Scalable Heavy Hitter Detection: A DPDK-based Software Approach with P4 Integration

Samia Choueiri, Ali Mazloum, Elie F. Kfoury, Jorge Crichigno
University of South Carolina, USA
{choueiri, amazloum, ekfoury}@email.sc.edu, jcrichigno@cec.sc.edu

***Abstract*—Identifying heavy hitters is vital for applications like Denial of Service (DoS) detection and traffic engineering. Current solutions fall into hardware or software categories. Hardware solutions (e.g., P4 programmable data plane switches) offer high performance but require adding hardware, which may not be ideal for virtualized environments (e.g., cloud). Software solutions are cost-effective and flexible but suffer from performance issues due to the packet processing overhead in the Operating System (OS) kernel. This paper presents a scalable heavy hitter detection algorithm in the software, bypassing the kernel using the Data Plane Development Kit (DPDK). The Count-min Sketch (CMS) data structure is used to estimate the frequency of packets per flow. The system is implemented in P4 and deployed on the P4-DPDK target running on CPU cores. The experiments analyzed the impact of various parameters such as the packet size distribution, the number of CPU cores, and the number of hash functions, on the performance and the accuracy of the detection. The system's performance is further evaluated through comparison with another DPDK-based approach for heavy hitter detection. The results show accurate identification of heavy hitters and improved performance, even at a high traffic rate approaching 100Gbps.**

***Index Terms*—DPDK, Heavy Hitter, P4, SmartNIC.**

I. Introduction

Heavy hitters are flows that contribute a significant amount of traffic to a link. Their presence can have a drastic impact on the performance and resilience of the network. They can exhaust the network bandwidth, degrade the Quality of Service (QoS), or even disrupt the communication in the network. Detecting heavy hitters is crucial across various applications such as congestion control, intrusion detection and prevention, traffic rerouting, and network capacity planning. For instance, a large data transfer, which results in a heavy hitter flow, can be separated from latency-sensitive short flows by assigning it to a dedicated queue. This will reduce the Flow Completion Time (FCT) of short flows and will mitigate their burstiness, thus averting degradation of throughput of the long flow [1]. Another example is identifying heavy hitters for mitigating Denial of Service (DoS) attacks [2].

A simple approach to identify heavy hitters is to count the number of packets per flow. If the flow's packet count exceeds a predefined threshold, the flow is identified as a heavy hitter. However, networks today can have millions of active flows, especially since the majority of these flows are short-lived (e.g., web browsing traffic). Thus, a simple counter-based solution is not scalable as it will have a large memory footprint. An alternative is to use the Count-min Sketch (CMS), which is a probabilistic data structure, to estimate the number of packets per flow. Compared with the simple counter-based approach, the CMS uses a much smaller memory footprint (i.e., $O(\log N)$ for N active flows).

Existing solutions for detecting heavy hitters are either hardware or software-based. Hardware solutions, such as P4 programmable data plane switches, can promptly detect heavy hitters without impacting the network performance. However, they require adding and maintaining the hardware, which may not be possible in certain environments (e.g., cloud computing). Software solutions, on the other hand, are more flexible and easier to deploy. However, they often suffer from performance issues due to the packet processing overhead, especially with high traffic rates.

This paper implements a heavy hitter detection system that leverages the Data Plane Development Kit (DPDK). DPDK is a kernel-bypass acceleration technology [3] that avoids the packet processing overhead incurred within the OS kernel. The proposed system implements a Count-min Sketch (CMS) to approximate the per-flow packet count. It is written in P4 code and deployed on the P4-DPDK target. The system was tested and evaluated with synthetic and real traffic, considering various parameters such as the packet size distribution, the number of CPU cores on which the system is running, the number of hash functions used in the CMS, etc. The implementation is made publicly available [4].

The contributions of the paper are summarized as follows:

- Implementing a scalable heavy hitters detection solution using DPDK. The system uses CMS to reduce the memory footprint.
- Analyzing the impact of parameters such as the packet sizes, the CPU cores, the number of hashes, etc., on the performance of the detector.
- Testing the system with real dataset on FABRIC [5], an international testbed used for research and experimentation.
- Comparing the performance of the system against a popular open-source IDS/IPS that detects heavy hitters.

