# Implementing a Heavy Hitter Detection on the NetFPGA OpenFlow Switch

Theophilus Wellem*‡, Yu-Kuen Lai†, Chung-Hsiang Cheng†, Yung-Chuan Liao†,
Li-Ting Chen†, and Chao-Yuan Huang†
*Department of Electronic Engineering, †Department of Electrical Engineering
Chung Yuan Christian University, Zhongli 32023, Taiwan
Email: {g10202604, ylai, g10578017, g10578009, g10578002, g10279008}@cycu.edu.tw
‡Department of Informatics
Satya Wacana Christian University, Salatiga 50711, Indonesia
Email: theophilus.wellem@staff.uksw.edu

***Abstract*—Heavy hitter detection is an important task in many network security and traffic measurement applications. In this work, we implement a heavy hitter detection accelerator based on the Count-Min sketch algorithm inside the NetFPGA-10G OpenFlow switch. By using only a small amount of extra memory and logic resources, the OpenFlow switch is capable of detecting the heavy hitter flows accurately without the intervention of the host CPU and the OpenFlow controller. The sketch data structure and identified heavy hitter list can be sent to the controller for further advanced processing.**

***Index Terms*—Heavy hitter; Count-Min sketch; OpenFlow switch; NetFPGA-10G.**

## I. Introduction

The task of detecting heavy hitters is important for network management and many network monitoring applications [1]. Heavy hitters are the flows that use more than a certain fraction of the total link capacity during a measurement interval. Given an input packet stream $\sigma = \{(k_i, v_i) \mid i \in [1, N]\}$ of $N$ packets, $k_i$ represents the key of the $i^{\text{th}}$ input packet, and $v_i$ represents its corresponding value (packet count or size) contributed by the $i^{\text{th}}$ input packet. For a given threshold $\phi$, $0 < \phi \leq 1$, a heavy hitter is defined as a flow identified by key $K$, whose total value $\sum_{\{i|k_i=K\}} v_i$ is at least $\phi \sum_{i=1}^{N} v_i$ [2], [3]. This is known as the $\phi$-heavy hitter [4]. In software-defined networking (SDN), monitoring applications (e.g. an anomaly detector) can be executed on the top of an SDN controller. Typically, wildcard rules are installed by the controller for traffic monitoring purpose [5]. Similar to the forwarding rules, the monitoring rules are placed in the ternary content-addressable memory (TCAM) that allows wildcard matching. However, due to the space constraint of the available TCAM resource, the measurement accuracy is sometime limited. Moreover, the main purpose of the TCAM in OpenFlow switch is to store the rules for forwarding traffic, not for monitoring purpose.

In this work, we implement a heavy hitter detector based on the Count-Min (CM) sketch algorithm [4] in the NetFPGA-10G OpenFlow switch [6]. The purpose is to allow the switch to monitor the heavy hitter flows without sacrificing the flow table entries intended for forwarding purpose.

## II. System Architecture and Implementation

Based on the original architecture of the NetFPGA-10G OpenFlow switch [6], the proposed heavy hitter detection module is placed after the header parser of the packet preprocessor module and in parallel to the OpenFlow datapath. As shown in Fig. 1, there are three main blocks in the heavy hitter detection module: CM sketch, Bloom filter, and heavy hitter FIFO. For each input packet, the header parser inside the packet preprocessor module extracts the fields from the packet header. The source IP address and packet length are taken as the inputs to the CM sketch module for updating the sketch. The CM sketch uses a two dimensional array of counters with $d$ rows and $w$ columns. After the update process, the sketch is then queried to estimate the total traffic volume contributed by the source IP address. If this value is greater than or equal to a threshold, then the corresponding source IP address is declared as a heavy hitter, and processed to the Bloom filter. The threshold value can be determined independently by the system administrator, and written from software to a register in the hardware. For those source IP addresses which are exceeding the threshold, a Bloom filter is used to check their presence in the heavy hitter FIFO. The purpose is to prevent a duplicated source IP address stored in the heavy hitter FIFO. If the source IP address does not exist in the Bloom filter, then the Bloom filter is updated, and the source IP address is stored in the heavy hitter FIFO. The size of Bloom filter and number of hash functions used for the Bloom filter must be chosen properly to reduce the false positive rate. At the end of measurement interval, the heavy hitter FIFO is read

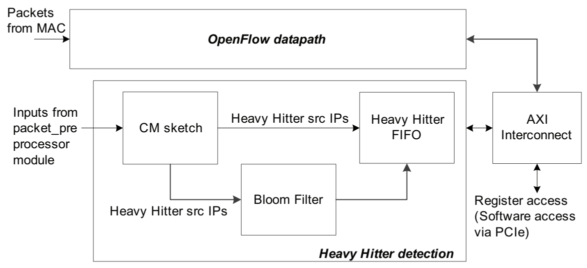

Fig. 1. Hardware blocks for the heavy hitter detection module.

