Low-Overhead Packet Loss Diagnosis for Virtual Private Clouds using P4-Programmable NICs

Soroush Aalibagi†, Mahdi Dolati‡, Sogand Sadrhaghighi†, Majid Ghaderi†
†Department of Computer Science, University of Calgary, Calgary, Canada.
‡School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
Emails:{soroush.aalibagi, sogand.sadrhaghighi, mghaderi}@ucalgary.ca, m.dolati@ipm.ir

***Abstract*—Virtual private clouds have become a huge trend because of their cost-efficiency. However, the complex and virtualized nature of clouds limits the ability of cloud tenants to pinpoint and amend performance degradation problems, such as packet drops. Existing monitoring systems are either designed for the physical network or insufficient to present the concrete reason for packet loss with low overhead. In this paper, we present a Packet Loss Diagnosis (PLD) system, a specific monitoring service designed to detect packet drops on cloud networks and report diagnosis information to tenants. PLD is based on the modern capabilities of P4 data plane programmable NICs and has a limited footprint in the network. It provides detailed information that enables tenants to locate and resolve their issues with respect to the abstraction of the services. It also meets the requirements of a monitoring system designed for large-scale multi-tenant clouds. We implemented the proposed scheme in P4 to demonstrate its viability and investigate its performance and overhead through extensive experiments and Mininet simulation. Our results show that PLD ensures full packet drop detection coverage and can notify tenants in real-time while imposing low overhead.**

***Index Terms*—Network monitoring, Packet loss, Programmable data plane, P4, Virtual networks, SmartNIC.**

I. Introduction

Background and Motivation. Cloud providers share their physical network among tenants to increase utilization and profit. They employ virtualization mechanisms to create abstracted and isolated virtual networks on top of the physical network for tenants based on their specifications of resource capacities, configurations, and policies. Tenants, in turn, can only access their allocated virtual resources, as infrastructure-related data, such as the physical network topology, is considered a corporate secret and guarded by cloud providers [1]. This abstraction of the physical network complicates debugging and diagnosis of network connectivity issues by tenants, as they can not inspect physical resources and do not know the mapping of their virtual to the physical network. Studies show that identifying the cause of connectivity issues is the bottleneck in mitigating network performance anomalies [2].

Packet drops are among the most severe events degrading the performance of cloud-based services. Recent surveys show that tenants' misconfigurations contribute significantly to the occurrence of these drops; for example, tenants are responsible for common causes of packet drops such as exceeding virtual link capacity and inserting incorrect access control rules [3]. However, tenants do not have the ability to quickly detect and resolve packet drops affecting their networks due to abstraction and virtualization mechanisms deployed by cloud providers. As a result, they may need to spend hours diagnosing the root cause of packet drops [4]. Therefore, a real-time end-to-end packet drop diagnosis service for cloud tenants can significantly improve tenants' ability to manage and operate their services efficiently.

The existing network monitoring and diagnosis tools can be divided into two general categories of network-based and host-based solutions. Most network-based solutions do not account for the virtualization of the environment (*e.g.*, SNAP [5] and SNMP [6]). Consequently, using them may weaken isolation across tenants and lead to scalability issues by imposing significant throughput overhead on virtual networks [7]. Nevertheless, deploying customized network-based tools is challenging, as network operators hesitate to use in-network applications that may affect the basic network functionality and lower the reliability [8]. On the other hand, most host-based solutions either insert probes to the network (*e.g.*, [9]) or store all the data on end-hosts and aggregate data on demand to answer network-wide monitoring queries (*e.g.*, [7]). The former suffers from bandwidth overhead, while the latter consumes the processing power that other tenants could have bought, reducing the profitability of the cloud's computing resources. More specifically, VND [7] is one of the pioneers and most relevant services to provide a diagnosis system for cloud tenants. As an host-based solution, VND mirrors and analyses traffic on end-hosts which consumes memory throughput and processing power that could have been allocated to tenants. Thus, using existing solutions in virtualized systems imposes a significant overhead that hinders their usage for a long duration to detect packet drops and collect supplementary root-cause information.

Recently, cloud providers have started deploying more capable programmable network interface cards (NICs), often called SmartNICs, to alleviate networking overheads that strain computing resources (*e.g.*, Microsoft Azure Cloud [10]). SmartNICs can implement custom packet processing functions with higher performance than software-based in-hypervisor solutions. In such deployments, tenants' virtual machines bypass the hypervisor's network stack via the Single Root I/O Virtualization (SR-IOV) [11] to directly access SmartNIC's functions, thereby alleviating the need for CPU processing. As most SmartNICs support the data plane programming language P4 [12], the availability of SmartNICs and the flexibility of the P4 language provide an opportunity to create host-based monitoring tools that neither consume servers' processing power nor add complexity to the core of the network underlying the cloud [13]. Our objective is to develop a packet drop detection and notification system as a service for cloud tenants based on the capabilities of SmartNICs at the edge of the network.

Our Work. In this paper, we present the design and evaluation of a monitoring tool capable of detecting packet drops in virtual private clouds and providing root-cause information to virtual network administrators. Our Packet Loss Diagnosis (PLD) system leverages the capabilities provided by SmartNICs on hosts to perform efficient packet drop detection and notification. Thus, unlike existing host-based tools, such as VND [7] and VTrace [3], PLD does not impose a monitoring burden on the processing capacity of hosts and avoids introducing complex functionalities into the cloud's core network.

PLD is implemented in P4 and adopts the match-action-based packet processing model for the implementation of a drop detection mechanism that covers the transient and permanent packet drops. As a result of employing P4, PLD is deployable in many targets, notably BMv2 software switch [14], Netronome SmartNIC [15], and TOFINO ASIC Switch [16].