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

II. Related Work

This section reviews the existing heavy hitter detection solutions, categorized by those deployed on hardware (programmable data plane switches) and those implemented in software with DPDK.

A. Hardware Approaches (Programmable Data Planes)

Programmable data plane switches allow the developer to devise custom packet processing algorithms running at line rate. By incorporating algorithms such as Count-min Sketch or Space-saving, P4 switches can efficiently identify heavy hitters [6], [7] and other attacks [8]. With P4's flexibility, operators can adapt detection mechanisms to suit evolving network demands, ensuring accurate identification of heavy hitters while minimizing resource overhead [9]. However, despite the advantages, using P4 switches has certain drawbacks like the requirement for additional hardware. This can result in increased deployment costs and complexity, especially for organizations with existing infrastructure that may not readily support P4-enabled hardware. Furthermore, it is not possible to use these systems in cloud environments where resources are dynamically allocated.

B. Software-based Approaches (DPDK)

Software-based detectors are more flexible than the hardware ones because they can be deployed in almost all environments, without the need to add specialized hardware. While DPDK-based approaches still require DPDK-compatible NICs, most NICs today already support DPDK [10].

Heavy hitter detection using DPDK has been explored in [11]–[13]. However, these approaches have some limitations: They are implemented in C using DPDK, lacking the flexibility of P4 for integrating new functionalities, particularly as the program's complexity grows; they are integrated within other functionalities as part of Open vSwitch (OVS), which could entail unnecessary processing overhead not required in environments solely focused on heavy hitter detection; they lack validation on real testbeds with actual traffic.

III. Background

A. P4 Language and Architectures

The Protocol Independent Switch Architecture (PISA) is a packet processing model for networking [14]. PISA is programmed using the Programming Protocol-independent Packet Processor (P4) language [15]. With P4, the developer implements the code for a programmable parser, programmable match-action pipeline, and programmable deparser. The programmable parser enables the developer to define and parse custom headers. The programmable match-action pipeline executes the operations over the packet headers and intermediate results. The deparser assembles the packet headers back and serializes them for transmission.

Although P4 was initially intended to program the data plane of PISA-based switches, it has demonstrated its versatility to program data planes for other packet processing devices. For instance, the Portable NIC Architecture (PNA) [16] is a P4 architecture that defines the structure and common capabilities for SmartNICs.

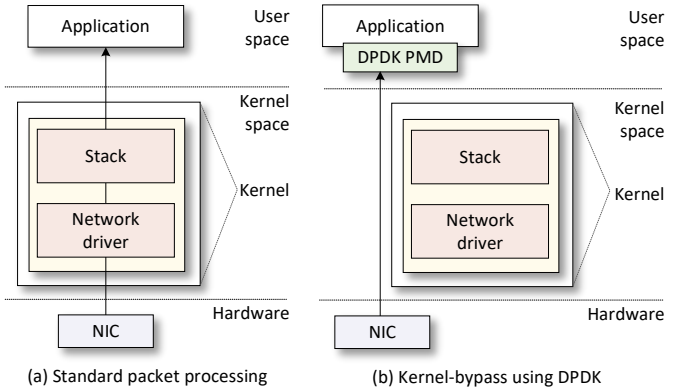

Fig. 1. Comparison between (a) standard packet processing and (b) software packet processing using DPDK (reproduced from [3]). DPDK bypasses the kernel, which avoids the packet processing overhead.

B. Data Plane Development Kit (DPDK)

Fig. 1 (a) shows the standard packet processing in the kernel. When a packet is received, the NIC triggers an interrupt that informs the OS about the packet's location in memory. The OS subsequently transfers the packet to the network stack which then initiates system calls from the OS kernel to deliver the packet to its intended user-level application. These steps induce overheads that dramatically degrade the bandwidth throughput. The Data Plane Development Kit (DPDK) comprises a collection of libraries and drivers designed to enhance packet processing efficiency by bypassing the kernel space and handling packets within user space (see Fig. 1 (b)). With DPDK, the ports of the NIC are disassociated from the kernel driver and associated with a DPDK-compatible driver. In contrast to the conventional method of packet processing within the kernel stack using interrupts, the DPDK driver operates as a Poll Mode Driver (PMD). It consistently polls for incoming packets. The utilization of a poll mode driver, combined with the kernel bypass, yields superior packet processing performance. DPDK's APIs can be used in C programs.

