Paper

- Ran Ben-Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, Erez Waisbard. “Constant time updates in hierarchical heavy hitters”. SIGCOMM 2017

Abstract

In our project, we present and explain our attempt at reproducing some of the results from the paper “Constant time updates in hierarchical heavy hitters” by Ran Ben-Basat, Gil Einziger et al. for the course “Introduction to Networks” at Ben Gurion University. We ran a few experiments on parsed packet traces using RHHH and 10-RHHH algorithms that were presented in the paper and tried to reproduce the paper’s performance and error metrics on our own system. We managed to reproduce 4 figures from the paper, and we believe that they represent correctly the results that were achieved in the original paper.

Introduction

Heavy hitters (HH) are devices responsible for a large portion of the network traffic. Detection of HH may help with finding anomalies in the network and prevent DoS attacks. However, distributed DoS attacks (DDoS) cannot be prevented only by blocking particular devices, as in such attacks each device generates only a small portion of the traffic and is not considered HH. In DDoS, a group of malicious (or compromised) devices generates data traffic required for the attack. To detect this group, *hierarchical heavy hitters* (HHH) measurement is used.

HHH represents HH which is not necessarily identified by a single IP but can belong to some network cluster associated with a particular IP prefix. The main idea of HHH-based algorithms is to manage aggregated traffic data per IP prefix, so if some of the prefixes are responsible for heavy traffic they can be easily labeled as HH. The aggregated data can be calculated on one or more dimensions. For example, only on source IP, or both source and destination IP, etc.

Previous works on the HHH problem either rely on custom hardware or introduce a significant overhead because of the counters updates cost which is proportional to the hierarchy’s size (this overhead is expected to worsen even more with the transition to IPv6). The focus of the paper we review here is on improving the performance of software-based approaches without sacrificing accuracy. The main idea of the presented algorithm (RHHH) is to apply a probabilistic approach to the state-of-the-art HHH software-based solutions.

RHHH utilizes randomness in two orthogonal levels:

1. Instead of updating counters of each level per packet, only a single (random) level is updated. This way, “update per packet”’s cost improves from O(H) to O(1), where H is the size of the hierarchy.

2. Instead of updating counters of each packet, only counters of a (random) subset of packets (e.g. 10%) are updated. Obviously, skipping a large part of the traffic results in a much better performance.

Paper's evaluation section reports a speedup of up to x62 compared to the previous work (when both techniques above are used), while delivering similar accuracy.

The only drawback of the proposed approach, compared to the previous work, is that it cannot be applied immediately to the current traffic and some minimal number of packets is required. However, in real-world systems, that should not be an issue, as the required number of packets can be achieved within several seconds.

Methodology

We ran all the experiments on our own local machine, with 1 CPU and 16 GB of RAM, running Ubuntu 20.04. The original experiments tested packet traces up to one billion (2^{30}) packets long, but we limited our experiments to traces up to 32 million (2^{25}) packets due to time constraints (and because the results in the original paper were clearly visible already in packet traces of that size). Like the original paper, we used $\varepsilon = 0.001$ and $\theta = 0.01$ for our experiments. We used data collected from four CAIDA packet traces (San Jose 2013, San Jose 2014, Chicago 2015 and Chicago 2016). For each packet trace, we ran each experiment five times (as the original authors did), and then computed 95% confidence intervals using a standard student-*t* test (which appear in our figures as error bars). The steps in our experiment are as follows:

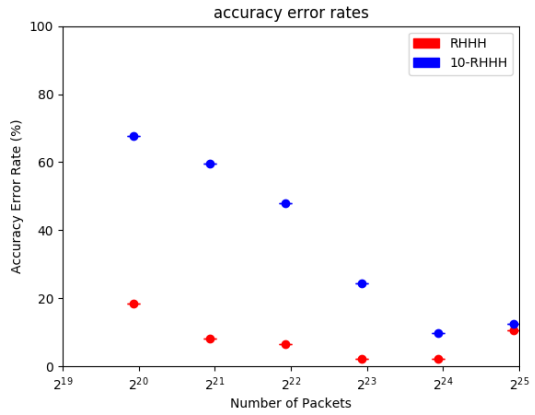
- Request access for the datasets and then download the packet traces from CAIDA and parse them into IP source, destination pairs.
- Run the selected algorithm for varying stream lengths ranging from 1 million to 32 million pairs.
- Repeat for a total of 5 trials for each algorithm.
- Run a checker program that compares our reported HHH with the actual ones and reports various error metrics.
- Run a python script to compute 95% confidence intervals and plot the error rates for each trace.