Since tenants' configurations directly affect packet processing in SmartNICs, PLD separates the detection of drops that occur in the SmartNICs from the physical network consisting of physical switches. Accordingly, PLD generates different notification reports for these two drop categories, as tenants should only have an abstract view of the physical network. To detect packet drops in SmartNICs, PLD uses match-action tables [12] with specific match criteria and specialized actions to insert functionalities into the ingress and egress pipelines of the SmartNIC, which allows tracing packets and detecting drops as soon as they occur. When PLD detects a packet drop, the SmartNIC's egress sends a report message to a pre-specified collector server providing information about the lost packet and the drop reason. It is difficult to identify physical network packet losses since we do not have visibility into the links, and we cannot easily reconstruct original packet flows from the dropped packets. Thus, PLD leverages the programmability at both sides of a flow to diagnose packet losses that happened in the physical infrastructure. In other words, PLD keeps track of all packets of the flow in the destination SmartNIC using the sequential IDs assigned to the flow packets by the SmartNIC at the origin. This enables PLD to provide real-time reports about packet drops containing information that tenants can use to identify and troubleshoot the problem. These reports do not violate cloud principles such as tenants isolation. We conducted experiments to evaluate the impact of PLD on the tenants virtual networks. Our experiments consistently confirm that PLD meets the requirements of a monitoring system in multi-tenant clouds. The main contributions of this paper are:

- We present the design of PLD to detect **all** packet drops that occur due to tenants faults in their virtual networks or faults that are out of the control of tenants happening in the cloud's physical network.
- We design mechanisms for **root-cause information** collection and **immediate notification** about packet drops in virtualized environments.
- We implement PLD, a **low-overhead P4-compatible** solution, that is deployable on SmartNICs in hosts and is **target independent** (independent of the specifics of the underlying hardware).
- We conduct **extensive hardware experiments and Mininet simulation** to study the **feasibility and efficiency** of PLD in a variety of realistic scenarios.

TABLE I: Common packet drops caused by tenants [3].

Cause of drop	Proportion (%)
Incorrect access control list	62.4
Incorrect routing-related configuration	26.4
Insufficient bandwidth capacity	11.2

Organization. Section II provides the necessary background on virtual networks, packet drops, and requirements of packet drop diagnosis systems. Section III describes the architecture and drop detection mechanisms of PLD. We evaluate PLD in Section IV. Related works are reviewed in Section V, while Section VI concludes the paper.

II. Background

In this section, we provide an overview of cloud infrastructure and cloud tenants' virtual networks. Then, we present a discussion about packet drops that affect the performance of tenants. Finally, we list the principal requirements of packet drop detection and notification tools for cloud tenants.

A. Cloud Infrastructure

In the most basic form, cloud infrastructure consists of physical servers connected through a physical network. Servers provide computing resources such as CPU and RAM that can be rented to tenants to generate revenue. Thus, cloud providers employ hypervisors to run multiple virtual machines on physical servers, sharing their resources in isolation. Conversely, the physical network focuses on the fundamental data movement task among servers. Traditionally, hypervisors consumed CPU cycles to provide the networking services demanded by each tenant's network of virtual machines running on different physical servers (*e.g.*, load balancing). However, data plane programmability is helping to replace simple network interface cards with SmartNICs, which can provide those networking services without consuming extra CPU cycles. We consider a cloud infrastructure where all physical servers possess SmartNICs to connect to the physical network, as is the case with major cloud providers such as Amazon [17].

B. Virtual Networks

Cloud tenants use the "big switch abstraction" to specify the virtual network that connects their virtual machines. In this abstraction, all virtual machines have a direct link to a single virtual switch that forwards traffic according to the corresponding tenant's networking plan. As a result, cloud providers can implement tenants' networking policies and requirements (e.g., access control and rate limiting) entirely at the edge of their physical network. We consider a cloud provider that uses SmartNICs to implement tenants' networking requirements.

C. Packet Drop in Virtual Clouds

Cloud tenants experience packet drops caused by several reasons that negatively affect the performance of their applications. Previous studies reveal that most experienced drops

are due to tenants' misconfigurations and mismanagement [2]. Table I shows tenant-induced causes for packet drops [3]. Furthermore, the physical network may drop tenants' packets for reasons such as faulty network equipment and congestion. Although these drops are rare compared to tenant-induced drops, sensitive tenants may desire to know such packet drops to understand the performance behavior of their applications.

D. Requirements of Drop Monitoring Tools

A packet drop monitoring system must meet several criteria to be effective when deployed in a large-scale multi-tenant cloud. These requirements include:

R1: Tenant Isolation. Tenants should be isolated from other tenants and details of underlying physical infrastructure. Thus, reports of packet drops in one tenant's virtual network should not expose any information about other tenants' networks. Furthermore, these reports should not disclose physical infrastructure details to tenants.

R2: Low Overhead. A drop monitoring system should not impose overheads that hinder its usage in large-scale systems. The most critical overheads are (1) computing resources (*e.g.*, CPU, RAM, storage) overhead, (2) bandwidth overhead, and (3) latency overhead imposed to existing flows in the network.

R3: Responsiveness. Responsiveness measures the time from a packet drop incidence until the corresponding tenant receives the report about that drop. According to previous studies such as [2], responsiveness is a vital feature that most existing monitoring tools lack.

R4: Root causes information. Pinpointing the root cause of packet drops is one of the most time-consuming steps in the drop diagnosis process [2]. Thus, drop monitoring tools must generate reports containing information to help tenants resolve the drop problem on time. Consequently, generated reports may enclose the location, time, and reason for packet drops.

III. PLD Design

In this section, we present the design of PLD, our packet loss diagnosis mechanism to detect and report packet drops discussed in Subsection II-C. First, we present the architecture of PLD in Subsection III-A. Then, we discuss the procedure of detecting packet drops in Subsection III-B. We present the detection of drops in SmartNICs and the physical networks in two separate parts, as these types of drops are fundamentally different and require specialized approaches. Finally, we propose a mechanism to reduce the throughput overhead of reporting in Subsection III-C.

