Revisiting Heavy-Hitters: Don't count packets, compute flow inter-packet metrics in the data plane

Suneet Kumar Singh
University of Campinas
ssingh@dca.fee.unicamp.br

Christian Rothenberg
University of Campinas
chesteve@dca.fee.unicamp.br

Marcelo Caggiani Luizelli
Federal University of Pampa
marceloluizelli@unipampa.edu.br

Gianni Antichi
Queen Mary University of London
g.antichi@qmul.ac.uk

Gergely Pongracz
Ericsson Research
gergely.pongracz@ericsson.com

CCS CONCEPTS

• **Networks** → **Network algorithms**; **Network monitoring**; **Programmable networks**; **Data path algorithms**.

KEYWORDS

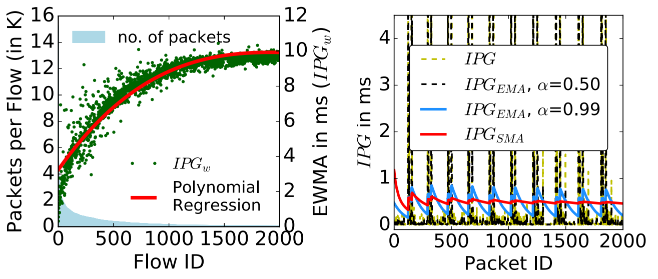
Network Monitoring; P4; Programmable Networks; Network Algorithms

ACM Reference Format:
Suneet Kumar Singh, Christian Rothenberg, Marcelo Caggiani Luizelli, Gianni Antichi, and Gergely Pongracz. 2020. Revisiting Heavy-Hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20 Demos and Posters), August 10–14, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3405837.3411388

1 INTRODUCTION

Detecting Heavy Hitter (HH) flows, i.e., flows exceeding a predetermined threshold in a time window, is a fundamental task as it enables network management and security applications like DoS attack detection/prevention, flow-size aware routing, and QoS. The recent breakthroughs of programmable data planes has provided an unique opportunity: detect them directly in the data plane to enable fast control decisions. State-of-the-art solutions leverage either probabilistic data structures [1, 2] or prefix trees [3] to store flow counters directly in the programmable pipeline of switches. However, the former approach still depends on the intervention of a central controller to identify the HH flows from the hash-buckets, thus partially diminishing the fast data plane reaction. The latter approach instead, while successfully implemented on FPGA, is not yet a feasible solution for today's programmable ASICs due to limited accesses to registers [4].

In this poster, we explore the possibility to take a completely different direction: keep track of per-flow Inter Packet Gap (IPG) metrics instead of counters. IPG analysis has been already employed in some networking applications, but, to the best of our knowledge, this work is the first to apply it to the HH detection problem. At the heart of the proposed method is the observation that HH flows can be characterized by small IPG metrics calculated as a function (e.g. weighted average) of the inter-packet time intervals. The "heaviness" (i.e. throughput over time) of a packet flow can be approximated by relating the average packet size to the observed IPG values. Notably, this approach does not require a measurement interval to be set upfront, thus eliminating common shortfalls of windows-based algorithms [5].



(a) (b)

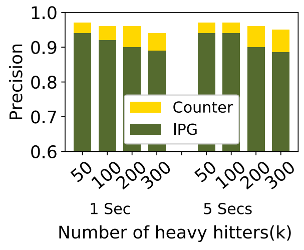
Figure 1: (a) $IPGf_w$ vs Throughput, and (b) Larger α smooths out abrupt IPG changes.

2 IS IPG METRIC FIT TO DETECT HH?

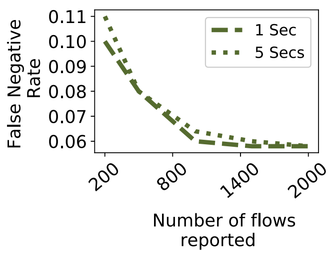
For the derivation of a suitable flow IPG metric that represents the heaviness of a given network flow $f \in F$, we consider an exponential weighted moving average (EWMA) function of the observed IPG values. The IPG metric is updated every time a packet of f gets into the switch pipeline. The current $IPGf_c$ metric is given by the difference between the last seen packet timestamp (TSf_l) and the current one (TSf_c). Next, an EWMA metric $IPGf_w$ can be calculated:

$$IPGf_w = \alpha \cdot IPGf_{w-1} + (1 - \alpha) \cdot IPGf_c, \quad (1)$$

where α is the degree of weighting decrease and $IPGf_{w-1}$ is the last noted IPG_w. As usual in EWMA, the choice of α impacts the timeliness and precision/accuracy for our HH detection objectives. Figure 1b shows how different α values influence the $IPGf_w$ metric of a real network trace. We observe that $\alpha = 0.99$ presents a good fit for HH detection.



(a) m=8000



(b) k=200

Figure 2: Performance of Space-Saving using IPG and Counter based approach: (a) for $m = 8000$ memory slots, (b) over-reporting flows to detect k heavy hitters

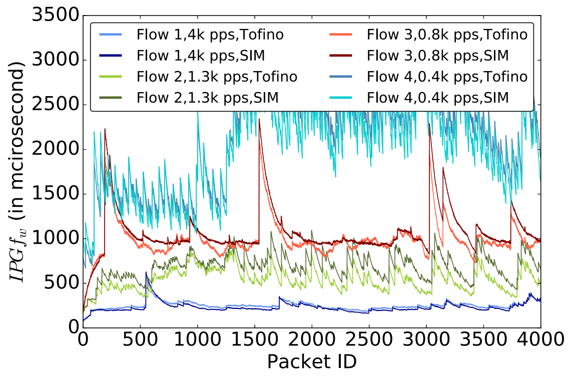


Figure 3: Running $IPGf_w$ values for different throughput flows using Tofino HW and Simulator (SIM).

