Detecting Heavy Hitters in Network-Wide Programmable Multi-Pipe Devices.

Thiago Henrique Silva Rodrigues
Department of Computer Science
Federal University of São Carlos
São Carlos SP, Brazil
tigohsr@gmail.com

Fábio Luciano Verdi
Department of Computer Science
Federal University of São Carlos
Sorocaba SP, Brazil
verdi@ufscar.br

***Abstract*—A way to contribute to network management involves detecting high-impact traffic flows, known as Heavy Hitters. Heavy Hitters are flows that account for the majority of bytes transmitted over the network, consequently consuming more resources. The use of programmable hardware, such as switches and DPUs, allows for the detection of these flows in-line rate, aiming to optimize their performance. While the literature reveals an extensive analysis of detection in single-pipe switches, this study presents two approaches to identify Heavy Hitters in programmable switches with multiple pipes. One approach has an accumulator in the switch, which centralizes data from all pipes and communicates with the control plane. In the other approach, communications with the control plane are independent for each pipe. Both approaches were developed and validated through an emulator, demonstrating effectiveness and improvement in detection in multi-pipe switches compared to single-pipe switches.**

***Index Terms*—Network monitoring, Heavy Hitters detection, multi-pipe switch, Network-Wide.**

I. Introduction

The Heavy Hitters (HH) flows represent less than 10% of the flows passing through the network; however, they consume the majority of resources, compromising the traffic of lighter flows shared on the same network [3], [5]. An effective strategy involves detecting HH in the data plane (DP) through high-speed programmable switches. This approach not only reduces communication overhead and data transfer between the DP and the control plane (CP) but also provides greater control over the DP, enabling the rapid implementation of network algorithms [2], [4], [11].

These switches utilize the concept of multi-pipes to process packets in a parallelized manner [5], where each pipe operates with its own isolated memory. Works in the literature, such as [1], [2], [6], [8], [9], do not detect HH in this context due to the complexity of multi-pipe switches. In these switches, flows can enter any pipe and switch pipes during their duration (the number of pipes can be configured). This independence of pipes leads to an incorrect detection of the actual amount of traffic passing through the switch. This problem is exacerbated when detection is performed in a Network-Wide context [7], [14].

We extended one of the works present in the literature, highlighted as a key reference for HH detection in a Network-Wide context [1]. Our work addresses the global detection of HH in programmable multi-pipe switches, using an adaptive thresholds technique. We developed an emulator for detection and devised two approaches. The "Accumulator" approach features an accumulator in the switch that calculates the sum of counts from each pipe and communicates with the CP. On the other hand, the "Local-pipe" approach does not use an accumulator in the switch, and each pipe communicates individually with the CP.

We compared the use of adaptive threshold in the switch with a fixed threshold, as well as the difference between using a multi-pipe switch and a single-pipe switch. Our goal is to detect HH while minimizing communication between the DP and the CP. Among the main contributions, we can highlight: an effective strategy for detecting and mitigating HH flows in multi-pipe switches; an adaptation of detection for larger-scale network environments; a flexible approach reducing communication overhead by adjusting thresholds that maintain precision with a high F1-Score value.

II. Related Works

The work [1] stands out for its HH detection in a Network-Wide context. The authors employ adaptive thresholds based on keys to reduce the communication overhead between the DP and the CP. Edge switches perform packet counting using hash tables for distinct flows. When a local threshold is reached, the switch communicates with the coordinator, and when the global threshold is reached, the coordinator calculates an average, refining the threshold. Our work is based on this concept, applying adaptive thresholds, network-wide detection, and communication between the DP and the CP. However, unlike the authors, we will be using multi-pipe switches.

The study [2] detects HH through time intervals, also allowing for the incremental deployment of new programmable switches in the network. Each switch dynamically maintains a list of flows exceeding the dynamic sampling threshold. After the defined time interval, it is checked whether any switch has reached its local threshold; if so, the switch reports to the central coordinator. A scan is then performed on the switches in the network, allowing estimation of the total volume and identification of HH flows in the network. Values are then reset, starting a new counting time interval. The authors

employ a different adaptive thresholds technique compared to our work, do not use multi-pipes, and address a single reporting method to the CP.

III. Heavy Hitters in Multi-Pipes

Our work stands out as the first practical study to apply HH detection in multi-pipe switches, analyzing the most efficient way to perform detection and quantifying its impact compared to a single-pipe-based architecture. Multi-pipe devices may lead to irregularities when executing applications designed for single-pipe hardware, such as HH detection and load balancing. This is due to the isolated nature of each pipe, resulting in incorrect counts or malfunctioning of the application. The most recognized multi-pipe hardware in the current market includes the Intel Tofino, with up to four pipes, and the Broadcom Tomahawk, with up to 16 pipes [10], [12], [13], [15].