A. Architecture

We explain PLD's four major architectural components (illustrated in Fig. 1) in the following.

Dashboard. Dashboards provide two functionality for tenants. First, tenants use the dashboard to submit monitoring tasks. Each task involves defining a set of filters that determine a subset of traffic in the virtual network for closer inspection.

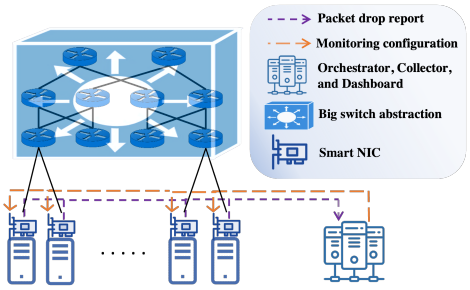

Fig. 1: The architecture of PLD.

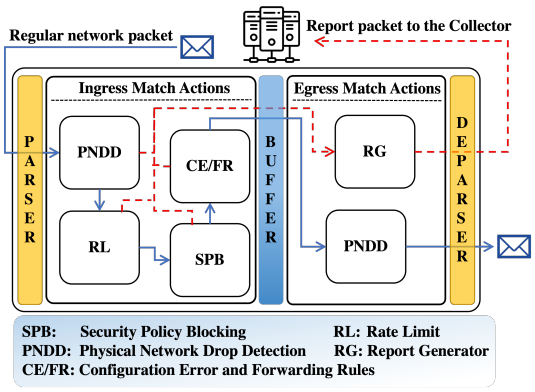

Fig. 2: Components of PLD in a SmartNIC.

Each filter is a condition on a subset of packet header fields. For example, the tenant can inspect the traffic of a web server. Second, tenants use the dashboard to see the monitoring reports. Specifically, the report shows the time, reason, and location of the drop along with the dropped packet's 5-tuple.

Orchestrator. The orchestrator uses its global network view to determine the set of SmartNIC functionalities to carry out the submitted monitoring tasks. Then, the orchestrator sends instructions to those SmartNICs to achieve its goal. Each instruction specifies a set of match rules and a set of actions.

SmartNIC. Each SmartNIC uses customized match-action tables and extern objects to detect and report packet drops, which happen in virtual networks, while enforcing security policies, switching packets, and limiting rates. These operations correspond to the listed causes of packet drop in Table I. Also, SmartNICs run an end-to-end mechanism inspired by the inter-switch packet drop discovery idea of [2] to detect packet drops in the physical network. Then, the SmartNIC reports any packet drops in the virtual network to the collector server along with their root cause information for tenants. Reports of packet drops in the physical network contain less information than those occurring in SmartNICs.

Collector. Collectors ingest monitoring data from SmartNICs. Note that the orchestrator carefully selects the destination collector of each SmartNIC to lower the overall overhead and increase the responsiveness, facilitating Requirement **R3**. Then, the collector can build and send appropriate reports to tenants.

B. Mechanisms

We present the design of PLD's components for detecting and reporting packet drops in this subsection. Fig. 2 rep-

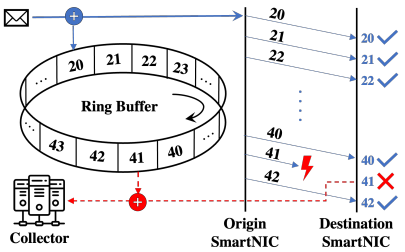

Fig. 3: Physical network packet drop detection model.

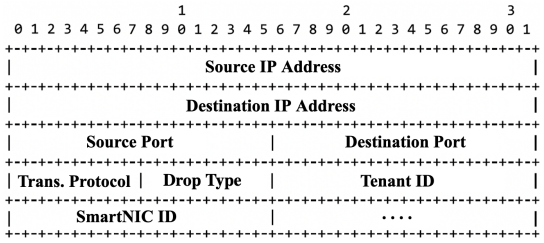

Fig. 4: PLD's report packet header.

resents a schematic of these components inside a generic programmable SmartNIC.

Drops in Virtual Network. We use three components to handle the detection of packet drops based on three most important causes of packet drops in a virtual network (Table I). When a packet arrives at the SmartNIC, it goes through components *RL* (Rate Limit), *SPB* (Security Policy Blocking), and *CE/FR* (Configuration Error and Forwarding Rules). Each SmartNIC first uses a table to check whether or not a packet belongs to a flow that is monitored for a tenant. The SmartNIC stores the result of this step as a metadata and uses it throughout the processing of the packet. Then, the packet is processed by the *RL* component, employs trTCM [18] extern as a Rate Limiter in the same manner as [19], [20]. Externs are architecture-specific constructs that can be manipulated by P4 programs via standard APIs, but whose internal behavior is hardwired. If trTCM decides to drop the packet due to rate limit violation, the *RL* creates a metadata for the packet to show that the packet need to be dropped as the result of rate limiting mechanism. The behavior of *SPB* and *CE/FR* components are similar to RL, except that they use regular P4 tables to implement their logic instead of using an extern object. For example, in order to apply ACL rules, PLD incorporates the table-based ACL function in the *SPB* component similar to [21]–[23]. Specifically, if any of the three aforementioned components decides the packet cannot reach its destination, the corresponding component calls the relevant action to remove the packet from the regular network packet flow (blue line in Fig. 2) by changing the destination address and redirecting the packet to the collector. Then, the *RG* (Report Generator) component is triggered with the appropriate drop reason code. Subsequently, the egress pipeline sends a report to the collector for packets with the special metadata.

