Enabling Fine-Grained Packet Loss Monitoring in Cloud Networks

Rohan Bose *†, German Sviridov*, Jorge Cardoso*‡
*Huawei Munich Research Center – first.last@huawei.com
†Technische Universität Dresden, Germany
‡University of Coimbra, Portugal

GLOBECOM 2023 - 2023 IEEE Global Communications Conference | 979-8-3503-1090-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/GLOBECOM54140.2023.10436938

***Abstract*—Excessive network packet loss is one of the strongest symptoms of the presence of some hardware or software-related infrastructure anomaly. As such, multiple techniques to rapidly and efficiently measure packet loss in physical network equipment have been developed throughout the years. Yet, most of the proposed techniques fall short in the case of cloud scenarios which combine the presence of physical and virtual network devices.**

In this paper, we tackle the problem of providing lightweight and fine-grained packet loss monitoring at virtual switches. We achieve our goal by combining packet coloring, with efficient packet loss signal extraction and aggregation entirely within the virtual switch level. To understand its feasibility in a production environment, the proposed system has been implemented and evaluated in a synthetic scenario and for real-world use cases. Our implementation on top of OvS-DPDK shows that the proposed system achieves 95% accurate packet loss measurement while introducing negligible switching throughput degradation.

I. Introduction

Nowadays, big cloud deployments may include thousands of physical and millions of virtual devices within their infrastructure. With such a scale, infrastructure faults become a daily occurrence that has to be rapidly detected and mitigated by cloud providers. Network monitoring has been widely used for such a task as it can span from application layer monitoring [1] down to the monitoring of fine-grained hardware components (e.g., optical transceivers) [2].

Packet loss is among the most important metrics monitored by cloud providers. Typically, high in-network packet loss is one of the main symptoms of severe infrastructure problems. While persistent packet loss detection can be achieved at scale with traditional techniques, transient or bursty packet loss is notably complex to detect and debug. Due to these difficulties, transient packet loss may persist for hours or even days before it is eventually detected and fixed which in turn makes it among the main contributors to SLA deterioration for most of the cloud customers. While there exist multiple packet loss monitoring techniques, most of the previous research effort focuses on monitoring hardware network devices and equipment [3], [4], [5]. Yet, the majority of networking services in the cloud are nowadays deployed as virtualized software components within general-purpose servers. When trying to apply traditional network monitoring technologies developed for hardware network equipment to virtual ones, one would inevitably meet issues related to resource exhaustion and performance degradation [6]. This is mainly due to the limited resources dedicated to virtual network devices and the lack of specialized hardware.

Open vSwitch (OvS) [7], and, specifically, its Data Plane Development Kit-based (DPDK) counterpart, namely OvS-DPDK, has been a standard for implementing virtual switches in most of the public and private cloud deployments. While there have been multiple proposals to provide monitoring functionality to OvS, some problems still remain unsolved. Among these is the possibility of performing fine-grained and transient packet loss monitoring at the virtual network layer.

In this paper, we propose a fine-grained packet loss monitoring system tailored to work on top of the OvS-DPDK. The system uses packet coloring to try to infer the number of packets lost during the data transfer over the network, or within the internal OvS pipeline. The system is built to work within the OvS-DPDK data path, which permits to monitor all of the transient traffic, thus enabling the possibility of detecting packet loss at the packet scale. Indeed, our system, as opposed to providing full network coverage [5], aims at helping cloud providers in understanding when and where exactly particular flows experience excessive packet drop. From our experience this information is crucial to debug cloud customers' issues and to pinpoint eventual anomalies related to tenants' applications [1] in a non-intrusive and transparent way.

The rest of the paper is organized as follows: in Sec. II we overview existing work in the field of packet loss monitoring. In Sec. III we describe key concepts that enable packet loss monitoring in virtual networks. We describe the architecture of the proposed system in Sec. IV and evaluate our prototype in Sec. V. Finally, we discuss limitations and future work in Sec. VI, and draw our conclusions in Sec. VII.

II. Related Work

This section provides a brief overview of the related works in the field of packet loss monitoring.

