Real-Time Flow Statistics Collection using RDMA and P4 Programmable Data Planes

Elie Kfoury*, Ali Mazloum*, Jose Gomez◇, Ali AlSabeh†, Jorge Crichigno*
*College of Engineering and Computing, University of South Carolina (USC)
◇Katz School of Business, Fort Lewis College (FLC)
†College of Sciences and Engineering, University of South Carolina Aiken (USCA)
Emails: {ekfoury, amazloum}@email.sc.edu, jagomez@fortlewis.edu,
ali.alsabeh@usca.edu, jcrichigno@cec.sc.edu

***Abstract*—Measuring network traffic in real time is essential for applications such as traffic profiling, anomaly detection, resource allocation, and network performance improvement. As network speeds and traffic volumes increase, traditional solutions (e.g., NetFlow, sFlow, Zeek) face challenges in processing and summarizing traffic efficiently, often leading to incomplete measurements. This paper introduces a system that summarizes network traffic and provides per-flow measurements in real time by leveraging P4 Programmable Data Planes (PDPs). The system computes per-flow traffic statistics directly in the data plane at line rate. The statistics are then transmitted to a server using the low-latency, high-throughput RDMA over Converged Ethernet (RoCEv2) protocol. On the server, worker threads process the received reports and update a global data structure that maintain the flows. The system was implemented and tested using an Intel Tofino-based PDP and an RDMA-capable SmartNIC (NVIDIA BlueField-2). Experiments on real packet traces show that the system is capable of analyzing traffic at scale without compromising the accuracy of the measurements, outperforming traditional Network Security Monitors (NSMs).**

***Index Terms*—P4, RDMA, SmartNIC, flow-based measurements, flow collectors.**

I. Introduction

Data traffic rates have increased dramatically [1], creating significant challenges for operators in monitoring networks and extracting actionable information. This information is essential for a variety of applications, including building traffic profiles, detecting anomalies, optimizing resource allocation, and improving network performance. Traditional monitoring solutions, such as NetFlow, sFlow, and Zeek [2], have scalability limitations [3]. They are not capable of capturing all flows or packets, especially when the network is busy. Moreover, because they rely on sampling packets or aggregating statistics into reports, they do not allow the implementation of custom per-packet processing algorithms. Having such algorithms would significantly enhance the visibility into the network.

Recently, Programmable Data Planes (PDPs) [4] have emerged as a promising technology that enables operators to implement custom packet processing logic capable of operating at Terabit-per-second (Tbps) speeds. PDPs are programmed by P4 [5], a domain-specific language for networking. P4 allows developers to define the packet processing logic that is then compiled and deployed on programmable Application-specific Integrated Circuit (ASICs). Several solutions have been implemented in PDPs to provide deeper insights into network traffic, addressing the limitations of traditional monitoring approaches. These solutions can be classified into two categories [4]: telemetry-based and flow-based approaches. Telemetry-based solutions query the internal state of the switch's data plane, offering fine-grained measurements of various metrics, including queue occupancy, link utilization, and queuing latency. Flow-based solutions track and aggregate flow statistics, allowing operators to monitor per-flow metrics at scale. Both categories often overlook the challenges associated with transmitting and processing the collected measurements. These measurements are typically sent to a server, referred to as the collector, which is equipped with a general-purpose CPU. When the number of reports is large, the collector struggles to keep up, leading to packets being dropped [6].

Considering the data processing limitations of traditional monitoring solutions, as well as performance bottlenecks in general-purpose CPUs, this paper presents a system that leverages PDPs to track flow statistics at scale and in real-time. The statistics are then efficiently sent to a collector using Remote Direct Memory Access (RDMA) [7]. The system is designed to offload the computations of per-flow metrics to the PDP switch. Such computations are not feasible to implement on general-purpose CPUs. By using RDMA, the system ensures fast, low-latency transmission of flow reports from the PDP to the collector, where these reports are processed in real time. The collector continuously updates data structures with flow metrics, allowing applications to query the collected flow information. The source code of the system is publicly available [8] to facilitate further research and development.