C. P4-DPDK

Writing P4 programs is generally considered more straightforward compared to writing DPDK. Consequently, there have been efforts to translate P4 codes into DPDK. The p4c compiler with the DPDK backend [17], known as p4c-dpdk, translates a P4 program into a DPDK API (.spec file) for configuring the DPDK software switch (SWX) pipeline [18]. The subsequent step involves the generation of a C code from the .spec file. This code includes C functions corresponding to each action and control block. A C compiler then generates a shared object (.so) from the C code.

IV. Proposed System

Consider Fig. 2 which shows a high-level overview of the proposed system. The system implements a heavy hitter detection pipeline running on multicore CPUs in the user

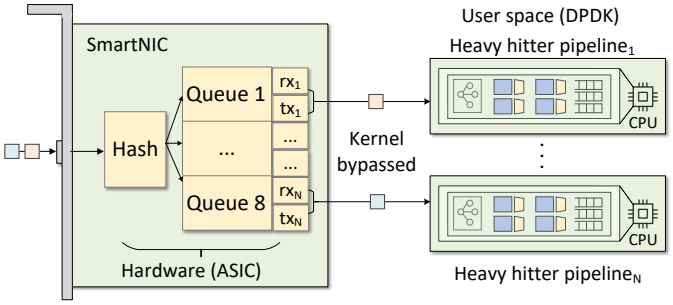

Fig. 2. Proposed system architecture. Heavy hitter detection pipelines are deployed on CPU cores. Packets are distributed to be processed by different CPU cores using RSS at the SmartNIC.

space. The system bypasses the kernel by leveraging DPDK. Upon receiving a packet, the SmartNIC forwards it to one of the pipelines running on a certain CPU core. The packet is then processed by the heavy hitter detection algorithm.

A. Distributing Packets to CPU Cores

When a NIC receives a packet, it typically goes through a single CPU core for processing. This can become a bottleneck, especially when the traffic rate is high, as the CPU core can become overwhelmed with processing incoming packets. To alleviate this bottleneck, the NIC can distribute the packets across multiple CPU cores using the Receive Side Scaling (RSS) technique. RSS uses a hashing algorithm to determine which receive queue should handle each incoming packet (see Fig. 2). The hashing algorithm uses the packet header information, such as the 5-tuple[1], to generate a hash value. This hash value is then used to determine the appropriate receive queue for the packet, and subsequently, the CPU core where it will be processed. Note that all the packets belonging to the same flow will be assigned to the same CPU core. The NIC then forwards the packet to the pipeline running on the CPU core.

B. Heavy Hitter Detection

Upon receiving the packet, the heavy hitter detection logic running in the pipeline executes. The algorithm estimates

[1]Source and destination IP addresses, source and destination port numbers, and protocol.

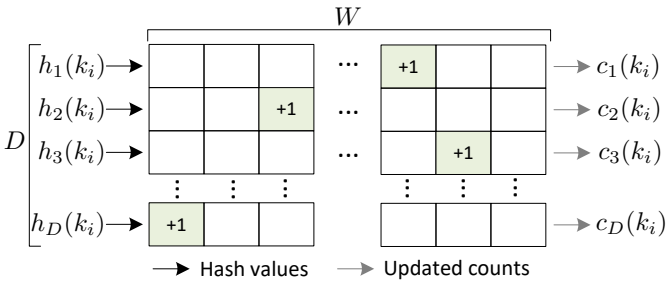

Fig. 3. Count-min Sketch (CMS) for estimating the per-flow packet counts. Different hash functions compute the indices to be used in the register arrays where the packet counters are incremented. The minimum of these counters is an estimation of the flow's packet count.

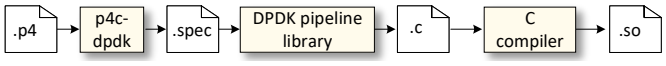

Fig. 4. The p4c-dpdk workflow.

the number of packets per flow and compares that number against a predefined threshold. The Count-min Sketch (CMS) data structure, which provides fast updates while significantly reducing memory requirements compared to exact counting methods, is used (see Fig. 3). The input to the system is a packet stream $S = \{(k_i)|i \in [1, N]\}$, where $N =$ is the total number of packets in the stream and k_i is a key to identify the ith input packet's flow based on the 5-tuple. Let ϕ be a predefined packet count threshold value. A heavy hitter is a flow identified by key K, whose estimated packet count $c_i(k_i)$ is at least ϕ.