Drops in Physical Network. Fig. 3 demonstrates a schematic of the procedure of the *PNDD* (Physical Network Drop Detection) component. In this figure, packets are sent from the origin SmartNIC and reach the destination SmartNIC after going through the physical data center network. To reduce bandwidth overhead and satisfy **R2**, *PNDD* only generates traffic when a drop occurs in the physical network. Specifically, first, the egress *PNDD* in origin SmartNIC stores the packet 5-tuple and a 2-byte sequential number, as the packet ID, in a ring buffer, which is stored in the SmartNIC on-chip memory. The sequential packet ID and the index of the packet's location in the ring buffer increment by one for each consecutive packet. Then, the egress *PNDD* attaches 4 bytes of data to the packet. This data includes the 2-byte packet ID and the 2-byte index number corresponding to the packet's location in the SmartNIC's ring buffer. To attach these 4 bytes of data, we can take advantage of existing unused bits in packets, such as IP options [2], or use a custom header. Then, the ingress *PNDD* in the destination SmartNIC detaches the packet ID and the index number and forwards it to its destination. Meanwhile, if *PNDD* in the ingress pipeline of the destination SmartNIC realizes a nonconsecutive packet ID, it sends a packet containing the missed packet ID and index number to the origin SmartNIC. The ingress *PNDD* in origin SmartNIC redirects the packet to the collector and triggers the *RG* component. The *RG* fetches the corresponding 5-tuple using the index number of the dropped packet and generates a report similar to reports for the virtual network drops, except that it does not include the exact location of the drop in the physical network.

Due to the size of the ring buffer, this mechanism has a certain capacity to detect packet drops. More accurately, when the destination SmartNIC reports a dropped packet to the origin SmartNIC, the dropped packet in origin SmartNIC's ring buffer may have already been replaced with a new packet that arrived at the origin SmartNIC due to insufficient ring buffer size. In this case, the origin SmartNIC is not able to report the dropped packet. Note that PLD does not mistakenly report the new packet that has overwritten the dropped packet since we can distinguish the two packets using their packet IDs. Actually, a 2-bytes Packet ID as well as the 2-bytes index number provide over 4 billion ($2^{4\times8}$) different combinations that our mechanism can distinguish. Therefore, the ring buffer should at least accommodate each packet for twice the propagation time between the origin and the destination SmartNICs. Today, a typical data centre network has a high traffic rate, and propagation delay is in the order of microseconds [24], [25]. Considering a $100Gbps$ traffic speed, $100\mu s$ as propagation delay, 175B as the average data center packet sizes [26], and the fact that we only store 15 bytes for each packet, the ring buffer needs at least $215KB$ memory to accommodate at least 14286 packets. Hence, this memory requirement plus a 2-byte variable as a buffer index is feasible for a SmartNIC.

Drop Reports. The *RG* (Report Generator) in the egress pipeline is responsible for producing the report for the collector. These reports contain the dropped packet's 5-tuple along with the reason and location of the dropping incident, satisfying **R4**. Since reports are sent to the collector immediately after detection, there is no need to include a timestamp in the reports. Instead, collectors timestamp incoming report packets to save the network bandwidth. *RG* generates reports

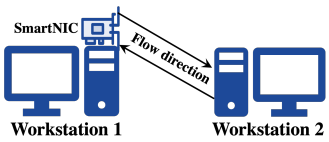

Fig. 5: Testbed Topology.

by first truncating the packet's payload and adding an 18 bytes custom header to store the complementary information. Fig. 4 shows the template of the custom header. Note that protocol-independent forwarding enabled network devices can forward packets with custom headers [12]. The first 13 bytes in the header are the dropped packet 5-tuple. The next byte indicates the type of drop obtained from the metadata set by a component in the ingress pipeline. Then, the header holds a 2-bytes tenant ID (to facilitate **R1**) and a 2-bytes SmartNIC ID. Note that although we use 2-byte IDs for both SmartNICs and tenants since we do not have more than 65000 SmartNICs and tenants in our experiments, PLD is configurable for larger fields. Collectors use the tenant IDs and SmartNIC IDs to infer the location of the packet drop with respect to the tenant's virtual network. Finally, *RG* can send multiple copies of the same report packet to handle the loss of reports in the physical network [2].

C. Bandwidth Optimization

Since the minimum Ethernet packet size is 64 bytes, PLD sends one 64 bytes report packet for every dropped packet. However, the content of each report is only 18 bytes, leading to more than 70% more bandwidth overhead than the report's content. It is possible to reduce the bandwidth consumption of PLD by using a configurable report batching method. To this end, we attach the reports at the end of a packet continuously circulating the SmartNIC pipeline via a separate internal port. Then, the collected reports are forwarded to the collector in one packet after a temporal or spatial threshold.

IV. Performance Evaluation

We present the evaluation of PLD in this section using a hardware testbed as well as a larger scale Mininet emulation. Subsection IV-A describes the testbed and Mininet setups. Then, in Subsection IV-B, we validate the functionality of the proposed scheme by showing its ability to detect drops in physical and virtual networks. Next, we evaluate the performance of PLD in terms of bandwidth, throughput, and delay overhead in Subsection IV-C. Finally, in Subsection IV-D, we compare PLD's effectiveness with the packet loss diagnosis method in VND [7], one of the most relevant virtual network diagnosis systems.

A. Environment Setup

Testbed Setup. We prototyped PLD on a small hardware testbed consisting of two Dell workstations running Ubuntu 18.04 LTS with kernel version 4.15 (Fig. 5). Each workstation has an Intel Core i7-6700 3.40GHz CPU and a 8GiB memory. The first workstation is equipped with a Netronome Agilio CX Dual-Port 10 Gigabit Ethernet SmartNIC, and the second workstation has two generic 1000Mbps NICs. We connected both physical ports of the SmatrNIC on the first workstation to the two NICs on the second workstation using 1000Mbps Ethernet cables. In our experiments, the source and destination of the traffic flows are always the dual-port SmartNIC. Actually, the SmartNIC can be configured to expose two virtual interfaces. The second workstation is used to generate packet loss in the physical network. Specifically, we use the python *Scapy* and *Random* libraries in the second workstation to forward the traffic and drop packets uniformly. The two workstations are synchronized using Chrony, an implementation of Network Time Protocol.

