Towards In-Network Time-Decaying Aggregates for Heavy-Hitter Detection

Xin Zhe Khooi, Levente Csikor, Min Suk Kang
National University of Singapore

Dinil Mon Divakaran
Trustwave

ABSTRACT

Keeping track of *heavy hitters (HH) entirely in the data plane* is an all-important aspect of many real-time monitoring tasks (e.g., load-balancing, attack detection). Existing interval-reset-based sketch and hash table approaches are incapable of delivering consistent and high accuracy when operating in heterogeneous scenarios where various applications with different purposes require the flows to be tracked at different time scales, not to mention their dependence on the control plane for data structure management.

We propose HashAge and SkAge, novel in-network time-decaying algorithms for hash table- and sketch-based HH detection. We show that our proposed algorithms offer consistent and higher detection accuracy while operating in heterogeneous demands whilst not requiring any data structure management from the control plane at all.

CCS CONCEPTS

• **Networks** → **Network monitoring**; **Network management**.

KEYWORDS

network monitoring, heavy-hitter detection

ACM Reference Format:
Xin Zhe Khooi, Levente Csikor, Min Suk Kang and Dinil Mon Divakaran. 2020. Towards In-Network Time-Decaying Aggregates for Heavy-Hitter Detection. In *ACM Special Interest Group on Data Communication (SIGCOMM '20 Demos and Posters), August 10–14, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3405837.3411402

1 INTRODUCTION

Different network applications can have heterogeneous notions of heavy hitters (HH). Network administrators may consider top-3 heavy flows in the last 5 seconds as HH for a DDoS defense application, while top-10 HH in every 10 seconds for a traffic engineering application, e.g., a load balancer. Existing in-network approaches (e.g., [7, 9, 11]), however, can satisfy only one application demand at a time as they have the problem of *interval management*,— i.e., the flow monitoring operation is divided into static time intervals and the data structure is reset in between. When the requirements for HH detection vary, operators may not be able to find a single specification to run in his network.

A naïve approach for diverse HH detection requirements is to simply run multiple, independent instances in parallel for all the network applications. Or, one can also execute a single, common heavy-hitter detection with a very short (e.g., 0.1 seconds) interval in the data plane and implement diverse heavy-hitter detection functions separately in the control plane. These naïve approaches, however, create severe performance challenges in existing programmable switches, e.g., parallel data structures require much more memory than the available (~1.4 MB per stage [5]). Moreover, the more instances we run the higher the overhead in the control plane is due to frequent data structure managements (e.g., resets). While utilizing a sliding window over the intervals seems a viable workaround, recent proposals are marked either slow and space-inefficient [1], or do not allow sufficiently large queries [2]. Thus, a more attractive *single, generic, in-network* HH detection algorithm is required that offers accurate detection performance for diverse HH notions.

Here, to resolve this issue, we design and prototype (in P4 [4]) HashAge and SkAge, novel in-network time-decaying algorithms (TDA) for hash table- and sketch-based heavy-hitter detection. Although TDA is known in streaming data-processing domain, its in-network implementation with line-rate performance is challenging due to the limited operations available and the wrap-around problem of the high-resolution clocks in today's programmable switches. Overcoming these challenges, we show that our algorithms offer consistent and, in most cases, higher detection accuracy compared to their interval-reset counterparts when queries with arbitrary HH demands can arrive at arbitrary times.

2 TIME-DECAYING AGGREGATES

While there have been several works on algorithms to efficiently answer streaming queries under time decay (e.g.,[8]), no such decay functions are readily available in programmable switches due to the *(i)* lack of support for floating point operations [10]. Furthermore, for TDAs to operate, we also need to introduce *(ii)* the notion of time to the data structures.

For *(i)*, we adapt binary *right shift* operations in our algorithms (which is a good approximate to exponential decay), while to resolve *(ii)*, we first divide the precise timestamps of the high-resolution clocks on the commodity programmable switches into broader observation phases, say, 10 seconds. When a flow arrives at a switch, we store the actual observation phase next to its counts. Whenever the clock wraps around (i.e., the new observation phase becomes less than the previous), we skew all consecutive observation phases accordingly. For example, for a 30 second time span, we divide it into $\phi = 3$ observation phases ($\omega = 1, 2, 3$), 10 seconds each, and we define a global observation phase as Ω. If ω becomes 1 again, Ω will be adjusted with #wrap-arounds $\times\ \phi$, i.e., to 4 after the first wrap around. Next, we give a brief overview of our hash table- and the sketch-based algorithms.

SIGCOMM '20 Demos and Posters, August 10–14, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8048-5/20/08…$15.00
https://doi.org/10.1145/3405837.3411402

Algorithm 1 Eviction policy in HashAge, *Input:* two tuples of entries F and T, current observation ID Ω.

```
1: procedure COMPARE(F, T)
2:     F.Cnt >> (Ω − F.Ω)          # decay F
3:     T.Cnt >> (Ω − T.Ω)          # decay T
4:     if F.Cnt > T.Cnt then       # compare counts
5:         swap(F, T)              # swap
6:     end if
7: end procedure
```

Algorithm 2 Update in SkAge, *Input:* incoming packet's data F, set of independent hash functions $\mathcal{H}_e$.

```
1: procedure UPDATE((F))
2:   for i ← 1 to e do
3:     CMS[H_i(F)] >> (Ω − F.Ω)    # decay counts
4:     CMS[H_i(F)] ++              # Increment counts
5:   end for
6: end procedure
```