Let D and W denote the number of rows (depth) and the number of columns (width) of the two-dimensional array representing the CMS. Let $C_i(d, w)$ represent the counter at row d and column w after processing the i-th packet. As shown in Fig. 3, for each packet in S, for all $d \in [1, D]$, $C_i(d, h_d(k_i)) = C_{i-1}(d, h_d(k_i)) + 1$. $c_i(k_i)$ is the estimation of the total number of packets by the flow with ID k_i after the i-th packet is processed. $c_i(k_i)$ is calculated as $\min\{C_i(d, h_d(k_i))|d \in [1, D]\}$. If $c_i(k_i) \geq \phi$, then the flow with a key k_i is reported as a heavy hitter.

C. DPDK Pipeline Generation

The system is implemented in P4 using the PNA architecture. The P4 code consists of all the packet processing functions, including the heavy hitter detection algorithm. It is translated into a DPDK pipeline that runs on the CPU core of the host bypassing the kernel. Consider Fig. 4. The p4c-dpdk compiler accepts the P4 code as input and generates a representation file (.spec file) that aligns with the DPDK SWX pipeline. Subsequently, using the DPDK codegen function, the C code is generated and then compiled. In the proposed system, the same pipeline generated from the P4 code is deployed on the CPU cores. Note that it is possible to use different programs and chain the pipelines.

V. Implementation and Evaluation

The system is implemented on FABRIC [5], an NSF-funded international testbed for large-scale research and experimentation. The FABRIC infrastructure consists of a distributed set

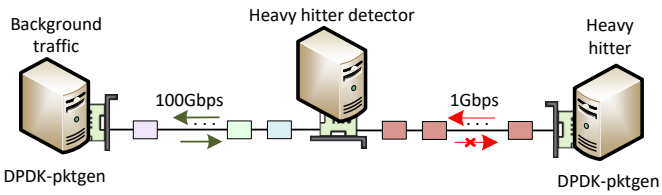

Fig. 5. Experiment topology. The background traffic is generated by randomizing the 5-tuple. The heavy hitter traffic is generated from a single 5-tuple.

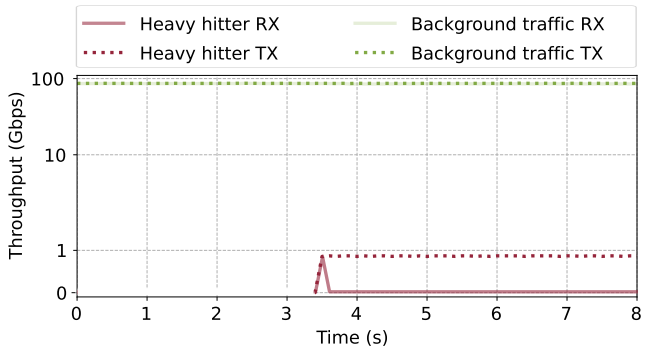

Fig. 6. Heavy hitter mitigation with 100Gbps background traffic. The proposed system promptly detected and blocked the heavy hitter.

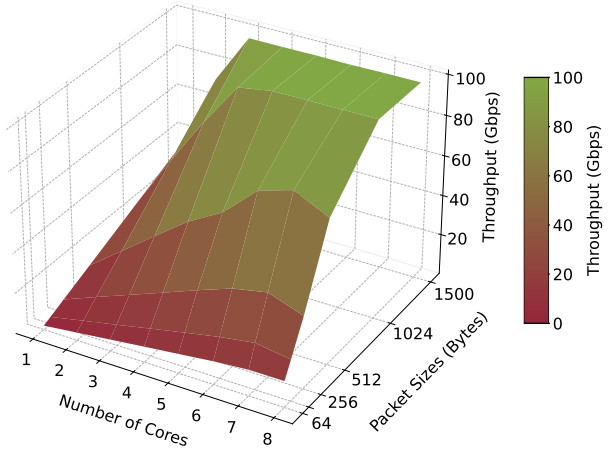

Fig. 7. Throughput as a function of number of cores and packet size. Increasing the number of cores and the packet sizes improves the throughput.

of equipment at commercial collocation spaces, national labs, research and education networks, Internet Service Providers, and campus networks. FABRIC provides resources such as servers, NICs, high-speed links, etc.

