Volumetric Hierarchical Heavy Hitters

Ran Ben Basat
Computer Science
Technion
sran@cs.technion.ac.il

Gil Einziger
Nokia Bell Labs
gil.einziger@nokia.com

Roy Friedman
Computer Science
Technion
roy@cs.technion.ac.il

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

Erez Waisbard
Nokia Bell Labs
erez.waisbard@nokia.com

***Abstract*—*Hierarchical heavy hitters (HHH)* identification is useful for various network utilities such as anomaly detection, DDoS mitigation, and traffic analysis. However, the increasing support for jumbo frames enables attackers to overload the system with fewer packets, avoiding detection by packet counting techniques.**

This paper suggests an efficient algorithm for detecting HHH based on their traffic volume that asymptotically improves the runtime of previous works. We implement our algorithm in Open vSwitch (OVS) and incur a $4-6\%$ overhead compared to a 42% throughput reduction experienced by the state-of-the-art.

I. Introduction

Network functionalities such as traffic engineering, load balancing, quality of service, caching, anomaly and intrusion detection often require timely measurements from the underlying networks [1]–[8]. In practice, supporting online measurements is a difficult challenge due to the rapid line rates and a large number of active flows.

Instead of monitoring all flows, previous works suggested identifying *Heavy Hitter* (HH) flows [9] that transmit a large portion of the packets. Approximate HH can be measured efficiently in practice [10]–[14]. Alas, in a *Distributed Denial of Service (DDoS)* attack, each device only transmits a moderate amount of traffic and is therefore not a heavy hitter. Yet, the attacker uses a large number of devices and their combined traffic can overwhelm the service. Therefore, other methods are needed to identify the origins of such attacks [15], [16].

The attacking devices of a DDoS attack are often located within a small number of subnetworks and thus have common IP prefixes. *Hierarchical Heavy Hitters (HHH)* [17], [19], [20] is a measurement technique that is designed to identify such attacks. Specifically, the technique is capable of observing that specific subnets are responsible for a large portion of the traffic even if no individual flow within these subnets is a heavy hitter by itself.

Hence, HHH identification is a powerful tool that is used to mitigate DDoS attacks [21], [22]. Intuitively, one can monitor the system and record which subnets usually deliver a significant portion of the traffic. Then, we can identify new and unexpected hierarchical heavy hitters as potential sources of a DDoS attack. Such an identification can then be used to block the attack traffic with minimal impact on legitimate traffic.

Measurement is often performed on a per-packet base. However, the rise of technologies such as jumbo frames motivates the need for per byte rather than per packet measurements [11], [13], [23].

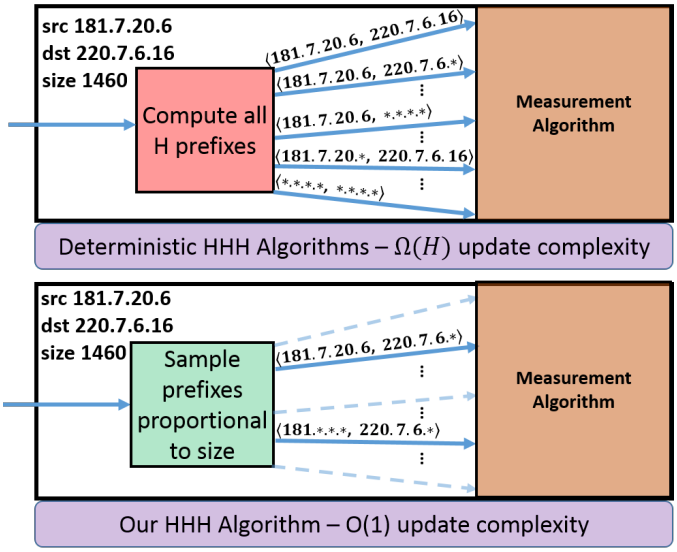


Fig. 1: A high-level overview of this work compared to prior art [17]. In [17], each HHH update is broken into H plain heavy-hitter updates, each can be performed in constant time [13], [18]. Our algorithm applies a sampling rate of β bytes (e.g., 1KB) per heavy-hitter update. Specifically, the number of bytes before the next sample is geometrically distributed with a mean of β. When a packet is sampled, a *single* prefix is computed. Thus, we may compute several prefixes for large packets, while smaller ones may be ignored altogether. By choosing β proportionally to the average packet size, we allow constant time updates while our analysis guarantees HHH identification with high probability.

Network devices increasingly support large packet payloads exposes a new attack vector that cannot be identified with per packet HHH measurement. Specifically, a rouge subnet can deliver an overwhelming volume of traffic using a relatively small number of large packets. Thus, its overall packet count is below the detection threshold and the subnet is not detected as a hierarchical heavy hitter. Yet, the combined volume of its traffic can exhaust the bandwidth network devices, increase packet drops and overwhelm the receiving network.

Thus, in order to identify such attacks, it is essential to efficiently measure *volumetric* hierarchical heavy hitters. That

is, to identify the subnetworks that deliver large portions of the overall bandwidth. Further, as volumetric attacks target network devices, detection capabilities are required at the network device level. The work of [24] showed capabilities to perform HHH detection at the Line's speed in network devices. Alas, the method utilizes random packet sampling which is insufficient to perform volumetric measurements. Intuitively, the reason for that is that large packets are sampled with the same probability as small packets.

A. Our contribution

In this introduces VRHHH, a randomized algorithm for identifying volumetric HHH at the line's speed. We motivate this problem by showing that on real Internet traces the volumetric HHH prefixes differ from the per-packet HHH prefixes by as much as 50%.

VRHHH samples random bytes in the traffic and the number of bytes sampled from each packet is proportional to its size. Moreover, for each sampled byte we only update a random prefix from the corresponding packet. Therefore, the total number of sampled prefixes is proportional to the packet size. Intuitively, This method allows us to cope with the line speed and still identify the volumetric HHH prefixes. VRHHH asymptotically improves the update time of previous volumetric HHH algorithms achieving $O(1)$ runtime.

We formally analyze VRHHH and show accuracy guarantees. That is, we show that once we process enough traffic then the HHH prefixes are detected with high probability. We also evaluate VRHHH on four real Internet packet traces and show that it converges after a reasonable amount of traffic. After convergence, it provides similar accuracy to previously suggested deterministic algorithms.

Next, we demonstrate a capability to perform volumetric HHH measurement in real network devices. We also integrate VRHHH into the data plane of a real network switch (Open vSwitch) and demonstrate HHH measurements with only a minor 4-6% decrease in the maximal throughput compared to the unmodified DPDK enabled OVS. In contrast, the fastest deterministic alternative reduces the throughput by as much as 42%.

II. Related Work

Heavy Hitters [14], [25], [26] are about identifying flows that transmit many packets [14], [25] or a large byte volume [11], [13]. In a DDoS attack, each attacking device transmits a moderate amount of traffic, but the combined volume is overwhelming. Thus, DDoS attackers avoid detection as heavy hitters since each individual device is not a heavy hitter. This motivated the definition of the hierarchical heavy-hitters (HHH) problem. HHH were first defined for a single dimension by [27]. Such a definition is used to identify the origin of an attack. The work of [27] also introduced the first algorithm for approximate HHH. Shortly after, [19] suggested a hardware HHH algorithm for one dimension based on TCAM memory. The HHH problem was then extended to multiple dimensions [15], [17], [20], [24], [28], [29]. This extension enables backbone routers to detect both the origin and the destination of a DDoS attack. Further, it can be extended to include additional information such as port numbers and protocol in the measurement and hence refine the decision process.