We have implemented the PLD system on the Netronome SmartNIC with ∼200 lines of code in addition to the SmartNIC P4 program without a packet drop detection system. We used Programmer Studio, the Windows-based SDK IDE, to develop the P4-16 [27] code and run the generated Network Flow Processor (NFP) firmware on the SmartNIC's microengines. The match+action tables are developed in JSON format. The runtime environment accepts commands to modify the tables. Our complete P4 application consists of less than 450 lines of code. It consumes 32 bytes of SmartNIC memory for virtual network drop detection and less than 250KB ring buffer for detecting physical network drop detection. We implemented orchestrator, dashboard, and collector in python as a virtual machine image.

Flow sizes in our experiments are less than 10KB, following common traffic statistics in the Facebook data center [26]. We chose the packet sizes to be 175B and 850B, which are the average data center packet sizes reported by [26] and [28], respectively. Also, as studied by [9], the packet drop rate in our experiments falls between 10^{-4} and 10^{-5} unless otherwise stated. Also, the report batching mechanism (see Subsection III-C) is disabled to show PLD's maximum overhead. Moreover, a combination of numerical and literal IDs is assigned to the packets to make tracking easier.

Mininet Setup. For the larger scale simulation, we implemented a 4-pod fattree topology with 20 switches and 16 servers in Mininet. We compiled our P4 code with the p4c compiler [29] for the behavioral model (bmv2) [14] to emulate SmartNICs in our data center network. The default queue size in bmv2 switches is 64 packets. We also set the links' bandwidth to 100MB/s. The details about flows, packets, and drops are the same as the hardware setup.

B. Functional Validation

In this subsection, we validate the functionality of PLD by investigating its coverage of packet drops in the physical network and virtual network:

Packet Loss Detection in Physical Networks.

Setup: In this scenario, we show the functionality of PLD in detecting physical network packet losses. We consider three tenants in the simulated data center. Two tenants generate an excessive amount of traffic to congest a core switch and simulate congestion-induced packet drops [2]. Specifically, we

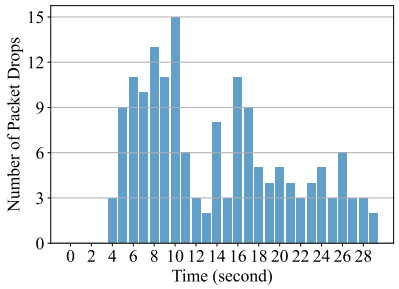

Fig. 6: End-to-end packet drop in physical network.

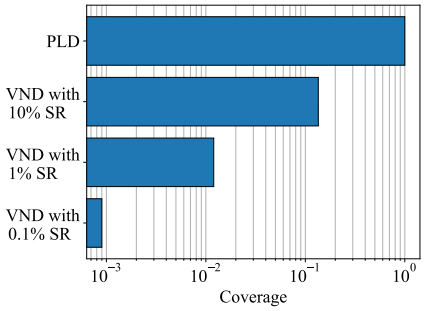

Fig. 7: Coverage comparison between PLD and VND with different Sampling Rates (SR) in virtual network.

```
Report 5
   Device ID:          2
   Reason:             Rate limit
   Source Addr.:       192.168.1.4
   Destination Addr.: 192.168.4.4
   Drop Time:          13:04:50

Report 17
   Device ID:          2
   Reason:             Routing table miss
   Source Addr.:       192.168.1.4
   Destination Addr.: 199.168.4.4
   Drop Time:          13:05:11
```

Fig. 8: PLD packet drop reports.

decrease the core switch queue rate (rate of sending packets through the egress queue) to 30 packets per second in order to saturate the bmv2 switch. Then, two tenants send packets at 56Kbps passing through the specific core switch. The third tenant also sends traffic from an IDS to a web server via the congested core switch. This tenant does not see any packet drop because of his fault (i.e., packet drops listed in Table I) on his dashboard. In the following, we present the result of using PLD between the IDS and the web server.

Results: Fig. 6 shows the number of packet drop reports corresponding to the packets sent from the IDS to the web server that the victim tenant received. Specifically, Fig. 6 illustrates a thirty seconds traffic monitoring session between the tenant's IDS and Web Server. This report indicates an average of 6 packet drops after the fourth second from the start of the simulation when the mentioned core switch gets congested. The fluctuations that appeared in this time interval are because of the non-deterministic behavior of the tenants' traffic and the sequence of receiving different tenants' packets by the switches in the physical network. Overall, the tenant finding out the physical network suffers from a significant packet loss can report the abnormality to the cloud administrator.

Packet Loss Detection in Virtual Networks.

Setup: In this scenario, the coverage of PLD is examined (on the hardware testbed) when various kinds of virtual network packet drops are injected into the network. We compare the coverage against the VND [7] mechanism when sampling is in effect (VND+sampling) to control the performance. Briefly, VND mirrors problematic traffic flows in the network. Afterwards, mirrored traffic stored on dedicated servers is examined to identify packet losses between hops. In this experiment, we consider two tenants sending traffic through the SmartNIC. The first tenant does some misoperations, resulting in the three types of packet drops mentioned in Table I. The tenant's dashboard running on the source machine should receive the monitoring reports. The second tenant traffic is background traffic to investigate isolation between tenants. For VND+sampling approach, we select packets uniformly in the source machine. Then, we check if they arrived at the destination machine using packet IDs.

Results: Fig. 7 demonstrates PLD coverage against VND+sampling. Our proposed mechanism successfully detects 100% of the injected packet drops, while VND+sampling fails to detect most of the drops. Although sampling 10% of the traffic requires data movement such that it consumes considerable bandwidth and processing power [7], it could not detect more than 15% of the packet drops.