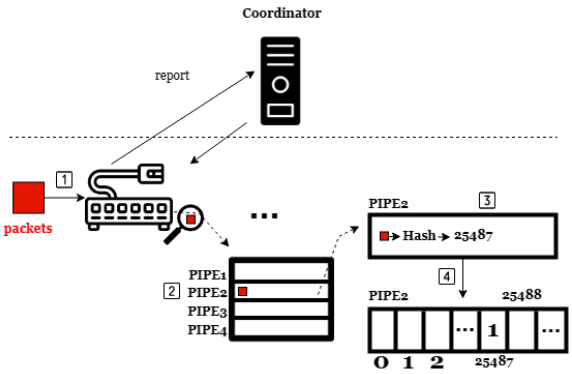

Fig. 1. Operation of the network to detect the HH.

The pipes are separate and parallel processing units within a switch that can be programmed. They enable the switch to process multiple data streams simultaneously, improving the device's processing capacity and efficiency. We created an illustrative example depicted in Figure 1, demonstrating a packet counting situation to identify a HH flow with a multi-pipe switch, similar to our approach. In the figure, there is a coordinator and a switch with four pipes receiving a packet from the network.

Initially, packets from a network flow (flow f1) arrive at an edge switch (1) and are directed to one of the pipes, arbitrarily sent to "pipe 2" (2). In the example, the threshold for a flow to be considered a HH is set at 100 packets, and f1 has 200 packets. A hash key is generated (3), and a count is initiated in a hash table at the corresponding key position (4). If all 200 packets of f1 were allocated to a single pipe, the HH flow would be counted twice. However, assuming that "pipe 2" has accumulated 50 packets from f1 and due to external factors such as network fluctuations or failures, a reconfiguration occurs, and f1 is now routed through "pipe 3", where another 50 packets are sent. In this way, the detection of HH will not occur.

This complexity intensifies in a Network-Wide detection scenario, compounded by the intricacies of multi-pipes and the challenge of packet distribution across various switches. In response, we devised two approaches for our work: **Local-pipe** and **Accumulator**. The **Local-pipe** approach triggers coordinator notifications when a pipe's local threshold is reached. Conversely, the **Accumulator** approach incorporates an accumulator in one pipe, receiving counts from all pipes and notifying the coordinator upon reaching the switch threshold. The coordinator manages a global threshold, distributed among edge switches, while each switch's threshold is apportioned among its pipes.

In both approaches, we implemented an adaptive threshold technique to find a more efficient threshold for detection. This adaptation is based on the Exponentially Weighted Moving Average (EWMA) of local and global counts, considering both past and current values. Whenever the global threshold is reached, the switch's threshold is adjusted. To achieve this, a smoothing parameter is employed, controlling the weight assigned to the data. This parameter should always be between zero and one, with values closer to zero giving more weight to historical data, while values closer to one prioritize recent data.

An example is if the threshold is set too low, a considerable number of potential HH could be false positives, likely capturing a significant portion of the network flows. Through adaptive thresholds, we can dynamically calibrate local thresholds, enabling the discovery of ideal values for each switch and each flow. This allows for a threshold value closer to a true HH in the network.

A. Local-pipe

In the Local-pipe solution, packet counting is performed individually by each pipe. In Figure 2, we have a programmable switch with four pipes, where each pipe has a threshold of 250 packets, resulting in a switch threshold of 1000 packets. The table inside the pipe represents a hash table, where each row corresponds to a hash position for a specific flow.

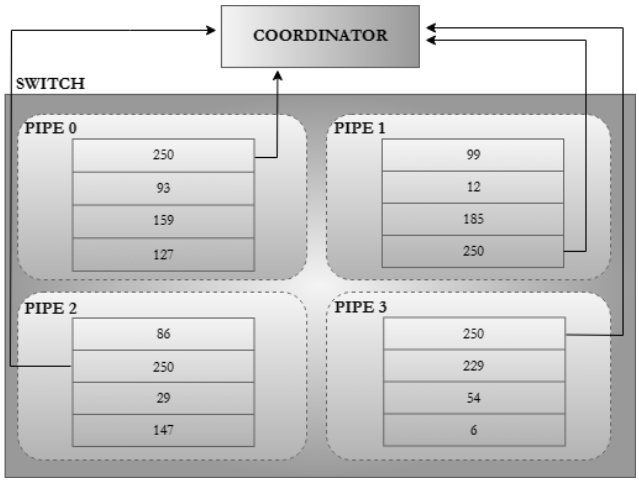

Fig. 2. Execution of Local-pipe on a switch.