The work of [20] solves the problem with a trie-based approach. It suggests two algorithms: Full Ancestry that is optimized for accuracy and Partial Ancestry that favors compactness. Asymptotically, both are the same and require $O\left(\frac{H\log(N\epsilon)}{\epsilon}\right)$ space and $O\left(H\log(N\epsilon)\right)$ update time.

A simpler algorithm for the problem was proposed by [17], who solve the HHH problem with multiple instances of the Space Saving algorithm (SS) [25] for the plain heavy hitter problem. Their solution uses a different SS instance for each prefix mask. Upon packet arrival, every SS instance is updated with the corresponding prefix. That is, they compute all possible prefixes and update multiple corresponding SS instances. The algorithm is illustrated in the upper part of Figure 1. This simple approach outperforms previous sophisticated algorithms.

This algorithm, nicknamed MST, provides similar accuracy guarantees to previous works, while improving the space and update time of [20]. Specifically, MST requires $O\left(\frac{H}{\epsilon}\right)$ space and its update time for unitary inputs is $O\left(H\right)$. For byte counting, it requires weighted inputs and requires $O\left(H\log\frac{1}{\epsilon}\right)$ update time. Recently, [11], [13], [23], [30] proposed weighted (plain) heavy hitters algorithms that operate in constant time. By replacing the SS instance in MST with these algorithms the update time of MST can be reduced to $O(H)$.

For unitary inputs, the work of [24] suggests *RHHH*, a randomized algorithm with a constant update complexity. The algorithm uses the same data structure as MST, but only updates a single heavy hitter instance per packet arrival. The intuition behind RHHH is that when the number of packets is very large, this randomized method converges and achieves comparable accuracy to deterministic algorithms.

In our context, RHHH uses uniform packet sampling which is inferior to byte samples for estimating the byte volume [11]. Formally, the accuracy of uniform samples depends on the packet size variance [31] which is an uncontrolled workload dependent variable. Intuitively, large and small packets have the same probability to be sampled which creates inaccuracies. Particularly, bytes in small packets are sampled with a high probability and bytes in large packets are sampled with a low probability.

The recent work of [11] suggests a byte uniform sampling method to resolves these issues. Their method *BUS* samples packets with a probability proportional to their byte size. This method yields convergence regardless of the size variance of the packets. If the maximal packet size is M, BUS supports sampling with probabilities of at most $\frac{1}{M}$. This is a significant limitation in the context of our work since the HHH problem often requires a larger sampling probability for timely convergence. In contrast, the method suggested here supports any sampling probability.

III. HHH Basic Terminology

We consider IP prefixes as a hierarchical relation. The *Fully general* prefix is denoted $*$ and generalizes all other prefixes. Similarly, a *Fully specified* IP prefix is the lowest level of the hierarchy and does not generalize any other prefixes. A fully specified prefix is also assigned a weight; the weight of prefix e is W_e.

A prefix p_1 generalizes another prefix p_2 if p_1 is a prefix of p_2. For example, $181.7.20.6$ is a fully specified prefix, and $181.7.20.*$ generalizes it. Similarly, $181.7.*$ generalizes $181.7.20.6$ as well as $181.7.20.*$. The *parent* of a prefix is the longest prefix that generalizes it; e.g., the parent of $181.7.20.6$ is $181.7.20.*$.

In two dimensions, we consider a tuple containing source and destination IP prefixes. A fully specified tuple is fully specified in both dimensions. For example, $(\langle 181.7.20.6\rangle \rightarrow \langle 208.67.222.222\rangle)$ is a fully specified tuple. In this case, each tuple has two parents, e.g., $(\langle 181.7.20.*\rangle \rightarrow \langle 208.67.222.222\rangle)$ and $(\langle 181.7.20.6\rangle \rightarrow \langle 208.67.222.*\rangle)$ are the parents of $(\langle 181.7.20.6\rangle \rightarrow \langle 208.67.222.222\rangle)$. Definition III.1 formally captures this concept.

Definition III.1 (Generalization)**.** *For two prefixes p, q, we denote $p \preceq q$ if in any dimension it is either a prefix of q or is equal to q. We also denote the set of elements that are generalized by p with $H_p \triangleq \{e \in \mathcal{U} \mid e \preceq p\}$, and those generalized by a set of prefixes P by $H_P \triangleq \cup_{p\in P} H_p$. If $p \preceq q$ and $p \neq q$, we denote $p \prec q$.*

In a single dimension, the generalization relation defines a vector going from fully generalized to fully specified. In two dimensions, the relation defines a lattice where each item has two parents. The notation H denotes the size of this relation. For example, in IPv4, a byte level one dimensional hierarchy means $H = 5$ as each IP address is divided into four bytes, and we also allow querying $*$. Similarly, a two-dimensional hierarchy implies $H = 25$ as illustrated in Table I.

Definition III.2. *The* volume *of a prefix p is:*

$$f_p \triangleq \sum\nolimits_{e\in H_p} w_e.$$

Next, we define a heavy hitter prefix.

Definition III.3 (Heavy hitter (HH))**.** *Given a threshold (θ), a prefix (p) is a **heavy hitter** if its volume (f_p) is above the threshold of a θ total volume B, i.e., $f_p \geq \theta \cdot B$.*

Our goal is to identify the hierarchical heavy hitter prefixes whose volume is above the threshold $(\theta \cdot B)$. However, simply returning the set of all prefixes that exceed the threshold has plenty of redundancy and double counting. Thus, there is a need to compress the HHH set. Intuitively, given an existent set, we ask how much additional traffic is added to the set by adding another prefix. This concept is called *conditioned frequency* and is denoted $(C_{p|P})$. Intuitively, if the frequency of a prefix exceeds the threshold then so is that of all its ancestors. For compactness, we are interested in prefixes whose frequency is above the threshold due to non-HHH siblings. This motivates the need for the conditioned frequency that measures the **additional** traffic a prefix (p) adds to a set of previously selected HHH prefixes (P), and it is defined as follows:

Definition III.4. *(Conditioned frequency) The conditioned frequency of a prefix p with respect to a prefix set P is:*

$$C_{p|P} \triangleq \sum_{e\in H_{(P\cup\{p\})}\setminus H_P} f_e.$$

Intuitively, the conditioned frequency of prefix p w.r.t. a prefix set P is the number of packets that are generalized by $P \cup \{p\}$ but are not generalized by$P$. Thus, conditioned frequency measures how much a prefix contributes to a set.

A. How to calculate (exact) HHH?

The work of [17] explains how exact conditioned frequency is calculated in one and two dimensions. Intuitively, in one dimension we first add to the HHH set all the fully specified prefixes above the threshold (θB). This process is recursively repeated with their parents, evaluating the conditioned frequency of each parent prefix with the already selected prefixes.