Fig. 8 represents two of the reports shown to the first tenant. These reports are shown in the tenant's dashboard in real-time and without needing the tenant's action. Such behavior demonstrates PLD addresses **R3**. This is because our model does not need to mirror the traffic to detect such packet drops, unlike traditional systems (*e.g.*, VND). Moreover, the second tenant does not get any reports since there is no packet drops happening in his virtual network. Thus, PLD satisfies **R1**. The first line in the reports, presented in Fig. 8, shows the exact location of the packet drop, which is the ID number of one of the tenant's machines. This *Device ID* is integrated with the logical view of the tenant's virtual network and does not reveal any details about the cloud physical infrastructure. Note that by combining network topology information with the SmartNIC ID, the collector can infer the mapped location of the culprit device in the tenant's network. The reports also show that the reason for the packet drops are that the tenant exceeded his bandwidth limit and the tenant made a mistake in the addressing, respectively. Providing such information, as well as the dropped packet's 5-tuple information, confirms that PLD can provide root cause information mandated by **R4**.

C. Overhead Analysis

Here, regarding **R2**, we perform multiple benchmarks in our testbed to investigate the performance of PLD from multiple aspects, such as bandwidth, throughput, and end-to-end delay.

Bandwidth.

Setup: In this experiment, we connect the two physical ports of the dual-port SmartNIC to each other using a 10GiB SFP, creating a loopback, to test bandwidth overhead at a high rate. We also assume that 5% of the packets are dropped due to the tenant's misoperations. In real-world scenarios, the bandwidth overhead is negligible if the packet loss rate is rare. We configured such high percentages to assess our system's bandwidth overhead in worst-case scenarios. The two sources of overhead in PLD are due to the 4 bytes of data we attach to packets to detect drops in the physical network and the report

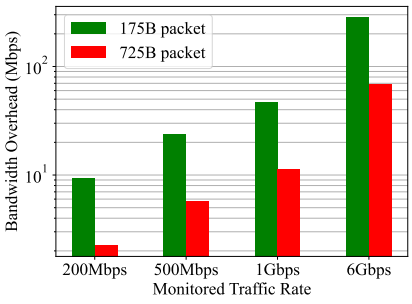

Fig. 9: Bandwidth overhead of PLD for different amounts of traffic volumes and average packet sizes.

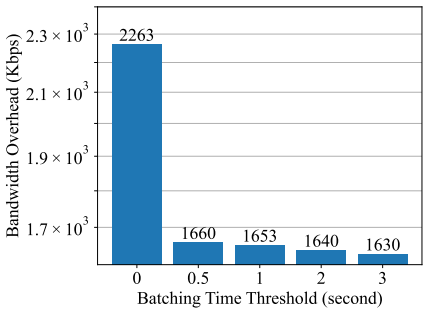

Fig. 10: Bandwidth overhead of PLD for 200Mbps traffic and 725B packets and different temporal thresholds for batching time.

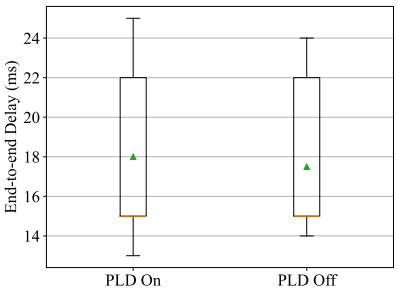

Fig. 11: Effect of PLD on the end-to-end delay.

packets that get forwarded to the collector. Note that, for each drop event, PLD sends a 64B report packet to the collector.

Results: Fig. 9 shows the bandwidth overhead of our system for different average packet sizes and amounts of monitored traffic. Due to the constant drop rate, the bandwidth overhead increases as traffic rate increases. Likewise, it is evident that with a fixed traffic volume, a smaller packet size (*i.e.*, 175B) means more packets are transmitted and dropped in the network. Thus, more drop reports result in more traffic overhead. Yet, even for such a high drop rate, the average packet size of $175B$, and 6Gbps traffic, PLD has 281Mbps overhead, which is less than 4.7% bandwidth overhead.

We also evaluated PLD when the report batching mechanism is active. In this experiment, we use the same loss rate described above and only focus on 200Mbps traffic rate with 725B average packet length. We used different temporal thresholds in Fig. 10 to compare the resulting overhead. The zero batching time threshold is identical to the system with no report batching mechanism in use. As shown, the overhead is significantly reduced at any threshold within the range $[0.5, 3]$ (seconds). The results show a 500ms threshold leads to $\sim$30% bandwidth overhead reduction. Also, a 3s threshold does not make more than a 2% improvement over the 500ms threshold. Hence, we can apply smaller temporal thresholds to maintain real-time reporting while reducing overhead.

Throughput. Using P4 programming data plane provides line-rate throughput [2], [30]. In this experiment, we want to show that PLD does not affect the tenant traffic throughput. We used *iperf* to generate traffic between our two workstations and measure the throughput. We also configured one tenant in the network to inject common packet loss mentioned in Table I. The results show that the maximum throughput is always 944Mbps in presence and absence of PLD, which means our proposed system does not decrease the throughput.

End-to-end Delay. In addition to throughput, we expect that end-to-end delay in the above experiment does not get affected by our system. Fig. 11 also shows no significant difference between end-to-end delays. This is because leveraging P4 data plane programming guarantees line-rate performance. The partial difference between plots is due to different traffic generation behavior. Overall, the influence of PLD system is negligible on the network.

TABLE II: Comparison of PLD with ProActive VND.