The paper's contributions are summarized as follows:

1) Implementing a system that tracks per-flow metrics in real-time and in a scalable manner using PDPs. The system generates RDMA packets with metrics reports and sends them to a remote collector for processing.
2) Testing the system with real traffic traces from a backbone network, demonstrating the correctness of the measurements and the scalability of the overall system.

This work was supported by the NSF under awards 2346726, 2417823, and 2403360.

3) Comparing the performance of the system against Zeek, a widely used open source NSM.

The rest of the paper is organized as follows. Section II provides a background on various technologies used in the system and reviews the related work. Section III describes the proposed system. Section IV describes the implementation details and evaluates the system. Section V discusses future work and concludes the paper.

II. Background and Related Work

This section provides a brief background on PDPs and RDMA, and then describes the related work.

A. Programmable Data Planes

Programmable Data Planes (PDPs) enable developers to customize the packet processing pipeline [4]. PDPs are programmed using the P4 language [5]. They include a programmable parser, a match-action pipeline, and a deparser. The programmable parser allows defining and parsing standard or custom headers. The match-action pipeline then performs operations on packet headers and any intermediate data generated during processing. The deparser reassembles and serializes the headers, preparing the packet for transmission. PDPs also offer high-precision timing and stateful memory elements, including registers, counters, and meters, which can be accessed at line rate. These capabilities have been used for applications such as enhancing QoS for media traffic [9], classifying malware in real time [10], improving routing [11] and TCP performance [12], and others.

B. Remote Direct Memory Access (RDMA)

Remote Direct Memory Access (RDMA) is a technology that enables direct memory access between systems, allowing one system to read or write to the memory of another without the intervention of the operating system or the CPU. RDMA achieves very low latency and high throughput by bypassing the traditional OS networking stack, reducing the overhead associated with data copying. With RDMA, the Network Interface Card (NIC) manage memory transfers.

RDMA over Converged Ethernet (RoCE) is a protocol that enables RDMA packet transfers over Ethernet networks. RoCEv2 is an extension to RoCE that enables packets to be routed. RoCEv2 uses headers from the InfiniBand architecture [7] in its packet. Among these headers, the Base Transport Header (BTH) provides details about how the NIC should handle the packet's payload, and the RDMA Extended Transport Header (RETH) specifies the exact memory location where the payload should be written, along with other metadata.

To perform RDMA operations, a memory region must be registered. This registration is tied to a Queue Pair (QP), which has a Queue Pair Number (QPN). When setting up the QP, a type must be specified. Reliable Connected (RC) is a QP type similar to TCP, ensuring that messages are delivered once, in order, and without corruption. Unreliable Connected (UC) is analogous to UDP, offering no guarantee of message delivery.

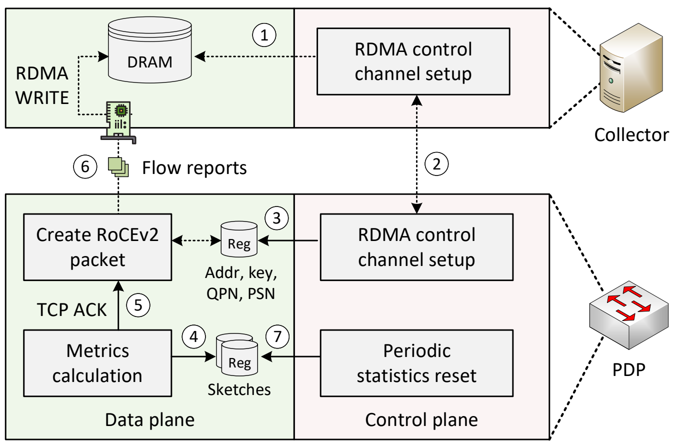

Fig. 1. Proposed system architecture. A control channel is established between the collector and the PDP. The PDP calculates per-flow metrics in the data plane and sends reports to the collector using RoCEv2.

When sending RDMA packets, the sender specifies the location in the remote memory where the data will be stored, the QPN, a Packet Sequence Number (PSN), the key (used to authenticate requests against the buffer), and the length of data to be written. The target NIC then processes the packet, performs boundary checks, and places the data in memory.