In two dimensions, each prefix has two parents so we need to use inclusion and exclusion principles to avoid overcounting. A set of prefixes solves the HHH problem if it satisfies the following definition.

Definition III.5 ($(\delta, \epsilon, \theta)-$Volumetric HHHs)**.** *An algorithm $\mathbb{A}$ solves $(\delta, \epsilon, \theta)$ -* Approximate Volumetric Hierarchical Heavy Hitters *if after processing any stream $\mathbb{S}$ of volume B, it returns a set of prefixes P that, for an arbitrary run of the algorithm, satisfies the following:*

- ***Accuracy:** for every prefix $p \in P$,*

$$\Pr\left(\left|f_p - \widehat{f}_p\right| \leq \varepsilon B\right) \geq 1 - \delta.$$

- ***Coverage:** given a prefix $q \notin P$,* $\Pr\left(C_{q|P} < \theta B\right) \geq 1-\delta$.

That is, we require that each prefix within the HHH set has an accurate frequency estimation to satisfy the **accuracy** requirement. Alternatively, given a prefix not included in the set, with a probability of 1-δ its conditioned frequency with respect to the set is below the threshold.

We also need the following definition for HHH calculation.

Definition III.6. *Given a prefix p and a set of prefixes P, we define $G(p|P)$ as the set of prefixes:*

$$\{h : h \in P, h \prec p, \nexists\, h' \in P \ s.t.\ h \prec h' \prec p\}.$$

Intuitively, $G(p|P)$ are the prefixes in P that are closest to p according to the generalize relation. E.g., let $p =< 142.14.* >$ and the set $P = \{< 142.14.13.* >, < 142.14.13.14 >\}$, then $G(p|P)$ only contains $< 142.14.13.* >$. For brevity, notations used in this work are summarized in Table II.

Src/Dest	*	d1.*	d1.d2.*	d1.d2.d3.*	d1.d2.d3.d4
*	(*,*)	(*,d1.*)	(*,d1.d2.*)	(*,d1.d2.d3.*)	(*,d1.d2.d3.d4)
s1.*	(s1.*,*)	(s1.*,d1.*)	(s1.*,d1.d2.*)	(s1.*,d1.d2.d3.*)	(s1.*,d1.d2.d3.d4)
s1.s2.*	(s1.s2.*,*)	(s1.s2.*,d1.*)	(s1.s2.*,d1.d2.*)	(s1.s2.*,d1.d2.d3.*)	(s1.s2.*,d1.d2.d3.d4)
s1.s2.s3.*	(s1.s2.s3.*,*)	(s1.s2.s3.*,d1.*)	(s1.s2.s3.*,d1.d2.*)	(s1.s2.s3.*,d1.d2.d3.*)	(s1.s2.s3.*,d1.d2.d3.d4)
s1.s2.s3.s4	(s1.s2.s3.s4,*)	(s1.s2.s3.s4,d1.*)	(s1.s2.s3.s4,d1.d2.*)	(s1.s2.s3.s4,d1.d2.d3.*)	(s1.s2.s3.s4,d1.d2.d3.d4)

TABLE I: The lattice induced by the two dimensional source/destination hierarchy. The two parents of each node are directly above it and directly to the left. The top left corner (*,*) is fully general while the bottom right corner (s1.s2,s3.s4,d1.d2.d3.d4) is fully specified.

Symbol	Meaning
$\mathbb{S}$	Stream
N	Current number of packets (in all flows)
B	Current byte volume (in all flows)
H	Size of Hierarchy
$\mathcal{U}$	Domain of fully specified items.
$\epsilon, \epsilon_s, \epsilon_a$	Overall, sample, algorithm's error guarantee.
$\delta, \delta_s, \delta_a$	Overall, sample, algorithm confidence.
θ	Threshold parameter.
H_q	Set of packets generalized by prefix q.
$C_{q\|P}$	Conditioned frequency of q with respect to P
$G(q\|P)$	Subset of P with the closest prefixes to q.
f_q	volume of prefix q
$\widehat{f_q^+}, \widehat{f_q^-}$	Upper,lower bound for f_q

TABLE II: List of Symbols

IV. Volumetric Randomized HHH Algorithm

We now present *Volumetric Randomized HHH (VRHHH)* that shares a similar structure to MST [17] and to RHHH [24]. We use multiple instances of heavy hitter algorithms each for counting the frequencies of a certain prefix pattern (subnets). VRHHH is composed of two methods, an update method that digests a packet and a query method that list the HHH prefixes. The novelty of VRHHH is its randomized constant time updates which are exemplified in Figure 1. The update procedure directly samples bytes in the traffic and for each sampled byte, it randomly selects a prefix from the corresponding packet's headers. Next, it updates the corresponding heavy hitter instance in a similar manner to RHHH. This means that small packets may be ignored, while large packets may be sampled multiple times. Our sampling method is more flexible than BUS [11] as it supports a wider range of sampling probabilities, and does not require knowing the maximal packet size in advance.

Our implementation uses geometric variables where: $X \sim Geometric(\beta^{-1})$ is a random variable with a mean of β, counting the number of bytes prior to the next prefix sample. This method is manifested in the Update procedure of Algorithm 1. The parameter d counts how many bytes to skip until the next sampled byte. In Line 2, we reduce d by the current packet weight and if $d \leq 0$ then a byte in this packet is sampled. Thus, we randomize how many bytes should be skipped before the next byte is sampled (Line 7) and add this value to d. We continue the process as long as $d \leq 0$ which means that some packets are sampled multiple times. For each sampled packet we randomize a prefix (Line 4, Line 5), and then increment the (plain) heavy hitter instance corresponding to that prefix (Line 6). Our implementation uses Space Saving [25] but can in practice utilize any plain heavy hitter algorithm.

The query method lists the HHH prefixes, by conservatively estimating their residual frequency. Intuitively, the HHH list describes the traffic in a compact manner as explained below.

A. Single Instance

We now explain how a single HH instance operates in VRHHH. Specifically, how it is modified to work with the sampling method. Intuitively, when sampling each byte with probability β^{-1}, we expect $\beta^{-1}B$ bytes to be sampled.

Since each HH instance counts the number of sampled bytes, we multiply its estimation by β. Further, since there are H different instances and each sampled byte only updates one of these instances, we also multiply the frequency estimation by H to compensate. Thus, in total the frequency estimation of the algorithm is multiplied by βH, and the overall sampling probability is $\beta^{-1}H^{-1}$. We denote by $\widehat{x}$ the estimated frequency of a prefix x. SS also provides $\widehat{x}^+$ and $\widehat{x}^-$ as upper and lower bounds on the frequency of x.

Definition IV.1. *We define the following estimators:*

- *Unbiased estimator:* $\widehat{f}_x \triangleq \widehat{X}\beta H$.
- *Upper estimation:* $\widehat{f}_x^{\,+} \triangleq \widehat{X}^+\beta H$.
- *Lower estimation:* $\widehat{f}_x^{\,-} \triangleq \widehat{X}^-\beta H$.

B. HHH calculation

Intuitively, the HHH procedures start from fully specified items and gradually and recursively build the HHH set. Algorithm 1 depicts this process. The difference between one and two dimensions is manifested in the calcPred method. We use Algorithm 2 for one dimension and Algorithm 3 for two. Also, note Line 15 in Algorithm 1 that makes the necessary adjustments to the conditioned frequency calculation to account for the sampling error.