VTrace [8] enables *persistent* packet loss detection for overlay networks. The solution to detect and analyze packet loss relies on packet coloring. For each colored packet, virtual switches generate a *per-packet-based* log message that is forwarded to a centralized data processor. Similarly, the work

TABLE I: Comparison of features for existing packet loss monitoring solutions

Related work	Measurement type	Data analysis	Log data to controller overhead	Virtual switch support
[8]	Passive	Centralized	High	Yes
[3]	Active	Decentralized	Low	No
[4]	Passive	Decentralized	Low	No
[10]	Passive	Centralized	Moderate	No
[5]	Active	Centralized	Low	No
[11]	Passive	Centralized	Low	No
This work	**Passive**	**Decentralized**	**Low**	**Yes**

in [4], utilizes packet coloring but only applies to segment routing-based networks [9].

[3] uses active probing combined with In-band Network Telemetry (INT) to detect packet loss and localize failures. Each switch adds an INT header containing path information to the probe packets which are ultimately collected and analyzed at a centralized analyzer.

LossSight [10] combines the approaches from [8], and [3] by using an ad-hoc INT header to color packets. Each switch sends per-packet-based logging events to a centralized controller where the data is processed to extract information about the location and extent of the packet loss.

The authors of [5] propose link-level probe-based monitoring for software-defined networks. It is based on the principle of covering all links in the network through intelligent analysis of selecting the network probe paths. While addressing issues related to test coverage and data overhead, [5] is only capable of providing coarse-grained per-link-based packet loss statistics.

LossDetection [11] is a packet loss monitoring solution that exploits TCP, and UDP protocol-related control signal data. This information is used to extrapolate the degree of packet loss. While providing a compact and memory-efficient data structure for packet loss detection, the approach assumes to be able to access the transport layer information which is not always available or feasible in cloud deployments.

Each discussed system has some limitations in terms of excessive data overhead, in terms of measurement granularity, or in terms of lack of support for virtual switches. Their comparison to our work is summarized in Tbl. I.

III. Packet loss measurement in virtual switches

In this Section, we provide background knowledge related to OVS-DPDK and packet marking and coloring techniques that are later used for the design of our system.

A. OvS-DPDK

Open vSwitch (OvS) is the most popular virtual switch implementation used in production cloud environments. OvS is a multi-layer software switch with the ability to connect virtualized hosts between each other and to the physical network. With the standard version of the OvS, the kernel is the primary data path used for packet forwarding, usually referred to as the ”fast path”. Consequently, all packets are processed within the kernel. This inevitably contributes to performance limitations due to the interrupt-driven packet processing done by the kernel network stack leading to context switching between user space and kernel space. A user-space alternative of OvS, namely OvS-DPDK has been introduced in an attempt to address the limitations of the fully kernel-based implementation. OvS-DPDK exploits the DPDK technology to move the fast path to the user space. This led to a considerably improved throughput for packet forwarding, yet at the expense of increased resource utilization [12], [13]. Nevertheless, even under such architecture, any stateful per-packet-based processing techniques (e.g., per-packet counting) would be required to be implemented entirely within the fast path [14]. Thus, to maintain a high level of performance the per-packet processing logic should be highly optimized in terms of memory access and required CPU cycles. These considerations will later influence the design of our system discussed in Sec. IV.

B. Alternate Marking Performance Measurement

Alternate Marking Performance Measurement (AM-PM) [15], [16] was introduced as a method for packet loss, delay, and jitter measurements on top of live traffic. Due to its simplicity and popularity, the approach has been implemented in commercial switches and is currently supported off-the-shelf by some switch vendors.

In AM-PM, groups of packets are alternately marked (colored) by changing particular values of some of the bits belonging to the packet headers. In its simplest form, a single bit is used to mark the packets. The value of the marking bit can assume a value of either 0 or 1. This bit is toggled and stays constant for a given number of packets B, namely the *batch size*. After a single batch of packets, the value of the bit is inverted for the same amount of packets. To detect the packet loss a monitor must perform two basic operations, i.e., i) counting the number of packets of each color, and ii) detecting the time at which packets transition from one color to another. The latter is usually referred to as *step detection*. Whenever a step is detected the system assumes that a full batch of packets has been received and can trigger the calculation of the actual packet loss of the previous batch. To do so, it is sufficient to check the value of the counter for the color of the previous batch and to compare it against the expected number of packets B. If exactly B packets have been observed, no packet loss has occurred. Otherwise, the packet loss will be measured as the difference between B and the value of the packet counter.