The topology, which is shown in Fig. 5, consists of three nodes, each equipped with an NVIDIA ConnectX-6 NIC (100Gbps). All the servers are on the same FABRIC site. DPDK-pktgen [19] is used to generate background and heavy hitter traffic. The heavy hitter traffic consists of packets all belonging to the same flow (i.e., same 5-tuple). The heavy hitter detector reflects all incoming packets. This allows measuring the performance of the system by comparing the number of sent packets with the number of received (reflected) packets. Unless otherwise specified, the experiments are conducted using four CPU cores and 1500 bytes packet sizes.

A. Detecting Heavy Hitter with High Traffic Rate

This experiment assesses whether the system can detect heavy hitters in the presence of high traffic rates. The background traffic generator is sending at a 100Gbps rate. The 5-tuples are randomized to ensure that the traffic does not belong to a single flow. Fig. 6 shows the results of this experiment. All the generated background traffic is reflected, which denotes that the pipeline was able to process all these packets. In the fourth second, a heavy hitter is generated. When its packet counter exceeds the predefined threshold value ϕ, a heavy hitter is identified and blocked, even while processing 100Gbps of background traffic.

B. Evaluating the Impact of the Number of Pipelines and Packet Size on the Throughput

This experiment studies the impact of the number of running DPDK pipelines (one pipeline per CPU core) and the packet sizes on the maximum achievable throughput. The experiment is tested with a number of cores ranging from one to eight and different packet sizes (64, 128, 256, 512, 1024, 1500 bytes). Fig. 7 shows the results of this experiment. The results show that as the number of cores and packet size increase, the throughput increases. Therefore, these three parameters are directly proportional. For the smallest packet size considered (64 bytes), the maximum throughput attained is 12Gbps running on eight cores. Using four cores is enough to achieve line rate while considering a minimum packet size of 1024 bytes.

C. Evaluating the Impact of the Number of Hash Functions on the Throughput

Another factor that significantly impacts the performance is the number of hash functions used in the CMS. As the number of hash functions increases, the complexity of the code and the amount of computations (i.e., increment the counters, calculating the minimum) in the CMS also increase. This process happens for every received packet. This experiment evaluates the throughput while considering various numbers of hash functions (two to eight) and various packet sizes, with two and four CPU cores. Fig. 8 (a) shows the results with two cores, and Fig. 8 (b) shows the results with four cores. The results in both experiment setups show that the number of hash functions is inversely proportional to the throughput. The throughput is also affected by the packet size. The drop in the throughput is clearly observed with two cores, even with large packet sizes. However, with four cores, the performance

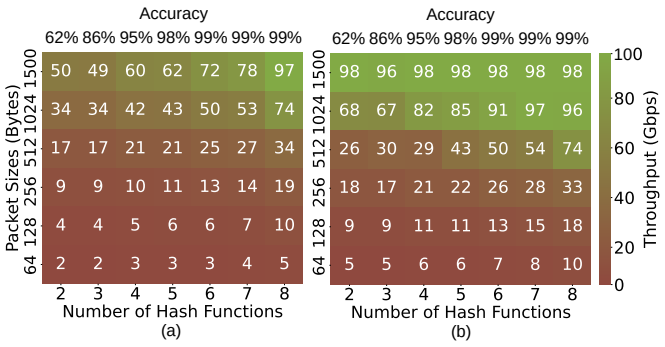

Fig. 8. (a) Throughput as a function of the number of hash functions and packet size using two CPU cores, (b) Throughput as a function of the number of hash functions and packet size using four CPU cores. The top of both figures shows the packet count accuracy δ based on the number of hash functions.

penalty is almost eliminated for larger packets, even with eight hash functions.

While the increase in the number of hash functions degrades the throughput, it offers a higher accuracy in the packet count estimation. The accuracy δ is calculated based on the number of hash functions h as follows: $\delta = 100 - \frac{100}{e^{h-1}}$. The accuracy for the different number hash functions ranging from two to eight is shown at the top of the two heatmaps in Fig. 8.