C. Related Work

PDPs have been used to collect telemetry data, such as queue occupancy, processing latency, and link utilization, which are sent to a collector for performance troubleshooting [4]. In-band Network Telemetry (INT) allows real-time collection of telemetry directly within data packets [6]. Some approaches combine RDMA with PDPs to transfer INT-based telemetry data to a collector more efficiently. Beltman et al. [13] developed a system that generates RDMA WRITE packets in P4 to store telemetry data on a server. Langlet et al. [14] introduced Direct Telemetry Access (DTA), which also uses RDMA to store telemetry reports from P4 switches in collector's memory. The proposed system differs by generating flow-based reports, tracking metrics like packet loss, RTT, bytes sent/acknowledged, starting time, packet counts, instead of just telemetry data. Other PDP-based approaches [15]–[17] collect flow-based information. However, these solutions either overlook the challenges associated with transmitting and processing the collected reports, or end up sending reports at lower rates (e.g., [17] sends 100s of reports per second). In contrast, the proposed system effectively process reports at high data rates (20 million reports per second).

III. Proposed System

Consider Fig. 1 which shows a high-level overview of the proposed system. The system leverages a PDP device to compute flow-level metrics at line rate. The PDP then sends reports using RoCEv2 to a collector. The steps are as follows:

1) On the collector, a control module allocates a fixed-size buffer in the Dynamic Random Access Memory (DRAM). It then registers UC QPs.

TABLE I
RAW METRICS CALCULATED/EXTRACTED BY THE DATA PLANE.

Metric	Description	Source
Timestamp	Time when report is generated	PDP's timer
Packet type	SYN, ACK, FIN, RST	TCP flags
ACK number	TCP ACK number	TCP header
Bytes count	Bytes sent before sketch reset	See §III-A1.
Packets count	Packets sent before sketch reset	See §III-A1.
Round-trip time	RTT using TCP handshake	See §III-A2.
Inter-arrival time	Time between last two packets	See §III-A3

2) The same module then establishes a control channel with a corresponding module running on the general-purpose CPU of the PDP. During the channel establishment, the collector provides the base memory address of the allocated buffer, key, QPN, and the initial PSN.
3) The control channel of the PDP writes these values to registers in the data plane.
4) The data plane continuously tracks flow-level metrics and updates its internal data structures.
5) When an TCP SYN, ACK, FIN, or RST packet is received, the data plane modifies the packet's headers and transforms it into a RoCEv2 packet. It then adds to it the flow report which contains the flow's measurements.
6) The data plane sends the packet to the collector. The RDMA-enabled NIC bypasses the collector's CPU and writes directly into the pre-allocated DRAM buffer.
7) A control plane module on the PDP performs periodic resets to the data structures to clear the statistics over time.

A. Metrics Calculation

The system calculates per-flow metrics, with some metrics computed exclusively by the data plane, and others derived by the collector using the metrics produced by the data plane. Table I lists the raw metrics computed by the data plane. The PDP has a high-resolution timer capable of providing nanosecond timestamps. These timestamps are used to identify the starting time of a flow (i.e., the time when the flow's SYN packet is received) and its end time (i.e., the time when the flow's RST/FIN packet is received). The PDP also extracts relevant information from the packet headers, including the TCP flags and the ACK number (if the packet is an ACK).

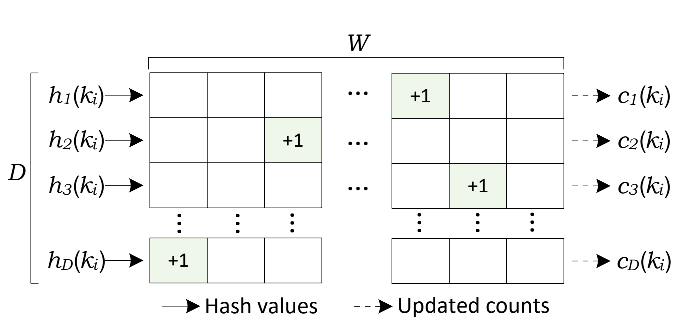

Fig. 2. Count-min Sketch (CMS) for estimating the per-flow packets count (also applicable for bytes count). Different hash functions compute the indices to be used in the register arrays where the packet counters are incremented. The minimum of these counters is an estimation of the flow's packet count.