IV. System design and implementation

In this section, we discuss the architecture, the methodology, and the design choices behind the developed packet loss monitoring system.

A. System design

The aim of this work is to provide lightweight and fine-grained per-flow packet loss monitoring. The ultimate goal

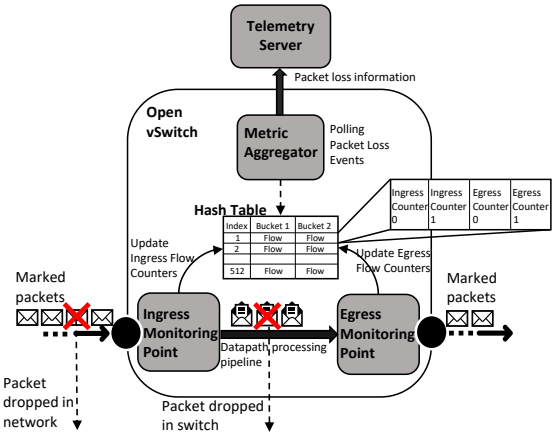


Fig. 1: Core components of our design

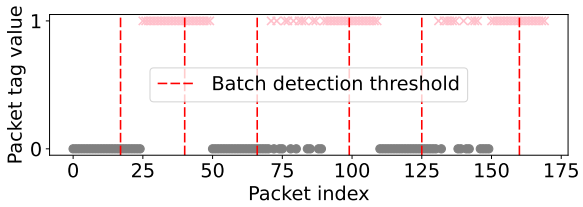


Fig. 2: Batch detection using a packet reordering-robust threshold

of our system is that of assisting the debugging of particular flows to understand exactly when packet loss occurs and to be able to correlate it later with monitoring signals coming from other sources. To achieve our goal we build our design on top of four key components. These components, alongside the overall architecture of our system, are illustrated in Fig. 1 and are as follows:

1) **AM-PM packet coloring** enables accurate and timely detection of packet loss
2) **Fast path-based packet counting** permits to reduce the resource overhead of the packet loss measurement algorithm
3) **Reordering-robust step detection algorithm** enables reliable results in case of severe packet reordering
4) **Data summarization within the fast path** permits us to reduce the data overhead between single switches and the telemetry server [10], [8]

In the following, we will provide an overview of how each of the components is integrated and the rationale behind the choices that have been made.

1) AM-PM packet coloring: We exploit AM-PM marking strategy as the basic strategy to detect packet loss. Notably, this choice leads to the least resource overhead as compared to INT-based solutions. We use the IP's DSCP field to carry this information across different virtual and physical switches. In our design, we assume that packets arrive already marked at the transient switches. We consider two alternatives for the actual location of the markers. In the first scenario, packets are marked at the hypervisor level which permits to exploit more resources at the expense of extending the coverage of the system beyond just the virtual network layer. On the other hand, we consider the possibility of packets being marked at the ingress port of the first virtual switch. In the latter scenario, the entirety of the tool remains constrained to the virtual network layer at the expense of added resource overhead.

2) Fast path-based packet counting: The most resource-demanding operation within the AM-PM system is packet counting. While it can be done efficiently on off-the-shelf silicon, implementing this mechanism in software poses considerable challenges due to resource constraints. We implemented the packet counting algorithm within the fast path of OvS by adding a new custom packet processing action, namely `pkt_loss_chk`. The action is executed whenever a new packet is received at the ingress or at the egress of the packet processing pipeline. Upon matching, the action will trigger a read of the packet color from the DSCP field and will subsequently increment the corresponding per-flow and per-color counter. We place two counters (namely, "0", and "1" counters) both at the ingress and at the egress of the OvS switch. This duplication of the counters permits us to measure both the packet loss that occurred between different network nodes, and the packet loss that occurred within the OvS internal pipeline. The counters are packed within a set of hash tables that enables the association of individual flows to the corresponding counters.

3) Packet reordering-robust step detection algorithm: The estimation of the packet loss is triggered by the transition between batches of opposite colors. To do so, in the most simplistic case, it is sufficient to detect the first packet that has a different color from the color of the currently measured batch. However, in realistic scenarios packet reordering may lead to some of the colors arriving interleaved as shown in Fig. 2. Under such a scenario, a simple step detection algorithm will lead to false results. For this reason, we implement step detection by looking at the percentage of the received packet of a given color in the last batch (dashed lines in Fig. 2). After a given "safe" percentage of packets of the *next batch* have been received we can assume that the step transition has already occurred. This effectively allows mitigation of the effect of packet reordering close to the color transition edge.