TABLE I
ORIG. OF SWITCH AND OF SWITCH+HH LOGIC RESOURCES UTILIZATION

| Resources | Total resources | Orig. OF | % | OF+HH | % |
|---|---|---|---|---|---|
| Registers | 149,760 | 69,340 | 46% | 71,581 | 47% |
| LUTs | 149,760 | 63,781 | 42% | 65,885 | 43% |
| BRAMs | 324 | 251 | 77% | 262 | 80% |

by the software running on host CPU to obtain the list of heavy hitter source IP addresses. The list is then sent to the OpenFlow controller.

The CM sketch is constructed with three sets of memory array ($d = 3$). Each memory array consists of 4,096 32-bit counters ($w = 4,096$). The total memory space required for the CM sketch is 48 KB. The size of the Bloom filter is 2 KB. The size of the heavy hitter FIFO is 8 KB and can store up to 2,048 heavy hitter source IP addresses during a measurement interval. Hash functions based on the $H_3$ class are implemented for the Bloom filter and CM sketch. The hardware design is synthesized by using Xilinx ISE/EDK design tools. The logic resources utilization of the original NetFPGA-10G OpenFlow switch and the proposed design are shown in Table I. The utilization of slice registers and LUTs are increased by only 1%. The consumption of BRAM resources has increased 3% more compared to that of the original NetFPGA-10G OpenFlow switch.

## III. Evaluation

The system is evaluated through trace-driven tests by using three different MAWI traces. Each trace is 15 minutes long. The trace is recorded as a part of 24h-long trace in the transit link of WIDE to the upstream ISP that is collected at MAWI's samplepoint-F [7]. We categorize the traces into Small, Medium, and Large, based on the number of distinct source IP addresses in the trace. The measurement interval is set to 60 seconds in the experiments. In this interval, the number of distinct source IP addresses in these three traces is approximately 15k, 20.5k, and 27k. Based on the measurement interval of 60 seconds, there are total of 15 intervals in each trace. For each trace, the threshold is chosen such that the heavy hitters are identified by flows whose byte counts are

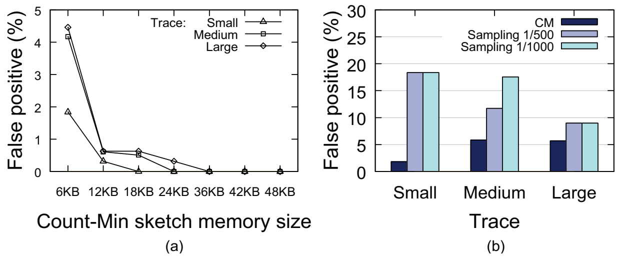

Fig. 2. (a) False positive of the system with increasing memory size. CM sketch memory size of 48 KB is used for the hardware implementation. (b) False positive comparison of the system and sampling to exact heavy hitter measurement results.

greater than $1\%$ of the average total bytes of these 15 intervals. The number of identified heavy hitters in the 60 seconds interval is about 9 to 25. Fig. 2a shows the false positive results with different Count-Min sketch memory size. Memory size of 48 KB corresponds to $3 \times 4096$ counters in the sketch. The false positive rate decreases to nearly zero with memory size of 36 KB for Large trace. There are no false negatives when the memory size is larger than 6 KB. The false positives of the proposed system and sampling-based method, compared to the exact heavy hitter measurement results, are shown in Fig. 2b. In conclusion, the proposed system can achieve lower false positive rate and no false negative in all experiments compared to sampling.

## Acknowledgment

This research was funded in part by the National Science Council, Taiwan, under contract number: MOST 105-2221-E-033-049 and MOST 104-2221-E-033-007.

## References

[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02. New York, NY, USA: ACM, 2002, pp. 323–336.

[2] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04. New York, NY, USA: ACM, 2004, pp. 101–114.

[3] D. Tong and V. Prasanna, "Online heavy hitter detector on FPGA," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2013, pp. 1–6.

[4] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, April 2005.

[5] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 25–30.

[6] T. Yabe, "OpenFlow Implementation on NetFPGA-10G Design Document," https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-OpenFlow-Switch.

[7] MAWI Working Group, "MAWI Working Group Traffic Archive," http://mawi.wide.ad.jp.