Hypothesis validation. To validate our hypothesis, we analyze the relationship between $IPGf_w$ and flow throughput. We use CAIDA traffic traces [6] from a 10Gb/s ISP backbone link and plot (Fig. 1a) the number of packets per-flow against the $IPGf_w$ calculated at the end of 5 secs time window (TW), i.e., around 180K flows and 2.7 M packets. The Fig. 1a shows the strong correlation between $IPGf_w$ and flow throughput, a behaviour that persists for different TW sizes.

3 PRELIMINARY EVALUATION AND DESIGN CHALLENGES

We evaluate the HH detection effectiveness of the proposed IPG-based method with a Space-Saving (SS) algorithm [7] implementation in Python. We use CAIDA [6] traces considering the usual definition of a flow through its 5-tuple key. A single Table is maintained with m number of memory slots. Each slot contains three values: 32-bit flow ID, 16-bit $IPGf_{w-1}$, and 16-bit TSf_l. Choosing less number of bits for TSf_l (μs) may miss the majority of long inter burst gaps. When a packet arrives, SS inserts the new flow with $IPGf_{w-1}$ initialized and TS_l set to the current ingress TSf_c. Upon each packet arrival, $IPGf_{w-1}$ is updated as per Eq. 1. When the table is full, SS replaces the entry with the largest $IPGf_{w-1}$.

Figure 2a presents the observed precision (i.e., ratio of correctly reported HHs and total number of reported flows) of counter-based and IPG-based methods for $m = 8K$. In Fig. 2b, SS using IPG provides significant improvement reporting around 2K flows for detecting top 200 flows. The results suggest that IPG metrics are a strong candidate for HH detection and other traffic management applications.

P4 Hardware Implementation and Evaluation. For our prototype implementation on a Tofino hardware (HW) switch, we leverage the HeavyKeeper (HK) [8] algorithm, which is amenable to programmable HW. The P4 implementation in HW of the HK pipeline with EWMA calculation requires overcoming some engineering challenges: **(i)** limited memory resources, **(ii)** limited number of R/W access of a register, **(iii)** arithmetic and comparison operations of non-integer values, and **(iv)** fixed number of pipeline stages.

Since multiplication and division present restrictions in P4 register actions, the resulting EWMA requires approximation. In addition, as any register can be accessed only once in packet's lifetime, to replace old entries with new ones in the case of flow ID updates, after comparing the $IPGf_w$ register value with a pre-defined threshold value, we rely on a packet re-submission. Finally, we able to successfully compiled P4 TNA code on a Tofino HW.[1]

Figure 3 presents the run-time IPG_w values in the Tofino HW and Simulator (SIM) implementations for four different flow throughput rates. Figure 3 confirms that Tofino HW and SIM do not exhibit major differences, confirming that $IPGf_w$ can be used to classify HH flows directly in existing programmable HW switches.

Future Work. One current limitation is that certain types of short lived HHs can be missed using the current IPG metrics. To this end, we are extending the method to compactly record the flow duration as a parameter to decide on HH. This additional metric will provide more flexibility to detect HHs, avoid inconsistent HH flows [9] in time and can easily detect hidden HH flows [5]. Another ongoing work is on bit space optimization considering IPG slot/range identified just with 8 bits instead of maintaining 16 bits for $IPGf_{w-1}$. On the performance path, we are doing latency evaluation for 10Gbps traces using T-REX and OSNT. Last but not least, the HH method will be incorporated in a hybrid SW/HW VNF offloading solution related to 5G mobile core functions, where HH flows will be kept in the HW pipeline while lightweight flows will be handled by x86 SW.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers. This work is partially sponsored by the UK's Engineering and Physical Sciences Research Council (EPSRC) under the EARL project (EP/P025374/1) and the EPSRC New Investigator Award NEAT (EP/T007206/1). This work is also partially supported by CNPq (427814/2018-9, 310930/2016-2), FAPESP (2018/23092-1) and by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.66.

[1]https://github.com/intrig-unicamp/P4-HH

REFERENCES

[1] G. Vorsanger V. Sekar Z. Liu, A. Manousis and V. Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. *SIGCOMM*, 2016.

[2] O. Rottenstreich S. Muthukrishnan V. Sivaraman, S. Narayana and J. Rexford. Heavy-Hitter Detection Entirely in the Data Plane. *SOSR*, 2017.

[3] H. Wang A. Moore J. Korenek J. Kucera, D. A. Popescu and G. Antichi. Enabling Event-Triggered Data Plane Monitoring. *SOSR*, 2020.

[4] C. Kim J. Lee R. Miao, H. Zeng and M. Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. *SIGCOMM*, 2017.

[5] A. W. Moore G. Bianchi S. Galea, G. Antichi and R. Bifulco. Revealing Hidden Hierarchical Heavy Hitters in network traffic. *SIGCOMM Poster*, 2018.

[6] CAIDA. The CAIDA UCSD Anonymized Internet Traces, 2016.

[7] D.Agrawal A.Metwally and A.ElAbbadi. Efficient computation of frequent and top-k elements in data streams. *International Conference on Database Theory. Springer*, 2005.

[8] H. Zhang H. Li S. Uhlig. S. Chen L. Uden J. Gong, T. Yang and X. Li. HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows. *USENIX Annual Technical Conference*, 2018.

[9] S. Bhattacharyya P. Thiran K. Salamatian K. Papagiannaki, N. Taft, L. Uden, and C. Diot. A Pragmatic Definition of Elephants in Internet Backbone Traffic. *SIGCOMM Workshop on Internet Measurement*, 2002.