TABLE II
DERIVED METRICS COMPUTED ON THE COLLECTOR.

Metric	Description	Source
Start time	The flow initiation time (first SYN packet)	Timestamp
End time	The flow termination time (FIN or RST packet)	Timestamp
Flow size	Total bytes sent by the flow (accounting for sketch resets)	Bytes count
Total packets	Total packets sent by the flow (accounting for sketch resets)	Packets count
Avg packet size (list)	The average packet size for the last acknowledged packets	Bytes count, packet count
Bytes in flight (list)	Data sent but not yet acknowledged	SEQ number, ACK number
Acknowledged bytes (list)	Total bytes acknowledged by the receiver	ACK number
Packet loss rate (list)	The flow's packet loss rate since the last ACK	Acknowledged bytes, flow size, SEQ number
Throughput (list)	The flow's throughput in bits per second (bps)	Bytes count, timestamp
Duration	The flow's active duration in seconds	Start time, end time

1) Packet and byte counts: The system tracks the per-flow packet and byte counts using two Count-min Sketches (CMSs). A CMS is a probabilistic data structure that reduces memory requirements, see Fig. 2. The input to the system consists of a packet stream $S = \langle k_i \mid i \in [1, N] \rangle$, where N indicates the total number of packets in the stream, and k_i serves as a key identifying the i-th packet's flow based on its 4-tuple: source IP, destination IP, source port, and destination port.

Let D represent the depth (i.e., the number of rows) and W represent the width (i.e., the number of columns) in the two-dimensional arrays used for the CMS. The counter at row d and column w after processing the i-th packet is denoted as $C_i(d, w)$. As depicted in Fig. 2, for each packet in S, the CMS for packet counts is updated as follows:

$$C_i(d, h_d(k_i)) = C_{i-1}(d, h_d(k_i)) + 1 \quad \text{for all } d \in [1, D].$$

The estimate of the total number of packets corresponding to the flow identified by k_i after the i-th packet is processed, denoted $c_i(k_i)$, is computed as:

$$c_i(k_i) = \min\{C_i(d, h_d(k_i)) | d \in [1, D]\}.$$

Similarly, the system maintains a separate CMS for the byte counts, which is calculated by accumulating the length of each packets associated with each flow.

2) Round-trip time: The system estimates the round-trip time (RTT) for a TCP flow by measuring the time difference between the SYN and SYN-ACK packets, using the PDP's precise timestamps. Upon receiving a SYN packet, the data plane records the current timestamp, T_{SYN}, and stores it in a register array indexed by the hash of the flow identifier $h_1(k_i)$. When the SYN-ACK packet is received, the system retrieves T_{SYN} from the register array using the hash of the reverse flow identifier $h_1(k_i^{\text{rev}})$. The RTT is then calculated as:

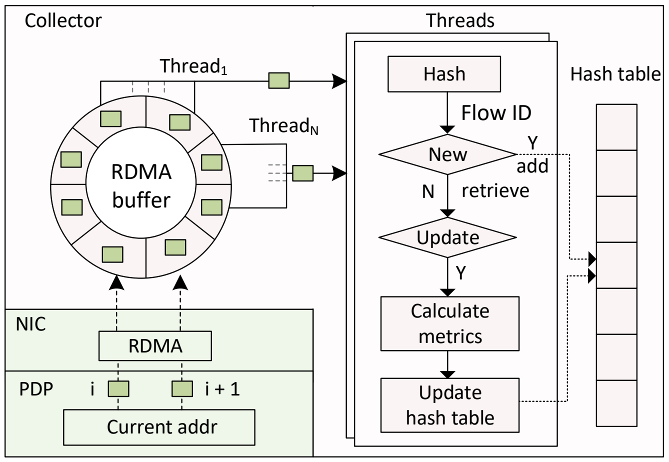

Fig. 3. Reports processing. The worker threads continuously iterate over their corresponding memory segments to retrieve updates. The threads then recompute aggregate metrics and update the flows' hash table.