4) Data summarization within the fast path: The step detection algorithm effectively enables the detection of packet losses, as, once the step has been detected, the difference between the configured batch size and the counter value will give us the exact amount of lost packets within two packet batches. A separate process, namely *Metric Aggregator*, running outside of the fast path, is informed every single time a packet loss event is detected. The task of this component is to periodically poll the flow counters to check for a packet loss event and to transmit aggregate packet loss messages to be sent to an external telemetry server. Effectively, a report is only sent to the telemetry server if a packet loss has been

detected for a batch as opposed to continuous reporting in [8].

B. Implementation

We implemented the described key components of the system within the OvS-DPDK fast path with 1.5K lines of C code. We use the open-source version of OvS-DPDK 2.17.90 [17]. For the step detection algorithm, we use 30% as the threshold for triggering the detection of a step. The implementation of the per-flow counters uses 8-bit unsigned variables which leads to a maximum batch size of 255 packets. Furthermore, we enforce memory alignment of counters belonging to the same flow to maximize cache hit ratio during packet processing, and, thus minimize the impact of the `pkt_loss_chk` action on the overall performance of the virtual switch. We use a simple hash table of 512 indices with two entries for each index, for the storage and mapping of flow counters. The 32-bit flow hash value in the packet metadata is used as the hash table key for retrieving the index. The 16th most significant bit of the 32-bit flow hash value decides the location out of the two entries for the same index in case of a hash collision. Finally, we implement the metric aggregator as an OvS-DPDK off-path module and the telemetry server as a standalone module in a separate process.

V. Evaluation

Since the proposed approach only affects single virtual switches we focus our evaluation on the introduced performance overhead at a single switch. Additionally, we evaluate the overall correctness of the system by evaluating it in a small, yet realistic testbed. In particular, we measure the total throughput degradation and we verify whether the system behaves correctly under specific real-life conditions, e.g., variable degree of packet drop, and congestion.

A. Testbed Setup

For performance evaluation we consider a testbed running a single OVS-DPDK switch, while for functional evaluation we consider a testbed with 4 switches connected in a bus topology as depicted in Fig. 3. This topology is analogous to an overlay-underlay network setup, where terminal OvS instances connect the guest virtual machines in different physical hosts in an overlay network, and the underlay network consists of multiple switches in a linear path between the 2 terminal OvS instances.

We use DPDK libraries version 21.11.1, and Pktgen-DPDK [18] to generate marker synthetic data flow. Experiments are performed on top of an Intel Xeon Gold 6161 CPU with 385GB of RAM. For the single-switch setup, the OVS instance is deployed directly on the host. For the multi-switch setup, each switch is deployed in a separate VM with 3GB RAM and 6 logical cores.

B. Functional Correctness

In this section, we evaluate the accuracy of packet loss measurements. For this purpose we developed a synthetic packet dropped as a separate model within the OvS switch itself. The model permits us to drop packets at both the ingress and egress measuring points, thus enabling the emulation of scenarios including link-level and OvS-internal packet losses respectively.

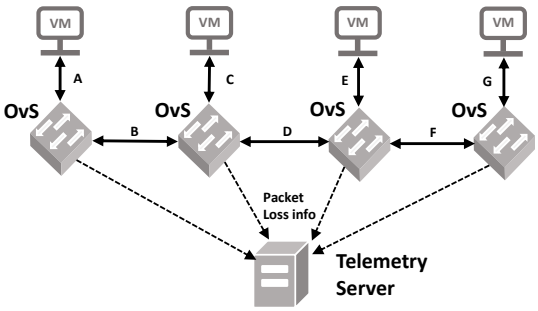
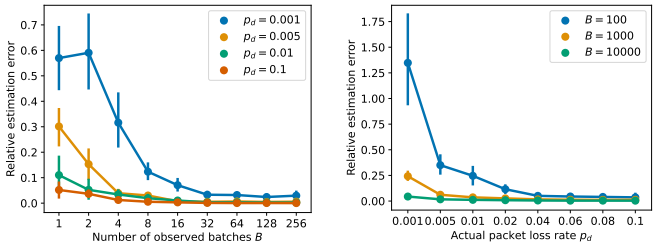