Algorithm 1 Randomized Volumetric HHH algorithm (VRHHH) — HH_Alg (ϵ_a^{-1}) is an instance of an arbitrary HH algorithm ensuring an error bound of ϵ_a^{-1}, used as a blackbox. $HH[e]$ holds an algorithm's instance for each level of the hierarchy e. It is associated with a corresponding network mask $HH[e].mask$. Invoking $HH[e].INCREMENT(p)$ updates subnet p in the corresponding algorithm's instance.

Init: $d \leftarrow Geo(\beta^{-1})$
$\forall e \in [H] : HH[e] =$ HH_Alg (ϵ_a^{-1})
function UPDATE(x, w)
  $d \leftarrow d - w$
  while $d \leq 0$ **do**
    $id = rand(0, H)$
    Prefix $p = x \& HH[id].mask$ ▷ Bitwise AND
    $HH[id].INCREMENT(p)$
    $d \leftarrow d + Geo(\beta^{-1})$
  end while
end function
function OUTPUT(θ)
  $P = \phi$
  for Level $l = |H|$ down to 0. **do**
    for each p in level l **do**
      $\widehat{C_{p|P}} = \widehat{f_p}^{+} + calcPred(p, P)$
      $\widehat{C_{p|P}} = \widehat{C_{p|P}} + 2Z_{1-\delta}\sqrt{NB}$
      if $\widehat{C_{p|P}} \geq \theta N$ **then**
        $P = P \cup \{p\}$ ▷ p is an HHH candidate
        $print\left(p, \widehat{f_p}^{-}, \widehat{f_p}^{+}\right)$
      end if
    end for
  end for
  return P
end function

Algorithm 2 calcPred for one dimension

function CALCPRED(prefix p, set P)
  $R = 0$
  for each $h \in G(p|P)$ **do**
    $R = R - \widehat{f_h}^{-}$
  end for
  return R
end function

Note that in two dimensions there are two ancestors for each prefix and thus we need the notion of glb defined below.

Definition IV.2. *Denote $glb(h, h')$ the greatest lower bound of h and h'. $glb(h, h')$ is a unique common descendant of h and h' s.t. $\forall p : (q \preceq p) \wedge (p \preceq h) \wedge (p \preceq h') \Rightarrow p = q$. When h and h' have no common descendants, define $glb(h, h')$ as an item with count 0.*

C. Correctness

The full analysis is given in Section VII. Here, we provide a highlight, of the correctness proof of VRHHH. Intuitively, we prove the correctness of the algorithm by a reduction to unweighted streams. Given the (weighted) stream $\mathcal{S}$, we consider the unweighted stream $\mathcal{S}_U$ generated by "spreading" each packet $\langle x, w\rangle$ (with identifier x and weight w) into w consecutive packets with identifier x. For example, the stream $\mathcal{S} = \langle x, 2\rangle \langle z, 1\rangle \langle y, 2\rangle \langle x, 1\rangle$ becomes $\mathcal{S}_U = xxzyyx$. Clearly, the byte volume of each prefix in $\mathcal{S}$ is identical to its frequency in $\mathcal{S}_U$. Finally, we observe that the operation of our algorithm on $\mathcal{S}$ is mathematically equivalent to running RHHH [24] on $\mathcal{S}_U$, for $V = H\beta$. By the correctness of RHHH we conclude that once the byte volume is large enough VRHHH solves the problem. Formally, we get the following:

Algorithm 3 calcPred for two dimensions

function CALCPRED(prefix p, set P)
  $R = 0$
  for each $h \in G(p|P)$ **do**
    $R = R - \widehat{f_h}^{-}$
  end for
  for each pair $h, h' \in G(p|P)$ **do**
    $q = glb(h, h')$
    if $\nexists h_3 \neq h, h' \in G(p|P), q \preceq h_3$ **then**
      $R = R + \widehat{f_q}^{+}$
    end if
  end for
  return R
end function

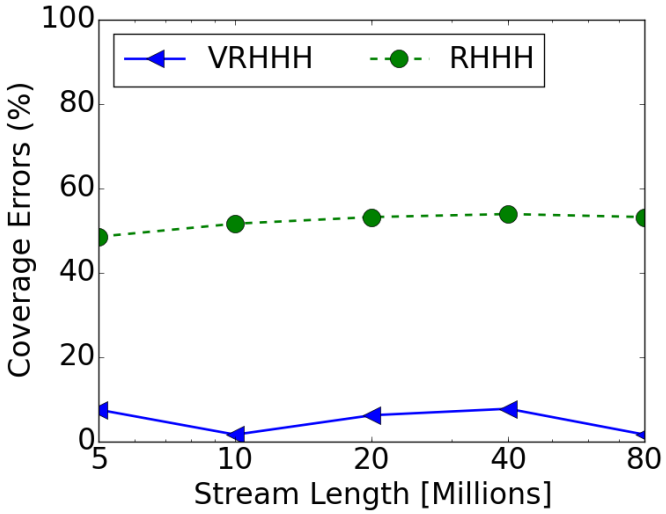


Fig. 2: Coverage errors of RHHH and VRHHH (Chicago 16)

Theorem IV.1. *If $B > \epsilon_s^{-2} Z_{1-\frac{\delta_s}{2}} H\beta$, then VRHHH solves the $(\delta, \epsilon, \theta)$ -* APPROXIMATE VOLUMETRIC HIERARCHICAL HEAVY HITTERS *problem.*

We emphasize that running RHHH on $\mathcal{S}_U$ is not a viable approach as the processing of $\langle x, w\rangle$ would take $O(w)$ time; this is rather an *emulation* of RHHH that allows constant time updates for weighted streams.

V. EVALUATION

We start the evaluation by exploring the differences between RHHH [24] that calculates per-packet HHH and VRHHH that performs the volumetric calculation. Next, we evaluate the properties of VRHHH including accuracy, coverage and false

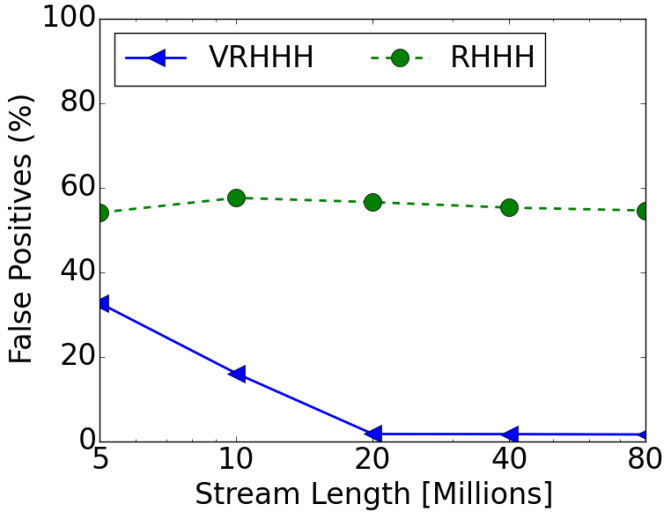


Fig. 3: False positives of RHHH and VRHHH (Chicago 16)

positives and explain how the sampling rate (β) and the accuracy ε affect the algorithm. We conclude our evaluation with an integration to OVS and show a throughput improvement compared to previous algorithms. This improvement is due to the faster runtime of VRHHH.