D. Performance Results of Heavy Hitter Detection with Replayed Real Traffic Data

This experiment evaluates the system with open-source real traffic captured from different Measurements and Analyses on the WIDE Internet (MAWI) datasets [20]. The MAWI packet traces are publicly available datasets of network traffic data. The packet traces are anonymized, with IP addresses scrambled, and do not contain packet payloads. Four different datasets were considered, each containing truncated packets captured on various dates in the year 2024. Datasets one through four are associated with the following dates respectively; January 27, February 10, March 23, and April 07. The packet size cumulative distribution function (CDF) of the datasets is shown in the first graph in Fig. 9 (a).

Since the packets in the datasets are truncated, a payload was generated and added to the packets based on the size of the packet. The traces are replayed using tcpreplay [21]. During the trace, a heavy hitter flow is generated. Fig. 9 (b) shows the throughput over a 30-second time frame during the experiment. All the replayed packets are processed and

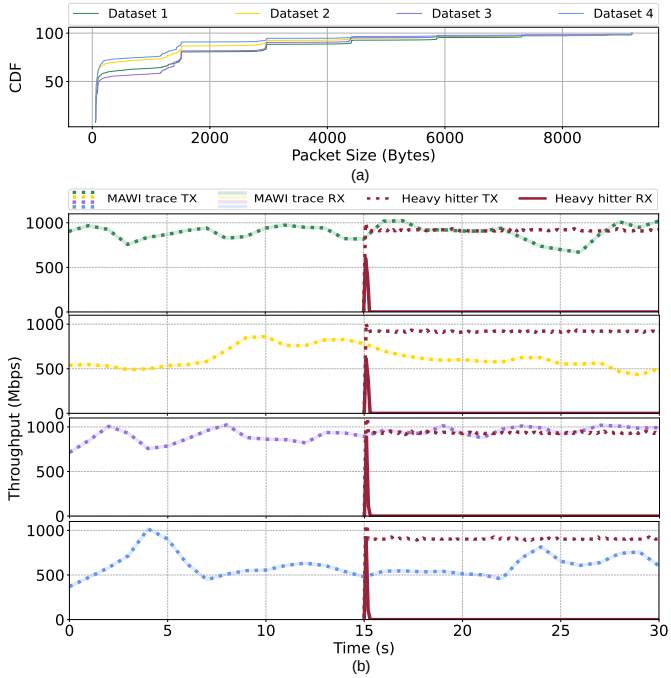

Fig. 9. (a) MAWI trace packet size distribution in four real traffic datasets populated on different dates. In some datasets, $\approx$ 40% of the packets are larger than 1024 bytes. (b) Heavy hitter mitigation while processing each trace. All the packets were processed by the pipelines.

```
alert tcp any any -> any any (msg:"Setting a flowint
    counter"; flowint:counter,notset; noalert; flowint:
    counter,=,1; sid:123;)
```

```
alert tcp any any -> any any (msg:"Adding to flowint
    counter"; flowint:counter,isset; flowint:counter,+,1;
     noalert; sid:234;)
```

```
alert tcp any any -> any any (msg:"fire this when the
    counter reaches 10000"; flowint:counter,isset;
    flowint:counter,==,10000; sid:345;)
```

Fig. 10. Suricata rules to detect a heavy hitter. The first rule creates a per-flow counter. The second rule increments the counter. The third rule alerts when the counter exceeds the threshold (10,000 packets).

reflected (there were no lost packets). Note that the traffic rates in this experiment are derived from the rates of real trace capture. The system is capable of achieving orders of magnitude higher throughput, as shown in Fig. 7.

E. Performance Comparison Against Suricata DPDK

This experiment compares the performance of the proposed system against Suricata [22]. Suricata is an open-source network IDS/IPS. It is multi-threaded and has an extensive ruleset and signature language to monitor network traffic for threats and events. In addition to the existing rules created and maintained by the community, Suricata supports scripting for custom threat detection. Suricata also supports DPDK [23], which accelerates the packet processing. For a fair comparison, the Suricata with DPDK enabled is considered.