Fig. 3: Multi-switch evaluation setup



(a) Experiments with variable observed packets

(b) Experiments with variable batch sizes

Fig. 4: Relative estimation error vs B and observed packets

1) Accuracy: For evaluating the accuracy of the developed packet loss monitoring system, we employ a single-switch setup and instrument the packet dropper with variable packet drop probabilities p_d and we measure the estimation accuracy of our system for variable batch sizes. We instrument `pkt_loss_chk` to monitor just a single flow to avoid interference from other flows for the packet dropper.

Fig. 4a shows the relative estimation error, alongside with the 95% confidence interval, of the measured packet drop probability $\hat{p}_d$ in function of number of observed batches and for variable p_d. The lower the relative error, the higher the accuracy of the system. Overall it is observed that the accuracy of measurement increases with the number of observed batches and greater batch size, with the number of observed batches larger than 25 not bringing considerable improvement in terms of performance. On the other hand, Fig. 4b depicts relative estimation error for variable batch sizes. As expected, it can be observed that accuracy improves with greater batch size and for higher packet loss ratios.

2) Congestion detection: To understand the performance of our system in a more complex scenario we emulate network congestion using the multi-switch setup. We consider a hypothetical scenario in which the terminal OvS instances from Fig. 3 are connected to virtual hosts in an overlay network and the 2 OvS instances in between are considered as part of the underlying physical network. Packet loss monitoring

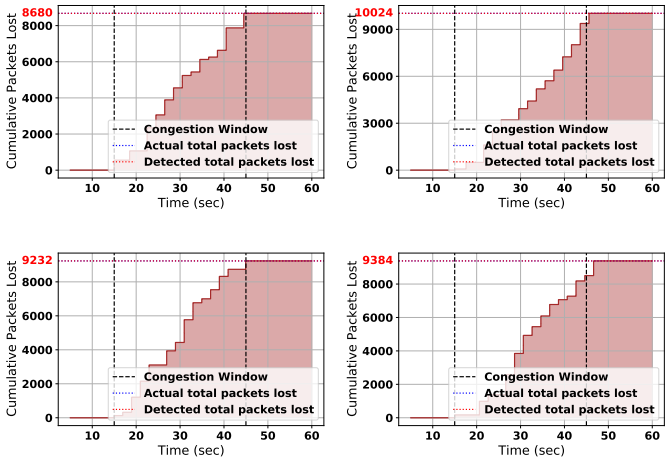


Fig. 5: Detected packet loss over time reported at the telemetry server for the congestion scenario (red and blue lines overlap).

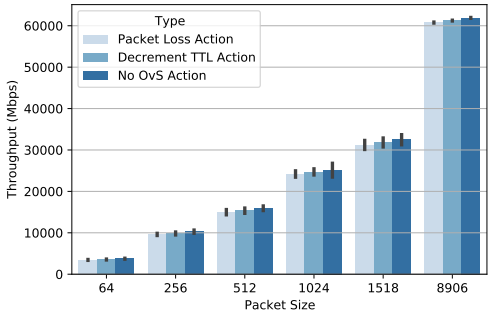


Fig. 6: Absolute throughput degradation for performance evaluation at maximum sending rate on OvS-DPDK for variable packet size and for different packet actions

is enabled only on the terminal OvS instances. The objective is to evaluate if the system is able to detect packet loss for flow A to G during congestion. Background traffic at a high rate (8 Mpps) is sent on the path $C \rightarrow D \rightarrow E$, while probe packets are sent from on the path $A \rightarrow B \rightarrow D \rightarrow F \rightarrow G$, thus simulating a transient congestion scenario in the underlay network which causes packet drops on link D. We run the network in the absence of background traffic for 15 seconds, after which the background traffic is turned on for 30 seconds and then turned off again. It is expected that packet loss will only be reported during the 30 seconds of congestion. We then compare the reported packet loss at every time with the ground truth packet loss obtained directly from the synthetic packet dropper.