Our evaluation is based on four real 80 million packet long traces from two backbone links in Chicago and San Jose [32]. The traces were recorded in the years 2013-2016. We consider source/destination byte hierarchies (H $= 25$) that were also used by [17], [20], [24].

A. Volumetric HHH vs per-packet HHH

In a gist, the main benefit of VRHHH over prior art is its much faster runtime. Thus, it is important to compare VRHHH to RHHH [24] for the volumetric problem since RHHH has a similar runtime to VRHHH. In the following experiments, we use $\varepsilon = 0.1\%$ and report HHH prefixes with conditioned frequencies of above 1% of the total traffic.

Figure 2 shows the coverage property in both RHHH and VRHHH. In this measurement, RHHH performs a per-packet HHH but the quality of this measurement is compared with the volumetric definition. As can be observed, VRHHH converges and yields a small number of coverage errors. In contrast, RHHH does not improve with the stream and yields about 50% coverage errors. Figure 3 completes the picture by showing the number of false positives. That is, the number of prefixes that are wrongly included in the returned HHH set. Also, the number of false positives ratio improves over time in VRHHH but is constant at about 50-60% in RHHH. We note that RHHH was reported to converge for this dataset and parameters after 32M packets [24]. Thus, per packet HHH measurement is fundamentally different than volumetric HHH measurement.

In conclusion, we get that about 50% of the per-packet HHH prefixes are not volumetric HHH prefixes and vise verse. Thus, in order to cope with volumetric threats, we cannot settle for per-packet solutions and require volumetric algorithms.

B. Coverage

Recall Definition III.5, which defines the accuracy and coverage properties of an HHH set. Unlike deterministic algorithms, our VRHHH algorithm exhibits coverage and accuracy errors. Figure 4 evaluates the percentage of coverage errors as a function of the length of the trace. We measure multiple β values ranging from 200 to 4000. As can be observed, VRHHH has a few coverage errors and these become rare as the trace progresses. Note that after 80 million packets, there are hardly any coverage errors, especially for $\beta \leq 1000$.

C. Accuracy

Figure 5 shows results for the accuracy property. Note that deterministic algorithms do not have accuracy errors. As can be observed, the accuracy improves as the stream progresses. Here, some of the configurations have non-negligible accuracy errors after 80 million packets. Hence, we recommend the $\beta = 1000$ configuration as it seems to have very few errors after 80 million packets. We further note that longer traces would imply fewer errors, as indicated by our analysis.

D. False Positives

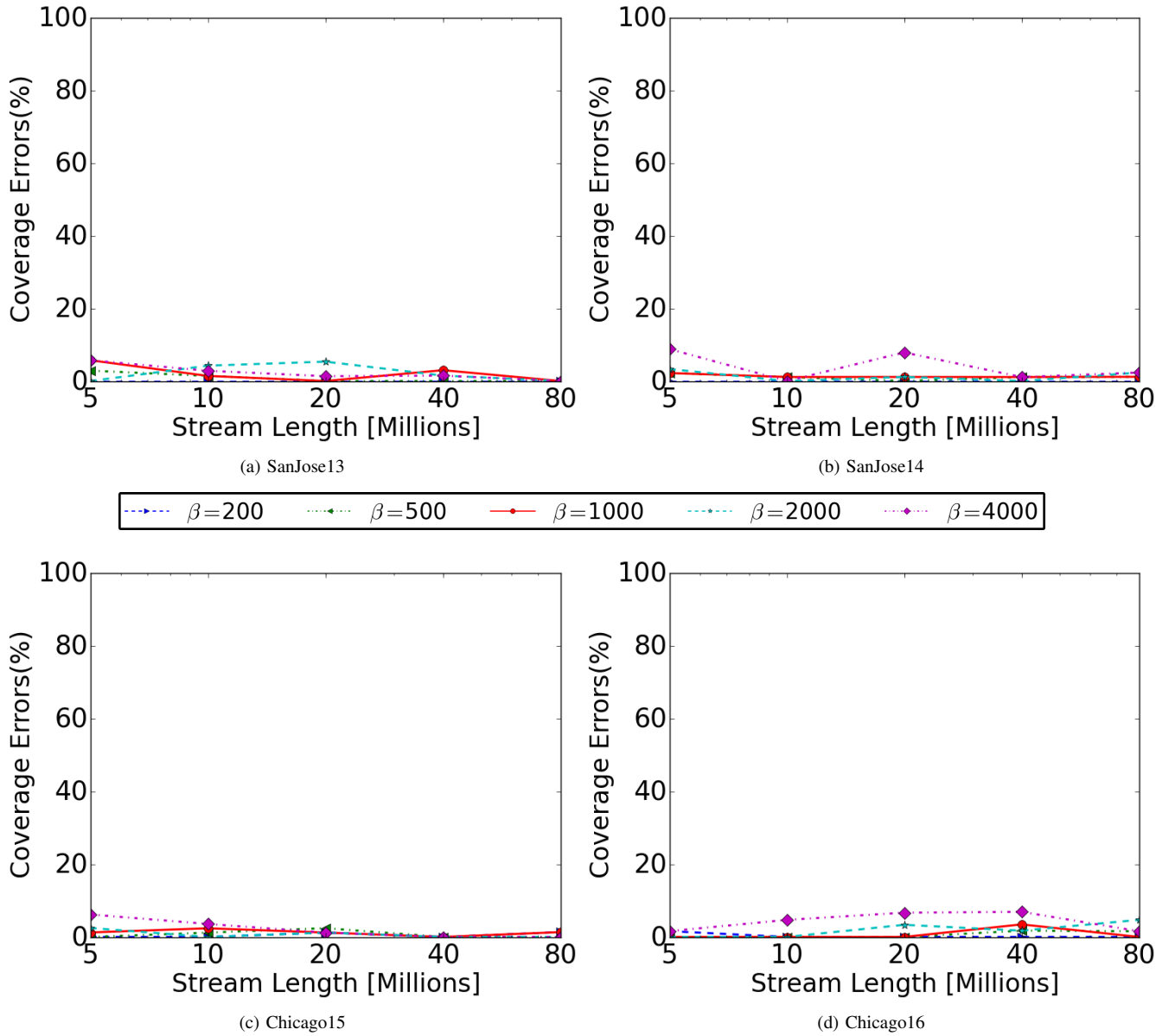
False positives exist in both deterministic and randomized HHH algorithms. Intuitively, such algorithms return a set that contains the HHH prefixes but not only HHH prefixes. The false positive probability reflects how many of the returned prefixes are not truly hierarchical heavy hitters. Here, we compare our VRHHH algorithm to the deterministic MST algorithm [17]. MST is known to be more accurate than the Partial Ancestry and Full Ancestry algorithms [17], [24]. As can be observed, in three out of the four workloads, VRHHH with $\beta = 1000$ yields a very small percentage of false positives. Similarly, the MST algorithm is more accurate than VRHHH and has no false positives at all. VRHHH has the highest false positive probability in the SanJos13 trace (Figure 6a), with about 20% false positives. This means that the returned HHH set is slightly larger than that of MST.