$$RTT(h_1(k_i^{\text{rev}})) = T_{\text{now()}} - T_{\text{SYN}}(h_1(k_i^{\text{rev}})).$$

3) Inter-arrival time (IAT): The IAT of packets is a crucial metric for assessing the burstiness of a flow. It is calculated by measuring the time difference between two successive data packets belonging to that flow. The main challenge with IAT is that it is inherently a per-packet metric. Given the limited memory footprint of the PDP, the system computes a single sample of the IAT based on the last two packets received before an acknowledgment (ACK) arrives:

$$IAT(h_1(k_i)) = T_{\text{now()}} - T_{\text{IAT}}(h_1(k_i)),$$

where $T_{\text{IAT}}(h_1(k_i)$ is the time of the previous packet.

4) Derived metrics: The collector computes additional metrics, referred to as derived metrics, using the raw metrics computed/extracted by the data plane. Table II lists the derived metrics, their descriptions, and their sources. Note that this list is not exhaustive; other metrics could be derived. Some of the derived metrics will hold multiple values over time[1]. This is useful to track values over time (e.g., throughput, packet loss).

B. Report Preparation

The data plane maintains registers to store the RDMA write address, key, QPN, and PSN. The RDMA control channel setup module initializes these values. When an ACK is received, the data plane performs the following actions:

1) Retrieve the current memory address, key, QPN, and PSN from the register, then increment the address by the report size and the PSN by one.
2) Replace the packet's destination IP with the collector's IP.
3) Invalidate the TCP header, set the UDP header as valid, update its length, and update the destination port to RoCEv2's port number assigned by IANA (4791).
4) Add the BTH header, and assign the PSN and QPN.

[1]Metrics marked with (list) in Table II indicate they contain multiple values.

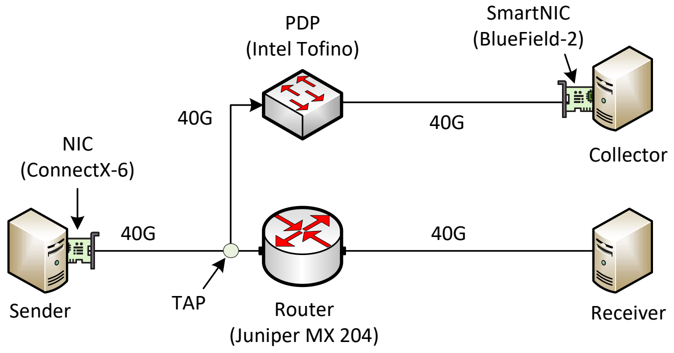

Fig. 4. Topology used to implement and test the system.

5) Add the RETH header and assign the memory address, key, and length.
6) Insert the payload with metrics of Table I.
7) Calculate the Invariant CRC (iCRC) checksum.
8) Send the packet to the collector.

C. Reports Processing

The collector initializes threads that continuously iterate over fixed-size segments in the allocated buffer, processing a subset of the reports pushed by the data plane. Each thread parses its reports and updates a global hash table that tracks the metrics for all flows. As shown in Fig. 3, each thread computes a flow ID by hashing the 4-tuple of each report. If the flow ID is new, it inserts the entry into the hash table. Otherwise, the thread compares the report's timestamp to the last recorded timestamp for the flow. If the report's timestamp is more recent, it recomputes the flow's derived metrics, and updates the hash table entry.

IV. Implementation and Evaluation

Fig. 4 shows the network topology. The sender is connected to a legacy router (Juniper MX 204) using an NVIDIA ConnectX-6 NIC. An optical tap is placed on the link between the sender and the router, which produces a copy of the traffic and forwards it to the PDP. The PDP is the Intel Tofino [18] on an Edgecore Wedge100BF-32X switch. The collector is connected to the PDP through an RDMA-capable SmartNIC (NVIDIA BlueField-2 [19]). All links operate at 40Gbps.