In this experiment, a heavy hitter detection is implemented in Suricata. The rules are shown in Fig. 10. The first rule matches on any TCP flow and creates a new variable `counter` which will serve as the flow's packet counter. The second rule increments `counter` by one. The third rule alerts when `counter` reaches a threshold (10,000 packets). Note that Suricata is configured to only use the heavy hitter detection rules. All other existing rules are disabled.

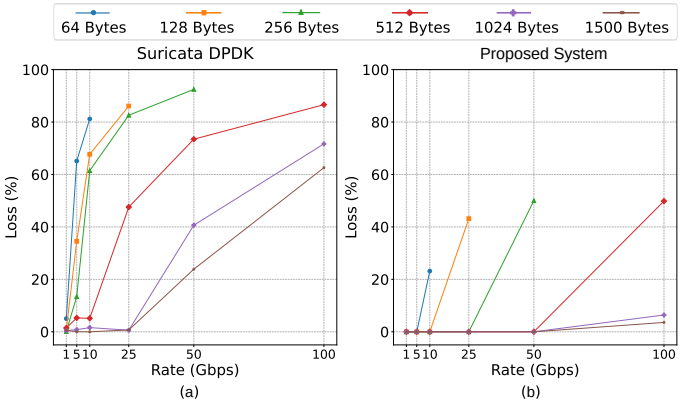

Fig. 11. Packet loss rates with different sending rates and different packet sizes using Suricata DPDK (a) and the proposed system (b). Suricata DPDK has large packet loss rates in almost all cases. The proposed system processes significantly more packets than Suricata DPDK.

TABLE I
PACKET DROP RATES WITH REAL TRAFFIC.

Trace	Replayed packets	Packet drop rate	
		Suricata-DPDK	Proposed system
MAWI 01-27-2024	95,172,026	1.59%	0%
MAWI 02-10-2024	75,845,886	0.85%	0%
MAWI 03-23-2024	87,786,793	18.01%	0%
MAWI 04-07-2024	90,774,252	19.47%	0%

Fig. 11 shows the packet loss rates of both systems. The packet loss rate in Suricata DPDK is significantly large, especially when the sending rate increases beyond 1Gbps and when the packet sizes are small. Even with large packets (e.g., 1500 bytes), Suricata dropped more than 60% of the traffic. The proposed system on the other hand has much smaller packet loss rates. With larger packets, the proposed system only dropped a small percentage at a 100Gbps rate. Note that these results are produced with four CPU cores; the results can significantly improve if more cores are used (Fig. 7).

The proposed system and Suricata-DPDK were both tested using real traffic datasets identical to those used in Section V-D. Table I shows the packet drop rates for each dataset. Suricata-DPDK's performance varies significantly based on the dataset, with packet loss rates ranging from as low as 0.85% to as high as 19.47%. The drop rate is influenced by the burstiness of the traffic, which increases the number of packets per second. In contrast, the proposed system exhibited zero packet drops, in all the tested datasets.

VI. CONCLUSION AND FUTURE WORK

This paper implemented a heavy hitter detection system using P4 and DPDK. The system uses the CMS data structure for estimating the number of packets per flow. When this number exceeds a predefined threshold, a heavy hitter is detected. While CMS introduces some estimation errors, it offers a tunable trade-off between accuracy δ and memory usage based on the number of hash functions h and the size of the register arrays. The experimental results show that increasing the number of hash functions improves the accuracy but degrades the throughput. Using two CPU cores, the performance penalty is high, especially with small packet sizes. With four CPU cores, the performance penalty is diminished even with a high number of hash functions. The results show that heavy hitters can be accurately detected even with high traffic rates. The system was also tested with real traffic and compared against Suricata DPDK. Future work includes 1) testing the system on a public cloud platform (e.g., Amazon's AWS [24]); 2) augmenting the P4 program to detect a wide range of cyberattacks; and 3) profiling the performance of P4-DPDK with various P4 programs and execution paths.

ACKNOWLEDGMENT

The authors would like to acknowledge the National Science Foundation (NSF) for supporting this work, under awards 2346726 and 2403360. The authors would also like to acknowledge the FABRIC [5] team.

