this is not required. In fact, we can also use a non-zero queue length threshold to define the start of a queuing period, to handle the case when the NF queues may be non-zero for a long time.

To implement Microscope with non-zero threshold, we just need to read the queue length from the NIC and compare it with the threshold. Unfortunately our NIC cannot report queue length, so as a workaround, we use the batch sizes to infer whether the queue is empty or not, which can only evaluate the threshold of zero. We leave the evaluation of non-zero threshold to future work.

8 RELATED WORK

In this section, we discuss works related to Microscope. Given the nature of network function virtualization, we discuss existing solutions for performance diagnostics in the domain of networks and distributed systems[5]. But before going into direct comparisons we'll first discuss a few of the recent works on NFV performance optimization and then discuss how Microscope is different from existing works based on its ability to diagnose problems at fine-grained timescales and across functions in service function chains.

Performance optimization: There has been a great effort on performance optimization for NFV systems and distributed systems in multi-tenant environments.

Performance optimization for distributed systems: Retro [44], Ernest [52], and HUG [20] are some of the first efforts in this regard. These systems are mainly focused on resource allocation optimization in a distributed system.

NF service chain optimization: NFP [51] and Parabox [56] try to reduce end-to-end latency by expliting parallelism in NFV chain. NFVNice [42] proposes a service chain management framework. It monitors loads of network functions to provide fair, efficient, and dynamic resource scheduling. Metron [38] and Slick [14] do resource optimization while implementing an NF chain inside server and network respectively by reducing inter-node communications within each server.

NF performance optimization: NF performance optimization discussed here can be divided into three lines of work based on main packet processing units, i.e., CPUs, GPUs, and FPGAs. PacketShader [31], SSLShader [33], Kargus [32], NBA [39], APUNet [29], and G-NET [55] leverage GPU to accelerate packet processing. ClickNP [43], SwitchBlade [15] offload packet processing logic to the FPGAs. RouteBricks [21], NetBricks [49] and E2 [48] propose packet processing optimization techniques to improve performance on CPUs. Besides leveraging accelerators, there has been some work optimizing state management. OpenBox [18] decouples NFV control plane and data plane, and Stateless Network Functions [34] decouple stateless processing logic and data store.

For all the NF optimization works on CPUs, GPUs, and FPGAs, clock cycles matter. With Microscope we provide a mechanism to diagnose performance issues of deployed network functions in a DAG at the granularity of hundreds/thousands of clock cycles. Now we'll discuss performance diagnostics works in NF and distributed systems domain.

Performance diagnosis in networked systems: We divide the diagnosis of networked systems into performance diagnosis for VNFs and for traditional network systems.

For VNFs, PerfSight[53] and Probius[47] diagnose persistent problems, like persistent high packet drop rate and long-term low throughput, on software dataplane. Whereas Microscope can diagnose transient (microseconds scale) service function chain performance problems. Perfsight uses packet drops and throughput numbers as indicators of bottleneck network elements. Although this is effective in identifying persistent bottleneck, it cannot diagnose problems at the tail (e.g., long tail latency, transient drops), which is a big headache to operators. There is no way to identify long latency in PerfSight, while transient drops themselves are insufficient for PerfSight to give the detailed causal diagnosis as provided by Microscope.

For traditional network systems, SCORE [41] focuses on identifying root causes of network faults across different *vertical* layers (*e.g.*, across IP layer and optical link layer), but it cannot figure out how faults propagate their impact across different network elements, *horizontally*. Sherlock [16] builds a graph to model the causal relationship between network components (*e.g.*, routers, hosts, links) and services, and use history monitoring data and time-based correlation to predict probabilities of the causal relationship. But Microscope identifies transient performance issues at smaller time granularity then Sherlock [16].

Performance diagnosis in distributed systems: There are several distributed system diagnostic tools that use statistical correlation of the logs [19, 46]. They face the same problem as NetMedic [36] faces to perform diagnosis at low granularity. Similarly, Retro [44] monitors and attributes the queuing delay to different users to quantify the usage of users, which is similar to how Microscope monitor the queue But beyond that, Microscope also derive ways to quantify problem propagation, which is one of our key contributions

Apart from diagnostics tools there are distributed tracing tools e.g, Jaegar[6], Zipkin [12] and [17, 26, 45]. They are good at understanding the causal relations between events that happen to the *same* request. However, we need to understand the causal relations between different packets, which needs to go beyond tracing. Microscope can diagnose problems across different packets at smaller timescale and then correlate that information in the aggregation stage (§4.4).