A. Measurements Accuracy with Real Traces

This experiment evaluates the accuracy of some of the metrics measured by the proposed system using two network traffic traces from the Measurements and Analyses on the WIDE Internet (MAWI) [20]. The MAWI traces are publicly available; they are anonymized and their payload has been truncated. The first trace, from October 25, 2010, represents lower traffic rates, with an average rate of 166.17Mbps and 27.65Mbps standard deviation. The second trace, from October 25, 2024, represents higher traffic rates, with an average rate of 2989.26Mbps and a standard deviation of 666.33Mbps. Fig. 5 shows the scatter plots for RTT, flow size, and packet counts. The left column shows the results for the lower-rate trace,

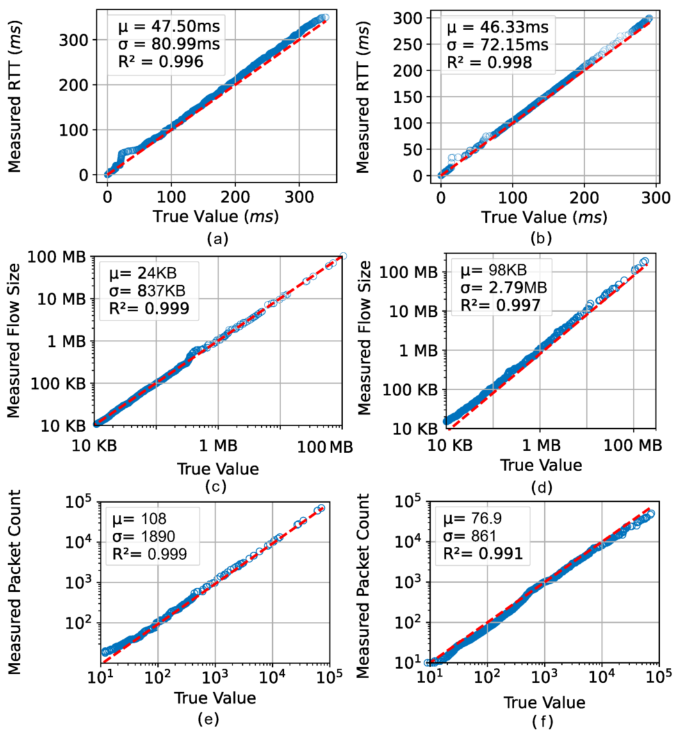

Fig. 5. Scatter plots for the RTT, flow size, and packet counts. The left and right columns show the results with the lower and higher rate traces, respectively. The red diagonal line represents a perfect match between the true and measured values.

while the right column shows results for the higher-rate trace. The x-axis represents the true values and the y-axis shows the measured values. Since both axes use similar scales, the red diagonal line represents a perfect match between the true and measured values. It can be seen in the figure that the measurements align closely with the true values. This is further quantified using the coefficient of determination, R^2, which measures how well the observed values approximate the true values. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where y_i represents the true values, $\hat{y}_i$ represents the measured values, $\bar{y}$ is the mean of the true values, n is the total number of observations. An R^2 value close to 1 indicates that the measured values closely match the true values, while a value closer to 0 indicates a weaker match. Note that all the experiments had an $R^2 > 0.99$. Due to space limitation, the results of only three metrics are reported. The other metrics, however, produced similar results.

B. System Responsiveness to Flow Dynamics

This experiment evaluates the system's responsiveness to changes in flow dynamics. The high-rate trace is replayed while simultaneously starting two iPerf3 flows. Fig. 6 presents the results: the top plot shows throughput, and the bottom plot shows the inter-arrival time (IAT). The gray striped bars indicate periods when the rates change, with specific rate values labeled above each bar. Rate limiting is controlled via the Linux TC queue disciplines (`tc qdisc`) only for the iPerf3 flows. The green-shaded area represents the throughput of the aggregate background traffic. It can be seen that the system promptly detects rate changes, accurately reflecting per-flow throughput and IAT. Note that as the flow rate decreases, IAT values increase, and vice-versa. Due to TCP burstiness, IAT values vary within the same rate.