Comparison criteria	PLD	ProActive VND
Coverage	%100	%100
Report granularity	root cause info.	only detection
Memory overhead	4.61 GiB	3.73 GiB
Responding time	23ms	2379ms

D. Comparison with ProActive VND Packet Loss Detection

Setup: Both VND and PLD intend to provide a monitoring system for cloud tenants. VND is a passive method. That means tenants need to submit their diagnosis requests before VND attempts to collect the flow traces. So, it cannot actively detect packet drops. Hence, we modified VND to actively collect traffic flows and inspect them for packet losses over time intervals of length τ. The packet drop detection mechanism in the modified VND, referred to as ProActive VND, dumps packet traces on the source and destination of a flow and executes the packet loss query on the source machine. We tested the ProActive VND with $\tau = 1s$. We also used the *scapy* library and *iperf* to generate an end-to-end flow with 900Mbps traffic rate between two hosts.

Results: As shown in Table II, although ProActive VND could detect all packet drop events, it was not able to report the root cause information by design. Moreover, the most significant difference between ProActive VND's packet drop diagnosis approach and PLD is that ProActive VND employs computing, storage, and memory resources in the tenant's virtual machine that hinders its performance and scalability, considering the current growing cloud traffic. To clarify, VND fails to respond to the tenants' requests when there are not enough resources left in the tenants' machines. However, we only utilize the programmability of the current widely-used SmartNICs in data centers and do not consume any resources in the tenants' machines. More specifically, one of the challenges regarding VND overhead is their memory consumption. Although the physical servers are equipped with more powerful processing units, all dumped traffic by VND needs to go through the memory.

In our experiment, we run the Linux *mbw* benchmark on the source side of the end-to-end flow, where the reports are

created and shown to the tenant. We set the *mbw* array size to 2048MiB and call *memcpy()*. Results show that ProActive VND decreased the memory throughput from 4.61GiB to 3.73GiB. The resulting reduction in memory throughput creates a considerable bottleneck for large multi-tenant cloud networks. Implemented on SmartNICs, PLD does not suffer from such bottlenecks, but we only consume negligible resources on the SmartNIC, leaving resources for other possible applications (*e.g.*, offloading network-traffic encryption). Furthermore, the responding time comparison reveals that PLD outperforms ProActive VND in reporting packet loss to the tenants. The responding time shows the average time from a packet dispatched from a source host until the tenant's dashboard displays the loss report to the tenant. The responding time in ProAvtive VND includes not only the time packets need to wait for inspection (which on average takes 500ms) but also the time it takes for the traces in the destination to be moved to the source host (23ms) and be joined to source traces by packet ID (1856ms).

V. Related Work

Although there are many works on network monitoring in the networking literature, there are few works proposing a virtual network diagnosis system dedicated to tenants. We organize our discussion into two categories of network-based and host-based monitoring. In general, network-based monitoring solutions require infrastructure access, while host-based monitoring solutions impose substantial computation and communication overhead on host servers.

Network-based Monitoring. Anteater [31] diagnoses network connectivity issues by checking static analysis of the data plane. However, Anteater, and similar approaches such as [32], [33], are not suitable for multi-tenant clouds since they can violate isolation across tenants as they expose raw network information about the infrastructure. NetSight [34] is a platform that captures packet histories and enables applications to retrieve packet histories of interest. In particular, it sends a postcard containing information about every packet that traverses network switches to a remote controller. The controller then uses the collected postcards to diagnose different network failures. While NetSight captures postcards about all traversing packets, our approach only sends a message when a drop occurs in the network. NetSeer [2], similar to our work, detects and reports packet drops in a programmable data plane. However, it tracks packet drops in physical links, which makes it more suitable for monitoring the physical infrastructure than tenants' networks. LossRadar [4] detects packet losses by employing bloom filters deployed on switches and forwards the collected information to a central controller without giving details regarding the drop reasons. It also needs specialized hardware support. LightGuardian [35] is a bandwidth-efficient network traffic measurement tool that takes advantage of sketches to capture flow-level information. SyNDB [36] stores packet-level information in the SRAM of programmable switches to perform transient faults root-cause analysis. LightGuardian and SyNDB require expensive programmable switches to deploy their mechanisms.

Host-based Monitoring. Pingmesh [9] measures network latency by injecting probes into the network. But, the transmission and storage overheads of the probes limit its effectiveness [37]. Similarly, SDN traceroute [38] uses a probing mechanism to pinpoint the location of network anomalies. However, neither of these works can diagnose the root cause of packet losses. Other works, such as [39] and [40], utilize information collected from end-host machines and virtual devices to diagnose network failures. Unlike PLD, These works require tenants to get involved in the diagnosis process. SIMON [41] uses LASSO, a machine learning algorithm, to infer the network state variables such as queuing times at switches, link utilization, and queue and link compositions at the flow level in data centers. VND [7] is a pioneering work for monitoring virtual networks in clouds. VND allows tenants to submit diagnosis requests to VND controller servers. Then, VND mirrors traffic on host servers and stores it in dedicated servers. The stored traffic is analyzed to answer tenants' requests. Mirroring and storing flows impose an overhead that limits the applicability of VND in the face of rapidly growing data center traffic. VTrace [3] automatically diagnoses packet loss over the cloud-scale virtual networks. It automates end-to-end traffic fluctuation analysis in the multi-tenant cloud network and determines the root cause. VTrace uses a series of "coloring, matching, and logging" rules in virtual forwarding devices to selectively inspect packets of interest. One drawback of VTrace is its incapability to detect transient packet drops. Additionally, VTrace is unsuitable to be applied as an always-on service because of performance degradation issues. The reason is that VTrace relies on many statistical logs to reconstruct the true forwarding path of lost packets, which can result in performance bottlenecks [42].

VI. Conclusion

In this paper, we designed and evaluated PLD, a monitoring service to detect and report packet drops in cloud networks. PLD is a P4-programmable SmartNIC-based service that differentiates drops based on their occurrence location, *i.e.*, physical infrastructure or virtual network. PLD provides tenants with detailed root cause information about virtual network drops by sending near-real-time reports. For drops that happen in the physical network, PLD offers sufficient information to tenants to help them share the problem with the cloud provider without jeopardizing the isolation and abstraction requirements in cloud environments. We implemented PLD in P4 and evaluated it through extensive testbed and Mininet simulation measurements. The results show that the proposed mechanism fulfills the requirements needed for a virtual network monitoring system. Observations indicate our solution is effective without imposing significant overhead. An interesting extension of PLD is to allow tenants to define queries to collect information about network events other than packet drops, and to present PLD as a library.