9 CONCLUSION

In this paper we presented Microscope to diagnose network performance issues. First, we showed how stringent performance requirements of VNFs can create a lasting impact across time and network functions causing latency and throughput issues for downstream VNFs. We then presented the design and the implementation of Microscope that can be used to diagnose such problems, based on the key insight of analyzing the queuing periods. This was followed by evaluation in a testbed of realistic service function chain where we used Microscope to diagnose several performance issues for realistic traffic scenarios. Specifically, we demonstrated that Microscope can diagnose performance problems caused by interrupts, software bugs, traffic bursts, resource exhaustion etc. accurately and correctly across a chain of various network functions. To the best of our knowledge Microscope is the first work that shows how microsecond level events can degrade performance several hops away, and how these problems can be diagnosed quickly with low overhead. While Microscope is not a panacea, we believe it can help operators in reaping benefits provided by virtualization while maximizing performance.

This work does not raise any ethical issues.

10 ACKNOWLEDGMENTS

We thank our shepherd Aurojit Panda and SIGCOMM reviewers for their helpful feedback. We thank Vijay Gopalakrishnan and Wei Bai for their helpful feedback. Junzhi Gong, Yuliang Li, and Minlan Yu are supported in part by the NSF grant CNS-1618138.

[5] Diagnosing distributed system is different from diagnosing networked systems. Networked systems'(VNFs, Routers etc.) packet processing has stricter per-packet latency requirement (e.g., 10 100 us) than distributed systems' per-request latency (ms level), therefore same event(e.g., interrupt) can impact larger number messages(packets) in Networked Systems than messages(requests) in distributed systems. For example, an interrupt (100s us long) may be fine in a distributed system, but it can introduce significant latency in servicing packets in networked systems. Furthermore, these smaller problems can propagate across the network(§2)

REFERENCES

[1] Brocade vyatta 5400 vrouter. http://www.brocade.com/products/all/network-functions-virtualization/product-details/5400-vrouter/index.page.

[2] The cooperative association for internet data analysis (caida). http://www.caida.org/.

[3] Data plane development kit. https://www.dpdk.org/.

[4] Evolution of the broadband network gateway. https://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2013/6756-evolution-the-broadband-network-gateway.pdf.

[5] Ieee standard 1588-2008. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4579757.

[6] Jaeger: open source, end-to-end distributed tracing. https://www.jaegertracing.io/.

[7] Microscope survey form and results. https://www.dropbox.com/s/66cp4k3wl8zm0q5/survey.pdf?dl=0.

[8] Migration to ethernet-based broadband aggregation. https://www.broadband-forum.org/download/TR-101_Issue-2.pdf.

[9] Nfv proofs of concept. http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc.

[10] Open vswitch. https://www.openvswitch.org/.

[11] Vpp. https://fd.io/.

[12] Zipkin: A distributed tracing system. https://zipkin.io/.

[13] Omid Alipourfard and Minlan Yu. Decoupling algorithms and optimizations in network functions. In Proceedings of the 17th ACM Workshop on Hot Topics in Networks, pages 71–77, 2018.

[14] Bilal Anwer, Theophilus Benson, Nick Feamster, and Dave Levin. Programming slick network functions. In Proceedings of the 1st acm sigcomm symposium on software defined networking research, pages 1–13, 2015.

[15] Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. Switchblade: A platform for rapid deployment of network protocols on programmable hardware. In Proceedings of the ACM SIGCOMM 2010 conference, pages 183–194, 2010.

[16] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. ACM SIGCOMM Computer Communication Review, 37(4):13–24, 2007.

[17] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In OSDI, volume 4, pages 18–18, 2004.

[18] Anat Bremler-Barr, Yotam Harchol, and David Hay. Openbox: a software-defined framework for developing, deploying, and managing network functions. In Proceedings of the 2016 ACM SIGCOMM Conference, pages 511–524. ACM, 2016.

[19] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In Proceedings International Conference on Dependable Systems and Networks, pages 595–604. IEEE, 2002.

[20] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. {HUG}: Multi-resource fairness for correlated and elastic demands. In 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), pages 407–424, 2016.

[21] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 15–28, 2009.

[22] Nick G Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. IEEE/ACM transactions on networking, 9(3):280–292, 2001.

[23] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), pages 523–535, 2016.

[24] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. Moongen: A scriptable high-speed packet generator. In Proceedings of the 2015 Internet Measurement Conference, pages 275–287, 2015.

[25] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pages 137–148, 2003.