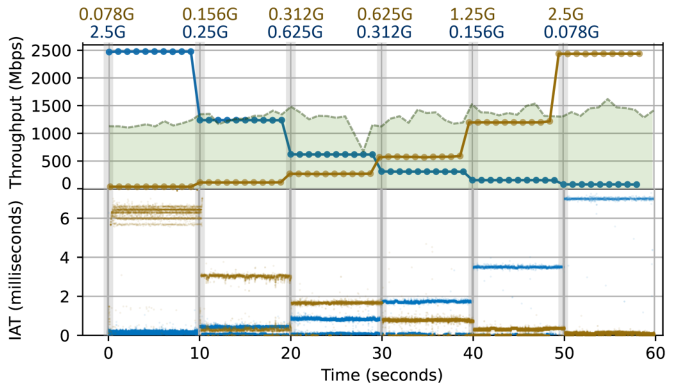

Fig. 6. System responsiveness to flow rate changes: Top is the throughput, bottom is the inter-arrival times (IAT). Gray striped bars indicate flow rate adjustments, and the green-shaded area represents throughput for background traffic. The blue and yellow lines/points represent the throughput/IAT values of two iPerf3 flows.

C. Reporting Rate Measurement

The proposed system generates a report (an RDMA packet) for each received TCP ACK, SYN, RST, and FIN packet. This experiment measures the expected number of reports per second from real traces. Fig. 7 (a) shows the Cumulative Distribution Function (CDF) from four traces from MAWI (2024). This report count varies based on the nature of the trace. For instance, a trace with many data transfers will have more ACKs than a trace with fewer data transfer flows, leading to more reports. The results show that the average number of reports range from 39K to 104K, depending on how busy the network is. Fig. 7 (b) shows the percentage of successful memory writes by RDMA as a function of the number of generated reports. For this experiment, the DPDK Pktgen tool

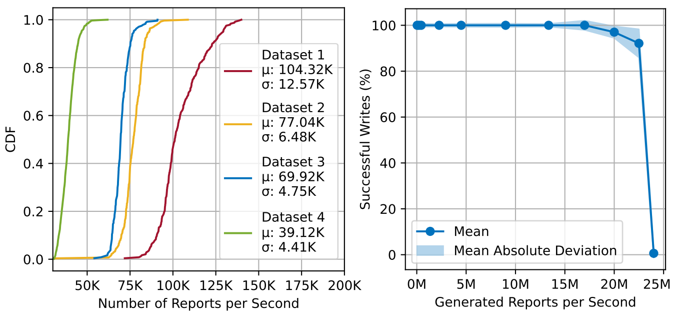

Fig. 7. Reporting rate performance. (a) the number of reports per second from real traces and (b) the percentage of successful memory writes as a function of the number of generated reports.

TABLE III
PACKET DROP RATES WITH REAL TRACES. ZEEK STARTS DROPPING PACKET AS THE RATE INCREASES.

Trace year	**Average rate (Mbps)**	**Replayed packets**	**Packet drop rate percent**	
			Zeek	**Proposed system**
2021	1,050	22,915,129	19.69%	0%
2022	1,200	24,606,314	26.71%	0%
2023	2,400	18,166,668	31.60%	0%
2024	2,800	25,495,454	37.93%	0%

[21] is used to send ACK packets at high rates. The system was able to process $\approx$ 22 million reports per second, with negligible packet losses. Beyond this number, the memory writes drop to $\approx 125,000$ report per second. Note that the system is able to process two orders of magnitude more reports than what is needed in traces from a backbone network.

D. Comparison with Zeek

Zeek Network Security Monitor (NSM) [2] is a widely used network monitoring system. This experiment evaluates the performance of Zeek by replaying four MAWI traces from 2021 to 2024, as shown in Table III. The more recent traces have increasingly higher packet rates. Zeek offers a broad range of features (e.g., regular expression matching, anomaly detection). For a fair comparison, all the default Zeek scripts are disabled. Instead, the metrics measured are implemented in a custom Zeek script. The same hardware and threading configuration is used for both Zeek and the proposed system. Zeek is started on the receiver device. As the packet rate increases, Zeek experiences growing packet losses due to the extensive computations it performs on raw packets. In contrast, the proposed system performs computations on the packets directly on the PDP device (i.e., hardware). The software on the collector is only used to process statistics. This approach handles high packet rates without any packet loss.