When a packet from a specific flow reaches the switch, it is directed to one of the existing pipes, where the flow

count takes place. When a pipe reaches its threshold, the attained value and the hash related to the flow are sent to the coordinator. In "pipe 1", one of the flows reached the pipe threshold, reported the information to the CP, and the value in the position of the pipe that reached the threshold is reset to zero.

If the pipe threshold for a specific hash is reached again, this value will be sent to the CP and added to the existing value at that position. This updating process continues until the global threshold is reached or exceeded for that specific hash, and when it happens, it is indicative that a HH flow may have been identified.

B. Accumulator

In the Accumulator solution, packet counting is also done per pipe, but there is an accumulator. One of the pipes of the switch is designated to accumulate packet counts from all pipes for each data flow. Therefore, the pipe allocated as the accumulator will perform two functions: count the individual packets transmitted through it and accumulate packet counts from all pipes of the switch.

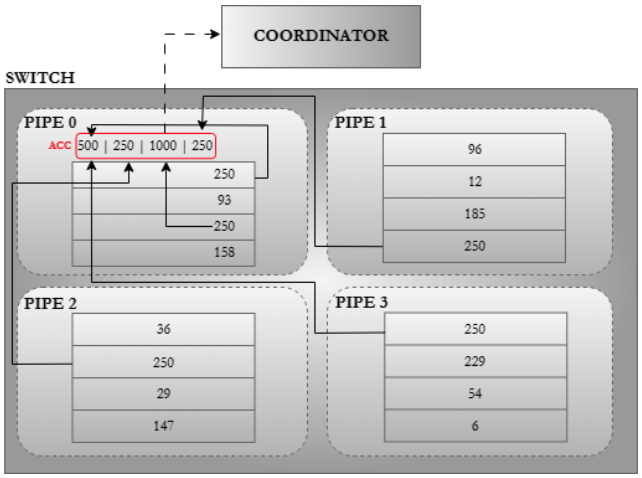

Fig. 3. Execution of Accumulator on a switch.

In Figure 3, there is a scenario with a coordinator and a switch having four pipes. The switch has a threshold of 1000 packets, while the pipes have a threshold of 250 packets. In our example, "pipe 0" has a hash table but also acts as the accumulator (ACC), which has the same threshold as the switch. When a pipe reaches its threshold, the information is recirculated within the switch, reaching the ACC at the position corresponding to the flow's hash.

In "pipe 2", a flow has reached the pipe threshold. Its hash and the attained value are recirculated to the same position in the ACC and added to the value present in that position. The count of "pipe 2" is reset to zero. The same process occurs twice in "pipe 0", and the value is recirculated to the pipe itself. One of the flows reached the threshold for "pipe 0", and when added to the existing value in the ACC, it reaches a count of 1000 packets. Therefore, the attained value and its hash are forwarded to the coordinator, as indicated by the dashed line. Since the Accumulator includes an accumulator with a threshold greater than the individual thresholds of the pipes, we observe a reduction in communications with the CP compared to Local-pipe, demonstrating it to be a more efficient solution than Local-pipe.

IV. Results

To conduct our experiments, we developed an emulator using the Python language, which supports a configurable number of switches, as well as a configurable number of pipes per switch. For our experiments, we defined a scenario with three to six network switches, all multi-pipes, supporting from two to 16 pipes. To emulate a real-world traffic distribution, we used a CAIDA trace containing 28 million trace packets.

The hash creation is performed using the MD5 method with 16 bits. We defined a flow as the four fields of the TCP/IP header (source IP, destination IP, source port, and destination port). Thus, in the trace used, there are 1.8 million distinct flows. Additionally, we set the threshold of 10,000 packets for a flow to be considered a HH, resulting in 299 flows in the CAIDA trace after this filtering. To represent the variation of the flow across the pipes, our emulator switches between the flow of the pipe whenever it exceeds half of the packet threshold of the switch for a HH in a pipe, randomly choosing the next pipe.