REFERENCES

[1] E. Kfoury, J. Crichigno, E. Bou-Harb, P4Tune: Enabling Programmability in Non-Programmable Networks, IEEE Communications Magazine 61 (6) (2023) 132–138.
[2] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen, M. Shagam, Mitigating DNS Random Subdomain DDoS Attacks by Distinct Heavy Hitters Sketches, in: Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies, 2017, pp. 1–6.
[3] R. Donato, What is DPDK?, [Online]. Available: https://www.packetcoders.io/what-is-dpdk/, Accessed: Apr. 1, 2024 (May 2018).
[4] samiachoueiri Github, P4-DPDK-heavy-hitter-detection, [Online]. Available: https://github.com/samiachoueiri/P4-DPDK-heavy-hitter-detection.git, Accessed: Apr. 1, 2024.
[5] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, P. Ruth, FABRIC: A national-scale programmable experimental network infrastructure, IEEE Internet Computing 23 (6) (2019) 38–47.
[6] D. Ding, M. Savi, G. Antichi, D. Siracusa, An Incrementally-deployable P4-enabled Architecture for Network-wide Heavy-hitter Detection, IEEE Transactions on Network and Service Management 17 (1) (2020) 75–88.
[7] A. AlSabeh, K. Friday, E. Kfoury, J. Crichigno, E. Bou-Harb, On dga detection and classification using p4 programmable switches, Computers & Security 145 (2024) 104007.
[8] A. AlSabeh, E. Kfoury, J. Crichigno, E. Bou-Harb, P4DDPI: Securing P4-programmable data plane networks via DNS deep packet inspection, in: Network and Distributed System Security (NDSS) Symposium, 2022.
[9] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, J. Rexford, Heavy-hitter Detection Entirely in the Data Plane, in: Proceedings of the Symposium on SDN Research, 2017, pp. 164–176.
[10] DPDK, DPDK - NICs, [Online]. Available: https://core.dpdk.org/supported/nics/, Accessed: Apr. 1, 2024.
[11] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Elastic Sketch: Adaptive and Fast Network-wide Measurements, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 561–575.
[12] R. B. Basat, G. Einziger, R. Friedman, M. C. Luizelli, E. Waisbard, Volumetric hierarchical heavy hitters, in: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2018, pp. 381–392.
[13] S. Scherrer, C.-Y. Wu, Y.-H. Chiang, B. Rothenberger, Low-rate Overuse Flow Tracer (loft): An Efficient and Scalable Algorithm for Detecting Overuse Flows, in: 2021 40th International Symposium on Reliable Distributed Systems (SRDS), IEEE, 2021, pp. 265–276.
[14] E. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, IEEE Access (2021).
[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, P4: Programming Protocol-independent Packet Processors, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 87–95.
[16] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, G. Carle, High-performance Match-action Table Updates from within Programmable Software Data Planes, in: Proceedings of the Symposium on Architectures for Networking and Communications Systems, 2021, pp. 102–108.
[17] P4lang, DPDK Backend, [Online]. Available: https://github.com/p4lang/p4c/tree/main/backends/dpdk, Accessed: Apr. 1, 2024.
[18] DPDK, DPDK Pipeline Application, [Online]. Available: https://tinyurl.com/227m8n85, Accessed: Apr. 1, 2024.
[19] Pktgen, Pktgen-DPDK, [Online]. Available: https://github.com/pktgen/Pktgen-DPDK, Accessed: Apr. 1, 2024.
[20] MAWI Working Group, Packet Traces from WIDE Backbone, [Online]. Available: https://mawi.wide.ad.jp/mawi/, Accessed: Apr. 1, 2024.
[21] AppNeta, Tcpreplay - Pcap Editing and Replaying Utilities, [Online]. Available: https://tcpreplay.appneta.com/, Accessed: Apr. 1, 2024.
[22] Suricata OISF, Suricata, [Online]. Available: https://github.com/OISF/suricata, Accessed: Apr. 1, 2024.
[23] Suricata OISF, Using Capture Hardware - DPDK, [Online]. Available: https://docs.suricata.io/en/latest/capture-hardware/dpdk.html, Accessed: Apr. 1, 2024.
[24] AWS, Automate Packet Acceleration configuration using DPDK on Amazon EKS, [Online]. Available: https://tinyurl.com/3k5rbs87, Accessed: Apr. 1, 2024.