Fig. 5 shows 4 different runs of the experiment. It is observed that packet loss in the network only occurs during the congestion time (timestamp seconds: 15-45). This is correctly detected by the packet loss monitoring system at the destination switch and coincides with this time period. Outside this time period, there were no packet losses observed for the monitored flow. This also concurs with no detection reported by the packet loss monitoring system.

C. Performance evaluation

The system has also been evaluated in terms of performance to establish that the designed concept is feasible to be used and that the performance variations observed are relatively tolerable in comparison to the clean version of OvS-DPDK and its predefined OvS actions. For evaluating performance, the throughput (in terms of bytes per second) of the system has been measured when monitoring multiple flows. All performance tests have been performed with a single-switch setup and a batch size of 100.

We use the *range* functionality of the *Pktgen-DPDK* to generate 10,000 active flows and monitor only a subset of them. The hash table size for storing the flow counters is defined to hold a maximum of 1024 elements, and out of 10,000 active flows, thus 1024 flows are monitored on a first come first serve basis. We consider a baseline the OvS `no_action` action, i.e., the pass-through configuration. The relative performance to the baseline throughput is calculated for both cases where the flow rule is configured with `pkt_loss_chk` action and then separately with the predefined `dec_ttl` OvS action. These two cases are compared relative to the baseline. Both actions are compared to the baseline values also to prove there is inherent degradation in switching throughput when an action is configured for a flow in OvS-DPDK in comparison to when no actions are configured. Note that for all cases, the `output` action is enabled to output the packet to a certain port.

Fig. 6 depicts the absolute average throughput in terms of Mbps alongside 95% confidence intervals for variable packet size and for different OvS actions enabled. It can be seen that the throughput grows considerably as the packet size increases, which is in line with previous findings [19]. At the same time, it can be observed that both the `dec_ttl` and `pkt_loss_chk` actions introduce some performance penalty when enabled. The `dec_ttl` action observes a maximum degradation in throughput performance of 5% with a packet size of 64 bytes, and minimum degradation in throughput performance of 1.1% with 8906-byte packet size, against baseline values. When enabling the `pkt_loss_chk` action we observed an additional maximum degradation of 3% with 64-byte packets and a minimum of 0.8% with 8906-byte packets with respect to the `dec_ttl` scenario. Additionally, we observed that the value of the batch size does not affect the forwarding performance of the switch and just affects the accuracy of the packet loss estimation.

Following this evaluation, we assessed the root cause of the performance degradation by analyzing the resource utilization of the machine running our modified version of OvS. Tbl. II depicts the effects of memory cache misses on the L1 and LLC caches and the measured normalized throughput. All values are normalized with respect to the `no_action` scenario.

A considerable increase in the rate of LLC-load misses and LLC-store misses can be seen when hash table entries are

TABLE II: Cache and throughput statistics for our experiments

Hash table size	Normalized L1-dcache miss rate	Normalized LLC-load miss rate	Normalized LLC-store miss rate	Normalized throughput
4	1.00	1.00	1.08	0.99
1024	1.01	1.00	1.12	0.97
2048	1.01	1.01	1.25	0.96
8192	1.02	1.07	2.49	0.87
16384	1.04	1.10	4.22	0.82
32768	1.10	1.14	8.06	0.76
65536	1.18	1.20	16.55	0.69

increased from 2048 to 8192. This also is in direct correlation with the significant drop in throughput. As the hash table size is increased even further, both LLC metrics show a directly correlated increase. A similar trend is also observed for the rate of L1-dcache misses with an increase in hash table size. However, we believe the LLC misses to be the main culprit for the performance degradation, as these are directly translated to memory accesses in the RAM.

VI. Discussion and future work

Our evaluation shows that the proposed system can measure packet loss with high accuracy and low overhead. Nevertheless, we acknowledge that there are certain limitations and further possible improvements to be done.

When more than 1024 flows are monitored simultaneously, a degradation of the forwarding throughout can be observed which was found to be related to the increased cache miss rate in our evaluation. A possible future work would target the use of more memory-efficient data structures, such as sketches [20], to maximize the cache hit ratio at the expense of some accuracy. Alternatively, some off-path monitoring techniques, such as the one proposed in [21] could be explored.