VI. OVS Implementation

This section describes how we extended Open vSwitch (OVS) to incorporate VRHHH. We start with a brief overview of virtual switching with an emphasis on the OVS architecture in Section VI-A. We then describe the experimental setup in Section VI-B and the evaluation results in SectionVI-C.

A. An Overview of Virtual Switching

Network Function Visualization (NFV) is about implementing network functionalities in virtual machines rather than on physical network devices. Virtual switching is a fundamental building block in these settings. It is essential to interconnect multiple *Virtual Network Functions (VNFs)* efficiently and compose a variety of network services. Therefore, virtual switching is important in datacenter networks.



(a) SanJose13

(b) SanJose14

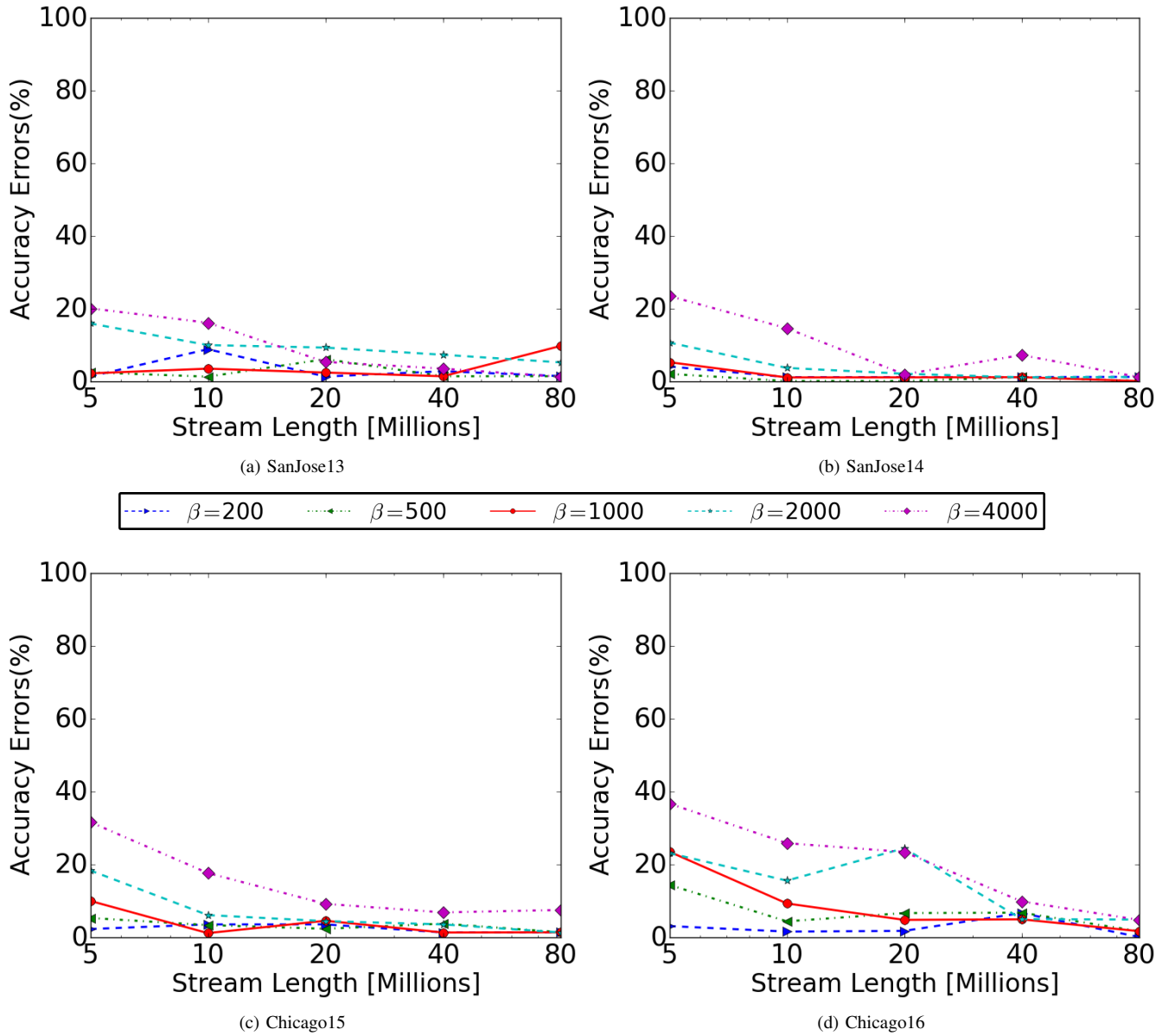
(c) Chicago15

(d) Chicago16

Fig. 4: Evaluation of the coverage of VRHHH for different sampling rates (β).

Virtual switching is required to achieve high throughput and low latencies to properly mimic physical network devices. Hence, efficient virtual switching is realized through sophisticated optimizations. A particularly important optimization is Intel's DPDK [33], which enables the entire packet processing to be performed in user-space. Intel's DPDK mitigates well-known performance overheads such as the ones involved in moving from kernel to user-space and reduces the number of memory copy operations by providing direct access to the NIC's buffers without memory copy. The DPDK library has received significant engagement from the NFV industry [33]. Therefore, our evaluation focuses on the DPDK version of Open vSwitch. The architectural design of OVS is composed of two main components, namely, ovs-vswitchd and ovsdb-server. Due to space constraints, we briefly describe the ovs-vswitchd component. The interested reader is referred to [34] for additional information. The vswitchd module implements control and data planes entirely in user space[1]. In short, network packets ingress the datapath (dpif/dpif-netdev OVS components) either from a physical port connected to the physical NIC or from a virtual port connected to a remote host (e.g., a VNF/virtual machine). Then, the datapath parses the headers and determines the set of actions to be applied. Such actions can be forwarding the packet to another port or rewrite a specific header.

[1]In contrast, in kernel-mode OVS, vswitchd implements the control plane in the user-space, while the dataplane is a module in the kernel.



(a) SanJose13

(b) SanJose14

(c) Chicago15

(d) Chicago16

Fig. 5: Evaluation of the accuracy of our method for different sampling rates (β).