In our experiments, we have switches with fixed thresholds and adaptive thresholds. Additionally, the same experiments were conducted for switches with a single-pipe-based architecture, with a fixed threshold (baseline). This allowed us to evaluate that even when applying HH detection in multi-pipe switches, a high F1-Score could be maintained compared to our baseline, reaching a value of 0.99, both with Local-pipe and Accumulator. Furthermore, with the Accumulator, we reduced the number of reports to the CP compared to our baseline.

A. Sensitivity to Pipes

The aim of our first experiment was to find the best smoothing factor for each number of pipes. In the experiment, we explored smoothing in a range from 0.2 to 0.99 while varying the number of pipes from two to 16. These values were analyzed with the number of reports to the CP. We concluded that the smoothing is linked to the quantity of active pipes in the switch; that is, each number of pipes has a more suitable smoothing factor to reduce the number of reports sent to the controller.

In the Accumulator approach, a switch with two pipes performs better with a smoothing factor of 0.8 (69 reports); conversely, when dealing with a 16-pipe switch, the ideal would be to use a smoothing factor of 0.99 (116 reports). The scenario flips when we employ eight pipes. In this context, the number of reports decreases when the smoothing gives more weight to historical data, 0.4 in this case (82 reports). In the Local-pipe approach, we observe that the communication overhead is much higher compared to the Accumulator since Local-pipe does not incorporate an accumulator in the switch.

If we consider the smoothing that generated the lowest report value for Local-pipe (0.2 with two pipes) and compare it with the same smoothing for the Accumulator, we observe an increase in reports for Local-pipe by 3277% (Accumulator 103 reports and Local-pipe 3481 reports).

In both approaches, we observe that each number of pipes has an ideal smoothing factor, making it challenging to define a universal value for any pipe configuration. However, we notice that the higher the number of pipes, the higher the likelihood of increasing communication overhead because more pipes lead to lower thresholds, causing them to reach their thresholds more quickly.

B. *Influence of Pipes*

In this experiment, we analyzed the impact of the number of pipes when using a switch with a fixed threshold compared to a switch with an adaptive threshold. We used three switches, adopting a smoothing factor of 0.8 with a threshold of 10,000. Observing the Accumulator in Figure 4, we notice that as the number of pipes increases, there is a reduction in the number of reports for a switch with a fixed threshold. On the other hand, when the switch has an adaptive threshold, we observe an increase in the number of reports as we increase the number of pipes used.

However, although the increase is small, the number of reports is still significantly lower when compared to the fixed threshold solution, confirming the advantage of using adaptive thresholds as the number of pipes in the switch increases. For example, considering four pipes in the Accumulator, we have a reduction of approximately 95% in the reports sent to the coordinator when analyzing both thresholds (adaptive threshold with 88 reports and fixed threshold with 1672 reports). Similarly, when using the best-case scenario for switches with a fixed threshold, with 16 pipes (1383 reports), it surpasses the worst case of the switch with adaptive thresholds (186 reports) by 643%.

In the Local-pipe approach, as illustrated in Figure 4, the scenario is reversed. As expected, Local-pipe exhibits an excessively higher number of reports to the controller. As we increase the number of pipes, we observe an increase in reports, regardless of using a fixed threshold or an adaptive threshold.

Despite both solutions showing an increase in the number of reports as we increase the number of pipes, the fixed threshold for the switch still tends to have more reports than using an adaptive threshold, as we can observe when applying eight or 16 pipes. Since the global threshold is divided among the switches and, consequently, among the pipes, the more pipes are used, the lower the local threshold for each one, leading to reaching them more quickly.

The approach described in [1] resembles the Local-pipe, as whenever the switch's threshold is reached, a report is forwarded to the coordinator. In this scenario, analyzing Figure 4 and comparing the single-pipe switch with a fixed threshold (2035 reports) with our methodology, involving a multi-pipe switch with two pipes and an adaptive threshold (69 reports), we observe a significant reduction of 96% in the number of reports.