REFERENCES

[1] P. Hasselmeyer and N. d'Heureuse, "Towards holistic multi-tenant monitoring for virtual data centers," in *IEEE/IFIP Network Operations and Management Symposium Workshops*, Apr 2010.

[2] Y. Zhou *et al.*, "Flow event telemetry on programmable data plane," in *Proc. ACM SIGCOMM*, Aug 2020.

[3] C. Fang *et al.*, "VTrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network," in *Proc. ACM SIGCOMM*, Jul 2020.

[4] Y. Li *et al.*, "Lossradar: Fast detection of lost packets in data center networks," in *Proc. ACM CoNEXT*, 2016.

[5] M. Yu *et al.*, "Profiling network performance for multi-tier data center applications," in *Proc. USENIX NSDI*, Mar 2011.

[6] J. D. Case *et al.*, "Simple network management protocol (SNMP)," Tech. Rep., 1989.

[7] W. Wu *et al.*, "Virtual network diagnosis as a service," in *Proc.ACM SOCC*, Oct 2013.

[8] T. Wang *et al.*, "Multitenancy for fast and programmable networks in the cloud," in *Proc. USENIX HotCloud*, Jul 2020.

[9] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM SIGCOMM*, Aug 2015.

[10] A. Caulfield *et al.*, "Beyond SmartNICs: Towards a fully programmable cloud," in *Proc. IEEE HPSR*, June 2018.

[11] S. Radhakrishnan *et al.*, "{SENIC}: Scalable {NIC} for {End-Host} rate limiting," in *Proc. USENIX NSDI*, Apr 2014.

[12] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, 2014.

[13] R. Kundel *et al.*, "P4-bng: Central office network functions on programmable packet pipelines," in *Proc. IEEE CNSM*, Oct 2019.

[14] "Github - p4lang/behavioral-model," https://github.com/p4lang/behavioral-model, (Accessed on 06/22/2022).

[15] "Smartnic overview - netronome," https://www.netronome.com/products/smartnic/overview/, (Accessed on 08/11/2022).

[16] "Intel® tofino™ series programmable ethernet switch asic," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, (Accessed on 08/11/2022).

[17] "Aws nitro system," https://aws.amazon.com/ec2/nitro/, (Accessed on 09/28/2022).

[18] J. Heinanen and R. Guerin, "RFC2698: A two rate three color marker," 1999.

[19] C. Chen *et al.*, "QoSTCP: Provide consistent rate guarantees to tcp flows in software defined networks," in *Proc. IEEE ICC*, Jun 2020.

[20] Y.-W. Chen *et al.*, "P4-enabled bandwidth management," in *Proc. IEEE APNOMS*, Sep 2019.

[21] A. Sivaraman *et al.*, "Dc. p4: Programming the forwarding plane of a data-center switch," in *Proc. ACM SOSR*, Jun 2015.

[22] M. Bonfim *et al.*, "A real-time attack defense framework for 5g network slicing," *Wiley Software: Practice and Experience*, vol. 50, no. 7, 2020.

[23] J. Liu *et al.*, "P4v: Practical verification for programmable data planes," in *Proc. ACM SIGCOMM*, Aug 2018.

[24] R. Mittal *et al.*, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, 2015.

[25] M. Alizadeh *et al.*, "Data center tcp (dctctp)," in *Proc. ACM SIGCOMM*, Aug 2010.

[26] A. Roy *et al.*, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, Aug 2015.

[27] "Specifications - p4 language consortium," https://p4.org/specs/, (Accessed on 06/22/2022).

[28] T. Benson *et al.*, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, 2010.

[29] "Github - p4lang/p4c: P4_16 reference compiler," https://github.com/p4lang/p4c, (Accessed on 06/22/2022).

[30] P. Benáček *et al.*, "Line rate programmable packet processing in 100gb networks," in *Proc. IEEE Field Programmable Logic and Applications*, Sep 2017.

[31] H. Mai *et al.*, "Debugging the data plane with anteater," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2011.

[32] P. Kazemian *et al.*, "Header space analysis: Static checking for networks," in *Proc. USENIX NSDI*, Apr 2012.

[33] N. Foster *et al.*, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, no. 9, 2011.

[34] N. Handigol *et al.*, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. USENIX NSDI*, Apr 2014.

[35] Y. Zhao *et al.*, "LightGuardian: A full-visibility, lightweight, in-band telemetry system using sketchlets," in *Proc. USENIX NSDI*, Apr 2021.

[36] P. G. Kannan *et al.*, "Debugging transient faults in data centers using synchronized network-wide packet histories," in *Proc. USENIX NSDI*, Jul 2021.

[37] L. Tan *et al.*, "In-band network telemetry: A survey," *Elsevier Computer Networks*, vol. 186, no. 1, 2021.

[38] K. Agarwal *et al.*, "Sdn traceroute: Tracing SDN forwarding without changing network behavior," in *Proc. ACM HotSDN*, Aug 2014.

[39] A. Khurshid *et al.*, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. USENIX NSDI*, Aug 2013.

[40] A. Roy *et al.*, "Passive realtime datacenter fault detection and localization," in *Proc. USENIX NSDI*, Mar 2017.

[41] Y. Geng *et al.*, "SIMON: A simple and scalable method for sensing, inference and measurement in data center networks," in *Proc. USENIX NSDI*, Feb 2019.

[42] L. Tan *et al.*, "A packet loss monitoring system for in-band network telemetry: Detection, localization, diagnosis and recovery," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, 2021.