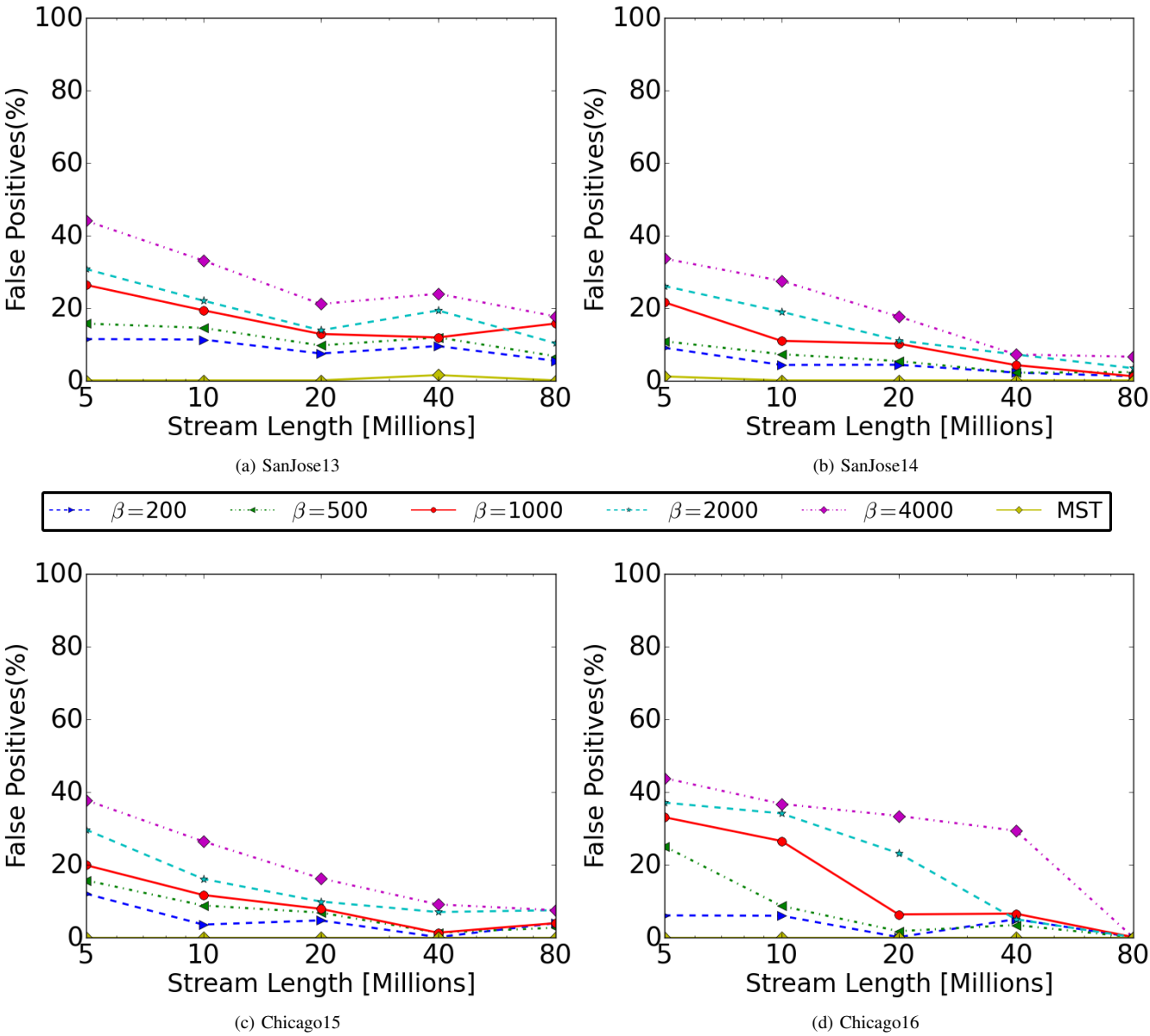
Our HHH measurement is performed as part of these actions. Specifically, in addition to the default set of actions performed by OVS, our extension performs HHH measurement. Therefore, it is essential that the computational overheads of such measurement are kept to a minimum.

B. Environment Setup

Our environment includes two identical HP ProLiant servers with an Intel Xeon E3-1220v2 processor running at 3.1 GHz with 8 GB RAM, an Intel 82599ES 10 Gbit/s network card and a CentOS 7.2.1511 with Linux kernel 3.10.0 operating system. The servers are directly connected through two physical interfaces. We use Open vSwitch 2.5 with Intel DPDK 2.02, where NIC physical ports are attached using *dpdk* ports.

One server is used as traffic generator while the other is used as *Device Under Test (DUT)*. Placed on the *DUT*, OVS receives packets on one network interface and then forwards them back to the second one. This is done to minimize the routing overheads so that we can evaluate the maximal impact of performing HHH measurements on the OVS throughput.

Traffic is replayed using the MoonGen traffic generator [35]. We choose MoonGen as it can generate packets considerably faster than our line rate. Moongen receives a real packet trace that only contains the packet headers. It then generates and transmits 100M packets. The generated packets preserve the original dataset's source and destination IP as well as the payload size. These fields are then used by our HHH measurement. The evaluation presented here includes the Chicago16 trace; similar throughput is achieved with the other traces.



(a) SanJose13

(b) SanJose14

(c) Chicago15

(d) Chicago16

Fig. 6: Comparison of our false positive rate to the state of the art – MST [17].

C. Throughput Evaluation

First, we evaluate the throughput of VRHHH for different byte sampling probabilities. That is, we vary β from 1 in 200 bytes to 1 in 4000 bytes. The evaluated accuracy is always $\varepsilon = 0.001$ in these measurements. Figure 7 depicts the results, showing that the throughput improves as we reduce β as expected. Yet, we reach saturation as some implementation overheads do not depend on β. For example, we still need to process the packet's headers and extracts its weight, randomize a number and so forth. At the smallest β value we measured, yielding a sampling ratio of 1 in 4000 bytes, we see a throughput reduction of only 4% compared to unmodified OVS. However, since the convergence time of VRHHH depends on β, we recommend sampling once out of 1000 bytes of traffic. This has a slightly higher overhead ($\approx$ 6%) but achieves similar accuracy compared to deterministic approaches.

Next, we evaluate our recommended VRHHH configuration ($\varepsilon = \beta = 0.001$) compared to previously suggested alternatives such as MST [17], Partial Ancestry and Full Ancestry [20]. The accuracy of these algorithms is also set to $\varepsilon = 0.001$. Figure 8 illustrates the maximum achievable throughput for each algorithm. As can be observed, our own VRHHH outperforms the alternatives. On average, VRHHH achieves a throughput of 9 Gbits which is only 6% less than an unmodified OVS (Vanilla OVS). In contrast, the fastest alternative is MST that degrades performance by $\approx 42\%$.

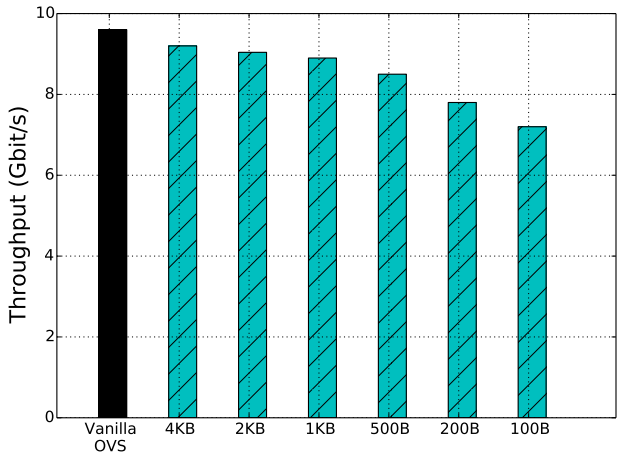


Fig. 7: Throughput of VRHHH with varying β ($\varepsilon = 0.001$)

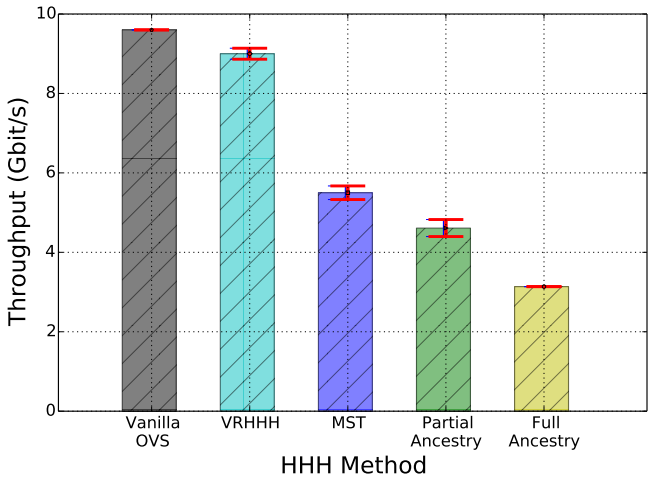


Fig. 8: A comparative throughput evaluation as function of the HHH algorithm ($\varepsilon = 0.001$, $\beta = 1KB$).