Additionally, an issue that has not been addressed in this work is active flow selection. Automatically identifying which flows to monitor and how to trigger the analysis requires further investigation. Nevertheless, we noticed that the proposed system is already able to address some of the critical scenarios that can manifest themselves in production environments. Indeed, when combined with other measurement metrics such as L7 metrics, by instrumenting our system with the monitoring of specific suspect flows, we able to pinpoint the exact time and location within the network of excessive packet loss.

VII. Conclusion

In this paper, we propose a low data overhead and fine-grained packet loss monitoring system for virtual switches. To achieve this goal, we exploit the traditional AM-PM scheme by carefully integrating it within the OvS-DPDK data path. Our solution has been successfully implemented and evaluated in an emulated environment with multiple instances of virtual switches. The evaluation showed that our design is able to achieve 95% accuracy for packet loss estimation while incurring a performance degradation for the switches' throughput in the range of 2-3%.

Following the evaluation results, we are confident about the possibility of applying our design in production cloud environments after introducing further optimizations. Indeed, we believe the developed system can help both cloud service providers to quickly detect and measure packet losses with a further understanding of the location and root cause of the fault.

References

[1] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred, "Taking the blame game out of data centers operations with netpoirot," in *ACM SIGCOMM*, 2016.

[2] P. Notaro, Q. Yu, S. Haeri, J. Cardoso, and M. Gerndt, "An optical transceiver reliability study based on sfp monitoring and os-level metric data," in *IEEE CCGrid*, 2023.

[3] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, and Y. Liu, "Rapid detection and localization of gray failures in data centers via in-band network telemetry," in *IEEE NOMS*, 2020.

[4] P. Loreti, A. Mayer, P. Lungaroni, S. Salsano, R. Gandhi, and C. Filsfils, "Implementation of accurate per-flow packet loss monitoring in segment routing over ipv6 networks," in *IEEE HPSR*, 2020.

[5] X. Zhang, Y. Wang, J. Zhang, L. Wang, and Y. Zhao, "Ringlm: A link-level packet loss monitoring solution for software-defined networks," *IEEE JSAC*, 2019.

[6] S. Lee, K. Levanti, and H. S. Kim, "Network monitoring: Present and future," *Computer Networks*, 2014.

[7] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *NSDI*, 2015.

[8] C. Fang, H. Liu, M. Miao, J. Ye, L. Wang, W. Zhang, D. Kang, B. Lyu, S. Zhu, P. Cheng *et al.*, "Towards automatic root cause diagnosis of persistent packet loss in cloud overlay network," *IEEE/ACM TON*, 2022.

[9] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "Segment routing over ipv6 (srv6) network programming," *Internet Requests for Comments*, 2021.

[10] L. Tan, W. Su, W. Zhang, H. Shi, J. Miao, and P. Manzanares-Lopez, "A packet loss monitoring system for in-band network telemetry: Detection, localization, diagnosis and recovery," *IEEE TNSM*, 2021.

[11] H. Wu, Y. Liu, S. Ni, G. Cheng, and X. Hu, "Lossdetection: Real-time packet loss monitoring system for sampled traffic data," *IEEE TNSM*, 2022.

[12] Q.-H. Nguyen and Y. Kim, "Performance of ovs-dpdk in container networks," in *IEEE ICTC*, 2020.

[13] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vswitch dataplane ten years later," in *SIGCOMM*, 2021.

[14] P. Chaignon, K. Lazri, J. François, T. Delmas, and O. Festor, "Oko: Extending open vswitch with stateful filters," in *SOSR*, 2018.

[15] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. Chen, and G. Mirsky, "Am-pm: Efficient network telemetry using alternate marking," *IEEE Network*, 2019.

[16] G. Fioccola, A. Capello, M. Cociglio, L. Castaldelli, M. Chen, L. Zheng, G. Mirsky, and T. Mizrahi, "Alternate-marking method for passive and hybrid performance monitoring," *Internet Requests for Comments*, 2018.

[17] "Open vswitch - source." [Online]. Available: https://github.com/openvswitch/ovs

[18] "Pktgen-dpdk packet generator." [Online]. Available: https://pktgen-dpdk.readthedocs.io/en/latest/

[19] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and latency of virtual switching with open vswitch: A quantitative analysis," *Journal of Network and Systems Management*, 2018.

[20] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *ICALP*. Springer, 2002.

[21] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "Flowatcher-dpdk: Lightweight line-rate flow-level monitoring in software," *IEEE TNSM*, 2019.