We used the original source code left behind by Ran Ben-Basat on his public GitHub repository. The repository has well documented instructions on how to run the algorithms on the parsed packet traces and measure both time and HHH precision metrics so we could produce the data for the visual figures. Furthermore, Tommy Fan and Austin Poore from Stanford University did a great job at re-creating the scripts for the figures reproduction which made our job at reproducing the paper's results by ourselves much easier. We used the same datasets from CAIDA (Center for Applied Internet Data Analysis) as used in the original paper, but probably not the same '.pcap' files (as the researchers did not specify which ones they used) so the traces are not necessarily the same but they do come from the same source.

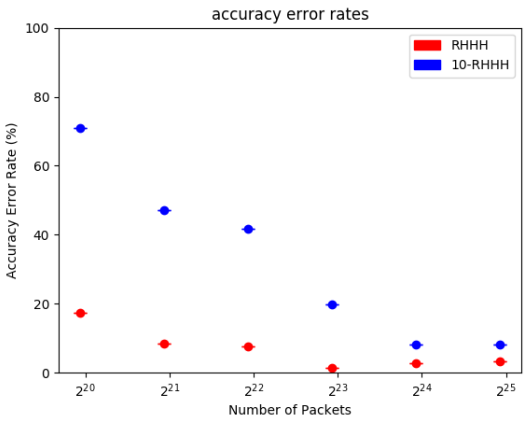
Results

Figure 2 in the paper plots the accuracy error rate against the stream length (number of packets). It compares the regular RHHH algorithm with the 10-RHHH algorithm. As expected, 10-RHHH has higher errors, but the plot shows that it quickly decreases as the stream size grows. We followed the researcher's criteria for an accuracy error where for a prefix P, if $|f_p - \widehat{f_p}| > 0.001N$we say that it is inaccurate. The error rate is calculated by the number of accuracy errors divided by the number of HHH that the algorithm found.

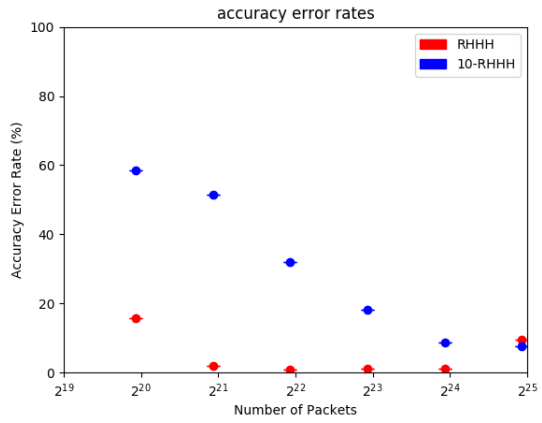
Our Figure 2 Accuracy Error Ratio:



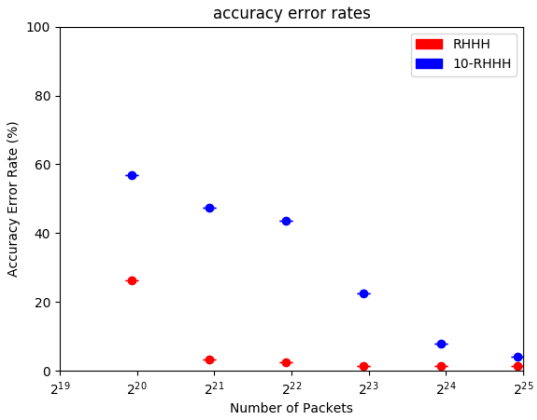
(a) Chicago 2015 – 2D Bytes



(b) Chicago 2016 – 2D Bytes



(c) San Jose 2013 – 2D Bytes

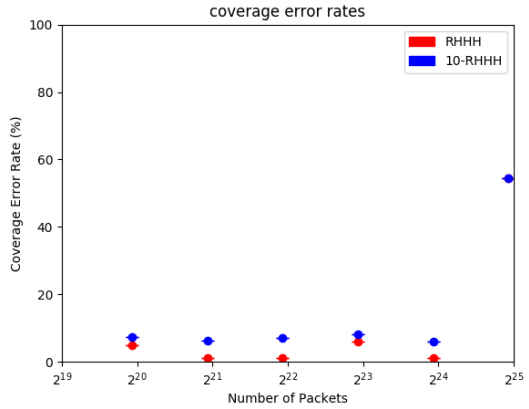


(d) San Jose 2014 – 2D Bytes

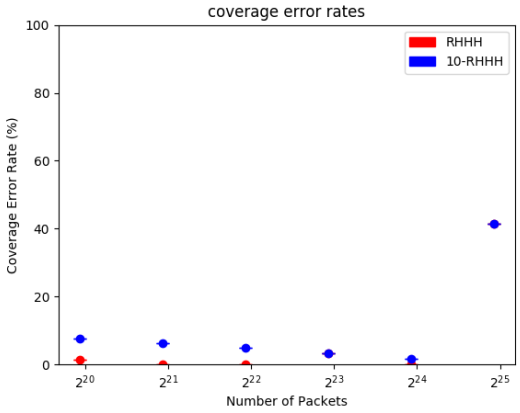
When we compare our figures with the figures from the paper's experiments, we can see that although the exact numbers may vary, both match the same trend and overall results. The accuracy error rate for 10-RHHH is drastically higher than that of RHHH, but for both algorithms, the accuracy error rate sharply decreases as we increase the number of packets. One notable difference is the exact value of the accuracy error rate for 10-RHHH, specifically where the number of packets is low. In our graphs, the value of accuracy error rate for 10-RHH can be a bit higher than the one in the paper but given the variability of its randomized algorithm it is expected.

Like Figure 2, Figure 3 of the paper plots another error metric, the coverage error rate, against the number of packets. This experiment reports about the false negatives, meaning the HHH that are missing from the algorithm's output. As in Figure 2, we used the same parameters as the researchers where $q \notin P$such that $C_{q \mid p} \geq N\theta$.

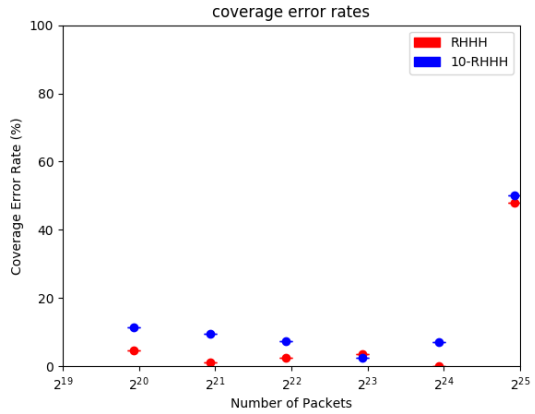
Our Figure 3 Coverage Error Ratio:



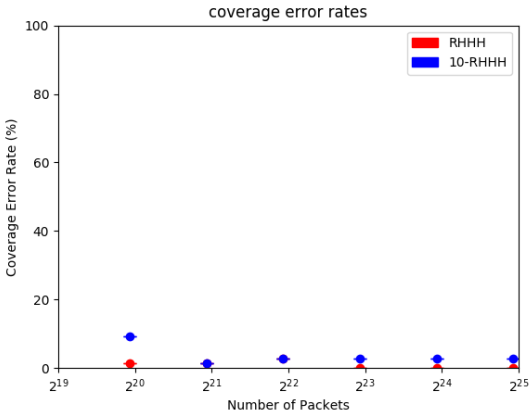
(a) Chicago 2015 – 2D Bytes



(b) Chicago 2016 – 2D Bytes



(c) San Jose 2013 – 2D Bytes



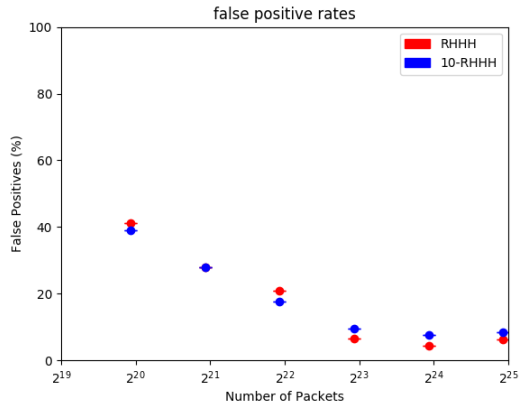
(d) San Jose 2014 – 2D Bytes

As the last figure comparison, we got similar results as the researchers got in the paper, our graphs demonstrate a slightly downward trend in the coverage error rate as the number of packets increases. While reviewing the figures, both ours and paper's, we noticed that the false negative rate is quite low for both algorithms, even at low stream length. This insight suggests that perhaps the algorithms over-estimates counts, which seems correct after reviewing Figure 4.

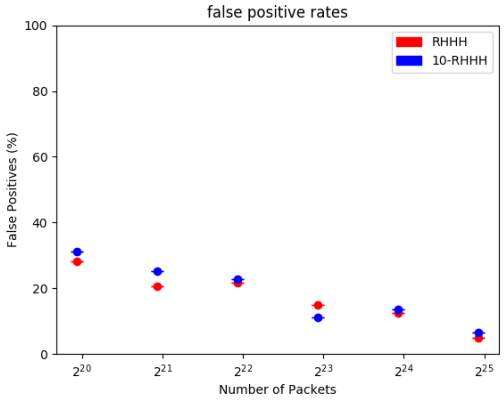
Figure 4 presents a third error metric, the false positive rate, against the number of packets in the stream. Those false positives are the non-HHH prefixes which were mistakenly classified as HHH by the algorithm.

Project by Amit Binenfed, Chen Mordechai and Grisha Weintraub

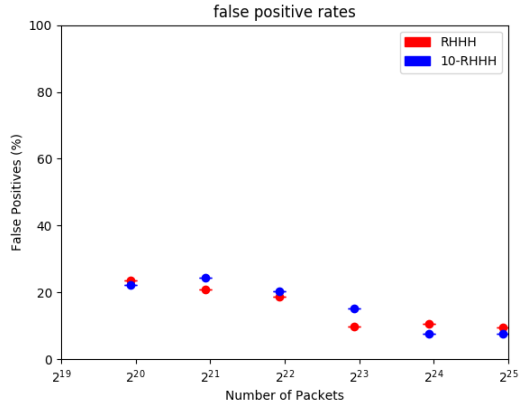
Our Figure 4 False Positive Ratio:



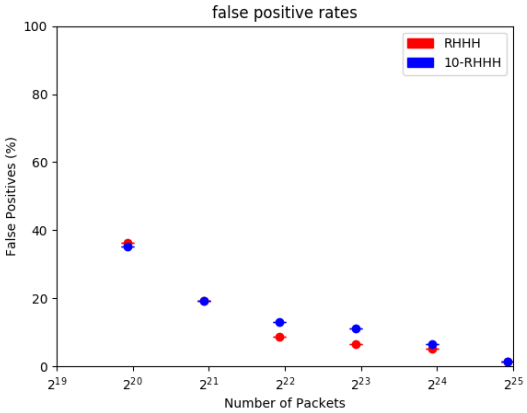
(a) Chicago 2015 – 2D Bytes



(b) Chicago 2016 – 2D Bytes



(c) San Jose 2013 – 2D Bytes



(d) San Jose 2014 – 2D Bytes

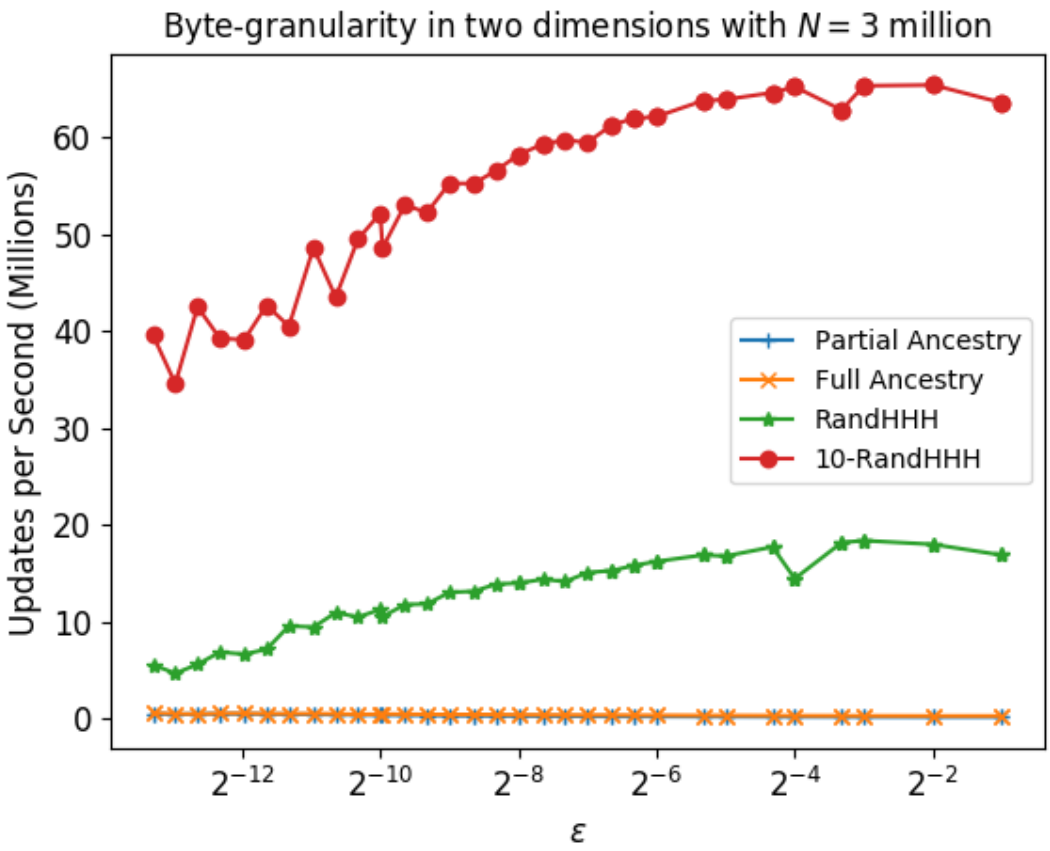
As in Figures 2 and 3, we can see that the false positive ratio is reduced as the packet trace gets bigger. However, when we compare the current graphs with the last two figures, we found that unlike the last two, this time 10-RHHH shows similar and sometimes even better accuracy rates against RHHH even at the early stages of the experiment. Overall, same as before, we see the same downward trend at the figures, as the original figures in the paper.

Figure 5 experiment does an empirical analysis of the runtime performance. This test is a bit different than previous ones mainly because this experiment is machine-dependent, which could end up being important when comparing the results. Since the main goal of the HHH algorithms is that they're able to keep up with the network, it makes sense that the performance of the update step is the one that we need to measure.

We run the experiment on 3 million packets instead of 250 million as in the original paper due to lack of resources and time and got quite similar results. We can see that even with much fewer packets the performance of each algorithm is quite similar and with the same trend in comparison to the one in the paper.

Project by Amit Binenfed, Chen Mordechai and Grisha Weintraub

Our Figure 5 Update Speed Comparison:



Chicago 2016 – 2D Bytes

Discussion

Overall, we consider this attempt to be a successful one at reproducing the paper's results. We were able to obtain similar numbers to the original paper when running the code on our system and all figures we generated ended up looking very alike to their counterparts. We do take into consideration that the algorithm is randomized, so of course the numbers will not be the same, but the overall quality is similar. Furthermore, the fact that we used the same source code as the researchers and used datasets from the same source (although not necessarily the same ones) and got similar results validates the paper's work.
However, we would have liked to test the algorithms on new packet traces and check if the same performance can be obtained on real systems in order to confirm the reproducibility of the paper's results.