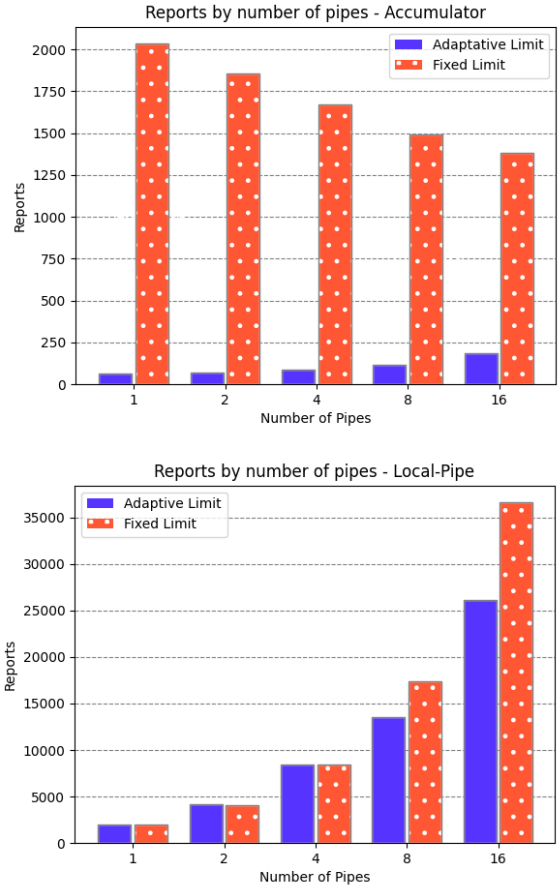

Fig. 4. Relationship between the quantity of reports per pipe.

V. Conclusion

The detection of HH stands out as a crucial method in the management and defense of modern networks. Our analysis, using real traffic, reveals a significant reduction in communication overhead, mitigating potential overloads, latency, and delays. This improvement is achieved by implementing an accumulator in one of the pipes, along with the use of adaptive thresholds. When a fixed threshold is employed, we observe an average increase of 92% in communication overhead between the data plane and the control plane. The Accumulator approach, with adaptive thresholds, demonstrated the ability to maintain a high F1-Score, compared to our single-pipe switch approach, while reducing communication overhead by 96.65%. Regarding adaptive thresholds, we observe that the ideal smoothing factor should vary according to the number of pipes used in the switch to achieve better results.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPESP (process 2021/14297-1) and CNPq (process 305665/2022-7).

REFERENCES

[1] HARRISON, R. et al. Network-wide heavy hitter detection with commodity switches. In: Proceedings of the Symposium on SDN Research. [S.l.: s.n.], 2018. p. 1–7.

[2] DING, D. et al. An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection, IEEE Transactions on Network and Service Management, IEEE, v. 17, n. 1, p. 75–88, 2020.

[3] LIN, Y.-B.; HUANG, C.-C.; TSAI, S.-C. Sdn soft computing application for detecting heavy hitters. IEEE Transactions on Industrial Informatics, IEEE, v. 15, n. 10, p. 5690–5699, 2019.

[4] BASAT, R. B. et al. Designing heavy-hitter detection algorithms for programmable switches. IEEE/ACM Transactions on Networking, v. 28, n. 3, p. 1172-1185, 2020.

[5] CHIESA, M.; VERDI, F. L. Network Monitoring on Multi-Pipe Switches. Proc. ACM Meas. Anal. Comput. Syst. 7, 1, Article 8 (March 2023), Sigmetrics, 31 pages.

[6] SIVARAMAN, V. et al. Heavy-hitter detection entirely in the data plane. In: Proceedings of the Symposium on SDN Research. [S.l.: s.n.], 2017. p. 164–176.

[7] BASAT, R. B. et al. Network-wide routing-oblivious heavy hitters. In: Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems. 2018. p. 66-73.

[8] TURKOVIC, B.; OOSTENBRINK, J.; KUIPERS, F. Detecting heavy hitters in the data-plane. arXiv preprint arXiv:1902.06993, 2019.

[9] BASAT, R. B. et al. Memento: Making sliding windows efficient for heavy hitters. In: Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies. 2018. p. 254-266.

[10] AGRAWAL, A.; KIM, C. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In: 2020 IEEE Hot Chips 32 Symposium (HCS). IEEE Computer Society, 2020. p. 1-32.

[11] BEN-BASAT, R. et al. Efficient measurement on programmable switches using probabilistic recirculation. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). IEEE, 2018. p. 313-323.

[12] Intel. Intel Tofino 2 Intelligent Fabric Processor Brief, 2022. https://www.intel.com/content/www/us/en/products/ network-io/programmable-ethernet-switch/tofino-3-brief.html.

[13] Broadcom. Tomahawk3 - BCM56980 12.8 Tb/s Multilayer Switch , 2019. https://docs.broadcom.com/doc/56980-DS.

[14] TANG, L.; HUANG, Q.; LEE, P. PC. A fast and compact invertible sketch for network-wide heavy flow detection. IEEE/ACM Transactions on Networking, v. 28, n. 5, p. 2350-2363, 2020.

[15] WHEELER, B. Tomahawk 4 switch first to 25.6 Tbps. Microprocessor Report, 2019.