V. CONCLUSION AND FUTURE WORK

This paper presented a system that monitors network traffic in real-time using PDPs and sends reports to collectors using RDMA. The measured metrics are important for a variety of applications. The system was tested using an Intel Tofino-based PDP switch and an NVIDIA BlueField-2 SmartNIC. The experiments conducted with real traces demonstrated that the system measures metrics with high accuracy, effectively responds to flow dynamics, and is capable of processing 20M+ reports per second. The system provides substantial performance improvement over conventional solutions such as Zeek. For future work, the authors will: 1) expand the system to include additional protocols beyond TCP (e.g., UDP, ICMP); 2) implement the PDP logic directly on the SmartNIC/DPU [22], removing the need for an external PDP; and 3) develop a Digital Twin Network (DTN) [23] that leverages the collected measurements. The DTN will be deployed on the FABRIC testbed [24], providing researchers with a twin network environment for testing network research experiments.

ACKNOWLEDGMENT

The authors would like to acknowledge the National Science Foundation (NSF) for supporting this work, under awards 2346726, 2417823, and 2403360. The authors would also like to acknowledge the FABRIC [24] team for facilitating the platform used for some tests.

REFERENCES

[1] Industry Market Research Reports and Statistics, "Internet traffic volume." https://www.ibisworld.com/us/bed/internet-traffic-volume/88089/. Accessed: Oct. 30, 2024.
[2] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.
[3] R. Hofstede *et al.*, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.
[4] E. F. Kfoury *et al.*, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE access*, vol. 9, pp. 87094–87155, 2021.
[5] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
[6] L. Tan *et al.*, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.
[7] InfiniBand Trade Association, "Infiniband™ architecture specification volume 1 release 1.3," tech. rep., March 2015.
[8] E. Kfoury, "Source code of P4-RDMA on GitHub." [Online]. Available: https://github.com/ekfoury/P4-RDMA. Accessed: Nov. 1, 2024.
[9] E. Kfoury *et al.*, "Offloading media traffic to programmable data plane switches," in *IEEE International Conference on Communications (ICC)*, 2020.
[10] K. Friday *et al.*, "Inc: In-network classification of botnet propagation at line rate," in *27th European Symposium on Research in Computer Security*, 2022.
[11] A. Mazloum *et al.*, "A survey on rerouting techniques with P4 programmable data plane switches," *Computer Networks*, vol. 230, p. 109795, 2023.
[12] J. Gomez *et al.*, "A survey on TCP enhancements using P4-programmable devices," *Computer Networks*, vol. 212, p. 109030, 2022.
[13] R. Beltman *et al.*, "Using P4 and RDMA to collect telemetry data," in *2020 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, pp. 1–9, IEEE, 2020.
[14] J. Langlet *et al.*, "Direct telemetry access," in *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 832–849, 2023.
[15] Z. Liu *et al.*, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 101–114, 2016.
[16] A. Gupta *et al.*, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 357–371, 2018.
[17] A. Mazloum *et al.*, "Enhancing perfSONAR measurement capabilities using P4 programmable data planes," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 819–829, 2023.
[18] Intel, "Intel® Tofino™." [Online]. Available: https://tinyurl.com/3dzu54a2/. Accessed: Nov. 1, 2024.
[19] NVIDIA, "NVIDIA Mellanox BlueField-2 data processing unit (DPU)." [Online]. Available: https://tinyurl.com/yrky7ee5.
[20] MAWI, "Packet traces from WIDE backbone ." [Online]. Available: https://mawi.wide.ad.jp/mawi/. Accessed: Nov. 1, 2024.
[21] Intel Corporation, "Pktgen-DPDK." [Online]. Available: https://github.com/pktgen/Pktgen-DPDK. Accessed: Oct. 30, 2024.
[22] E. Kfoury *et al.*, "A comprehensive survey on SmartNICs: Architectures, development models, applications, and research directions," *IEEE Access*, vol. 12, pp. 107297–107336, 2024.
[23] P. Almasan *et al.*, "Network digital twin: Context, enabling technologies, and opportunities," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 22–27, 2022.
[24] I. Baldin *et al.*, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.