[26] Rodrigo Fonseca, George Porter, Randy H Katz, and Scott Shenker. X-trace: A pervasive network tracing framework. In 4th {USENIX} Symposium on Networked Systems Design & Implementation ({NSDI} 07), 2007.

[27] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. ACM SIGCOMM Computer Communication Review, 44(4):27–38, 2014.

[28] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18), pages 81–94, 2018.

[29] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. Apunet: Revitalizing {GPU} as packet processing accelerator. In 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), pages 83–96, 2017.

[30] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. Softnic: A software nic to augment hardware. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155, 2015.

[31] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. ACM SIGCOMM Computer Communication Review, 40(4):195–206, 2010.

[32] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. Kargus: a highly-scalable software-based intrusion detection system. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 317–328. ACM, 2012.

[33] Keon Jang, Sangjin Han, Seungyeop Han, Sue B Moon, and KyoungSoo Park. Sslshader: Cheap ssl acceleration with commodity processors. In NSDI, pages 1–14, 2011.

[34] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. Stateless network functions: Breaking the tight coupling of state and processing. In 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), pages 97–112, 2017.

[35] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19), pages 345–360, 2019.

[36] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. Detailed diagnosis in enterprise networks. ACM SIGCOMM Computer Communication Review, 39(4):243–254, 2009.

[37] Rishi Kapoor, Alex C Snoeren, Geoffrey M Voelker, and George Porter. Bullet trains: a study of nic burst behavior at microsecond timescales. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, pages 133–138, 2013.

[38] Georgios P Katsikas, Tom Barbette, Dejan Kostic, Rebecca Steinert, and Gerald Q Maguire Jr. Metron:{NFV} service chains at the true speed of the underlying hardware. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18), pages 171–186, 2018.

[39] Joongi Kim, Keon Jang, Keunhong Lee, Sangwook Ma, Junhyun Shim, and Sue Moon. Nba (network balancing act): A high-performance packet processing framework for heterogeneous processors. In Proceedings of the Tenth European Conference on Computer Systems, page 22. ACM, 2015.

[40] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. ACM Transactions on Computer Systems (TOCS), 18(3):263–297, 2000.

[41] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C. Snoeren. Ip fault localization via risk modeling. In Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05, pages 57–70, Berkeley, CA, USA, 2005. USENIX Association.

[42] Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pages 71–84. ACM, 2017.

[43] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In Proceedings of the 2016 ACM SIGCOMM Conference, pages 1–14. ACM, 2016.

[44] Jonathan Mace, Peter Bodik, Rodrigo Fonseca, and Madanlal Musuvathi. Retro: Targeted resource management in multi-tenant distributed systems. In 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), pages 589–603, 2015.

[45] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. Pivot tracing: Dynamic causal monitoring for distributed systems. ACM Transactions on Computer Systems (TOCS), 35(4):1–28, 2018.

[46] Karthik Nagaraj, Charles Killian, and Jennifer Neville. Structured comparative analysis of systems logs to diagnose performance problems. In Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), pages 353–366, 2012.

[47] Jaehyun Nam, Junsik Seo, and Seungwon Shin. Probius: Automated approach for vnf and service chain analysis in software-defined nfv. In Proceedings of the Symposium on SDN Research, pages 1–13, 2018.

[48] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: a framework for nfv applications.

In Proceedings of the 25th Symposium on Operating Systems Principles, pages 121–136, 2015.

[49] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of {NFV}. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pages 203–216, 2016.

[50] Luigi Rizzo. Netmap: a novel framework for fast packet i/o. In 21st USENIX Security Symposium (USENIX Security 12), pages 101–112, 2012.

[51] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. Nfp: Enabling network function parallelism in nfv. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pages 43–56. ACM, 2017.

[52] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: efficient performance prediction for large-scale advanced analytics. In 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), pages 363–378, 2016.

[53] Wenfei Wu, Keqiang He, and Aditya Akella. Perfsight: Performance diagnosis for software dataplanes. In Proceedings of the 2015 Internet Measurement Conference, pages 409–421, 2015.

[54] Shaula Alexander Yemini, Shmuel Kliger, Eyal Mozes, Yechiam Yemini, and David Ohsie. High speed and robust event correlation. IEEE communications Magazine, 34(5):82–90, 1996.

[55] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. G-net: Effective {GPU} sharing in {NFV} systems. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18), pages 187–200, 2018.

[56] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. Parabox: Exploiting parallelism for virtual network functions in service chaining. In Proceedings of the Symposium on SDN Research, pages 143–149, 2017.