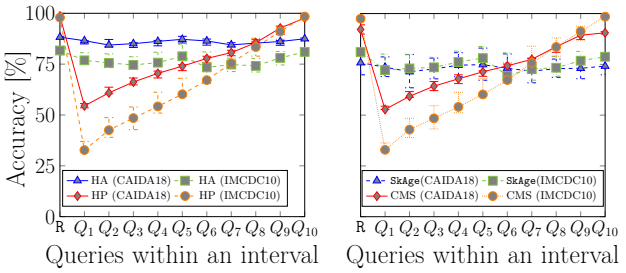
HashAge (HA). We adapt HashPipe (HP, [9]) that, in a nutshell, works as follows. The incoming packet's flow ID is looked up in the corresponding hash table and it is placed with count 1 in a slot if it was *empty*. In case of a *hit*, its counter is incremented, while in case of a *miss*, according to the actual stage there are two eviction policies. In the first stage the packet's flow ID and counter will be placed into the hashed slot *unconditionally*, while the resident entry is evicted and carried along to the next stage. In latter stages, however, the carried flow will only evict a resident flow if its corresponding count is greater. After the last stage, the evicted flow ID is "smoked out" completely.

In HashAge we keep the same policy as HP has in its first stage; but for any insertion at a latter stage, we check to what extent the last update time of the carried flow and the colliding resident flow are dropping behind Ω, and *decay* the counts appropriately before compare or update (see Alg. 1). In particular, we right shift the counts of flow F and T with the difference $(\Omega - F.\Omega)$ and $(\Omega - F.\Omega)$, respectively (Line 2–3). After the decay, everything works exactly the same as in HP; a flow ID will be either inserted into an empty slot, the decayed counts will be summed up in case of a *hit*, or the flow ID with the greater decayed count evicts the other one in case of a collision (Line 4–6), whereas the whole process starts over until the last stage.

SkAge. Here, we adapt Count-Min Sketch (CMS, [7]). In a CMS, whenever a packet arrives, its flow ID is hashed with e different hash functions, looked up in e rows then the corresponding counters will be incremented by one. The minimum of the e values is taken as the flow size. Correspondingly, within a given memory, sketches only have false positives (as once a flow is seen, its count will be stored), while hash tables only have false negatives (due to the eviction policy when collision occurs).

To adapt the exponential decay in CMS, we maintain the last observation phases alongside the counts as register pairs. Accordingly, each time we update (i.e., increment) the counts of a flow F (Line 4), we first decay its previous counts if $\Omega > F.\Omega$ (Line 3). In contrast to CMS, SkAge chooses the count that has the smallest observation phase Ω, as the practical count instead of the minimum of all counts as it is the closest to the actual last update for that particular flow.

Observe that in both algorithms, we dynamically perform time-decay for each individual count. This means that a heavy flow will be retained as long as its count is sufficiently large and new, at the price of introducing a slight estimation error in the true counts due to decaying. Otherwise, the flow will eventually disappear when there was no or negligible number of packets for a long time.



(a) Accuracy of hash tables. **(b) Accuracy of sketches.**

Figure 1: The impact of QAT on the accuracy within an interval (FI) in the hash table *(a)* and sketch-based *(b)* algorithms. 10 queries arrive at every second after the reset (R), each ask for the top-300 HH while the length of the interval-reset algorithm's intervals, FI is configured to 10 seconds.

3 PRELIMINARY EVALUATIONS

Here, we evaluate the impact of the query arrival times (QAT) on the detection accuracy by employing two different traces. We consider 10 queries arriving at every second after the reset (R), each asks for the top-300 HH within the last 10 seconds. We repeat this 10-second measurement through the chunks of [6] and [3] (also used in [9]) and the average results (with deviations) are depicted in Fig. 1.

Depending on whether the queries arrive following the ideal (i.e., before the reset) or worst cases (i.e., after the reset), the accuracy of the interval-reset algorithms fluctuates significantly; the accuracy drop in the worst case (for query Q_1) is as high as $40 - 45\%$. In comparison, our time-decaying HashAge (SkAge) shows a steady performance of ~85% (~75)% irrespective to when a particular query arrives (for all 10 queries).

This shows that when an underlying interval-reset algorithm is well aligned with the requirements of the sole application, i.e., QAT is at the end of each interval (FI), then it provides the most accurate results. However, when queries arrive at arbitrary times, performance drops are noticeable while the TDAs maintain consistent accuracy over time as compared to their interval-reset counterparts.

ACKNOWLEDGEMENT

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

REFERENCES

[1] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2016. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 1–9.

[2] Ran Ben-Basat, Gil Einziger, Isaac Keslassy, Ariel Orda, Shay Vargaftik, and Erez Waisbard. 2018. Memento: making sliding windows efficient for heavy hitters.. In *CoNEXT*, Xenofontas A. Dimitropoulos, Alberto Dainotti, Laurent Vanbever, and Theophilus Benson (Eds.). ACM, 254–266. http://dblp.uni-trier.de/db/conf/conext/conext2018.html#Ben-BasatEKOVW18

[3] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. 267–280. https://doi.org/10.1145/1879141.1879175

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[5] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. 99–110. https://doi.org/10.1145/2486001.2486011

[6] CAIDA. [n.d.]. The CAIDA UCSD Anonymized Internet Traces - 2018 March. http://www.caida.org/data/passive/passive_dataset.xml [Accessed: Oct 2019].

[7] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *J. Algorithms* 55, 1 (April 2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[8] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. 2009. Forward Decay: A Practical Time Decay Model for Streaming Systems. In *2009 IEEE 25th International Conference on Data Engineering*. 138–149. https://doi.org/10.1109/ICDE.2009.65

[9] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. 164–176. https://doi.org/10.1145/3050220.3063772

[10] The P4 Architecture Working Group. [n.d.]. Portable Switch Architecture. https://p4.org/p4-spec/docs/PSA-v1.0.0.pdf [Accessed: Jun 2020].

[11] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*. 29–42. http://dl.acm.org/citation.cfm?id=2482626.2482631