VII. Analysis

Our analysis shows that VRHHH solves the $(\delta, \epsilon, \theta)$ - Approximate Volumetric Hierarchical Heavy Hitters problem (Definition III.5) after encountering a certain amount of traffic. Our analysis utilizes the previous analysis of RHHH [24] that only handles unit weights for packets. Therefore, we start with a reduction in the original stream.

Definition VII.1. *Let $\mathbb{S}$ be the original packet streams with packets of the form $< x, w >$, where x is the packet id and w is the packet weight. The stream $\mathbb{S}_{\mathbb{U}}$ is generated by replacing each packet ($< x, w >$) in $\mathbb{S}$ with w identical packets of the form $< x, 1 >$.*

Intuitively, $\mathbb{S}_{\mathbb{U}}$ is created by replacing each packet in the stream whose size is x with x packets of size 1. E.g. a 1000-bytes packet is replaced with $1,000$ single byte packets.

Observation VII.1. *Algorithm 1 operates the same on $\mathbb{S}$ and on $\mathbb{S}_{\mathbb{U}}$.*

To see that Observation VII.1 holds, it is enough to observe Line 2 in which each packet decrements d by w. Thus, instead of processing 1 packet, we would process w packets of size 1. The difference $\mathbb{S}$ and $\mathbb{S}_{\mathbb{U}}$ is manifested in the case where d is zeroed or becomes negative by a certain packet. In that case, in $\mathbb{S}$ it would be negative, but in $\mathbb{S}_{\mathbb{U}}$ it would be zeroed. Note that in Line 6, the next sample would be the same for both cases. Moreover, the sampled packet has the same id in both cases. Hence, processing $\mathbb{S}$ is the same as processing $\mathbb{S}_{\mathbb{U}}$.

Next, we observe that the number of packets in $\mathbb{S}_{\mathbb{U}}$ is B, which is the number of bytes in $\mathbb{S}$.

Observation VII.2. *The number of packets in $\mathbb{S}_{\mathbb{U}}$ is the number of bytes in $\mathbb{S}$.*

Next, we observe that RHHH [24] and VRHHH are indistinguishable when running on $\mathbb{S}'$ as all the inputs are unitary.

Observation VII.3. *RHHH and VRHHH are indistinguishable when all packets are of weight 1. That is VRHHH is identical to $RHHH$ of the same configuration and $V = \beta H$.*

We now repeat Theorem VII.4 that is proved in [24]. The parameter V is a performance parameter similar to our own β, while the notation Z_α is the z value that satisfies $\Phi(z) = \alpha$ and $\Phi(z)$ is the cumulative density of the normal distribution with mean 0 and standard deviation of 1. For practical reasons, when $\delta > 10^{-6}$, $Z_\alpha \leq 5$. The theorem yields a traffic bound in **packets** that guarantees the accuracy of RHHH.

Theorem VII.4 (RHHH correctness [24])**.** *If $N > Z_{1-\frac{\delta_s}{2}} V {\varepsilon_s}^{-2}$, then RHHH solves $(\delta, \epsilon, \theta)$ -* Approximate Hierarchical Heavy Hitters.

Finally, we are ready to derive our own correctness proof.

Theorem VII.5. *If $B > \epsilon_s^{-2} Z_{1-\frac{\delta_s}{2}} \beta H$, then VRHHH solves the $(\delta, \epsilon, \theta)$ -* Approximate Volumetric Hierarchical Heavy Hitters *problem.*

Proof. Since running RHHH and VRHHH on $\mathbb{S}_{\mathbb{U}}$ is the same, then we can apply Theorem VII.4 and conclude that VRHHH solves the $(\delta, \epsilon, \theta)$ - Approximate Volumetric Hierarchical Heavy Hitters problem on $\mathbb{S}_{\mathbb{U}}$ if the number of packets is greater than $\epsilon_s^{-2} Z_{1-\frac{\delta_s}{2}} \beta H$. The number of packets in $\mathbb{S}_{\mathbb{U}}$ is the same as the number of bytes (Observation VII.2). Moreover, Observation VII.1 assures us that running VRHHH on $\mathbb{S}_{\mathbb{U}}$ and on $\mathbb{S}$ is the same. Thus, we get that if the number of **bytes** satisfies: $B > \epsilon_s^{-2} Z_{1-\frac{\delta_s}{2}} \beta H$, then VRHHH solves the $(\delta, \epsilon, \theta)$ - Approximate Volumetric Hierarchical Heavy Hitters on $\mathbb{S}$, concluding the proof. □

Theorem VII.5 is the main theorem in this work and provides a traffic bound for the correctness of VRHHH. That is, the algorithm provides its accuracy guarantee once enough packets were processed.

VIII. DISCUSSION

Our work is motivated by rapid increases in the intensity of DDoS attacks [36]. The rise is mainly attributed to the success of the Internet of Things (IoT) paradigm where large numbers of poorly secured connected devices are being deployed. In parallel, modern networks are supporting jumbo frames and large packet payloads which enable an attacker to deliver more traffic within the same number of packets.

To cope with this challenge, our work studies the volumetric hierarchical heavy hitters problem, which is an essential technique to identify the origins of a DDoS attack. Our work showed that volumetric measurement differs from the traditional per-packet measurement and that volumetric tools are required. Unfortunately, existing algorithms are too slow and admit inefficient software implementation.

We suggested a randomized algorithm named VRHHH which is considerably faster than existing deterministic algorithms but requires a certain traffic volume to converge. We showed that VRHHH asymptotically improves the update time of previous works, achieving $O(1)$ updates. VRHHH samples the traffic according to byte volume, and for each sampled byte it randomly selects which HHH prefixes to update. The average number of updates is proportional to the packet's byte size. Doing so enables VRHHH to successfully identify the HHH after a certain amount of byte traffic regardless of the packet sizes' distribution. We analytically proved this property and showed guaranteed accuracy and attractive traffic bounds.

We evaluated VRRRH over four real Internet traces and showed that its accuracy is comparable to that of the alternatives after just 80 million packets. We believe that the number of packets required for convergence is reasonable given modern network technology with 10-100 Gbit/s lines. We also implemented VRHHH on a dpdk enabled Open vSwitch (OVS) and showed that it enables two-dimensional volumetric HHH measurement with reasonable 4%-6% overheads. In contrast, the fastest alternative incurs as much as 42% throughput reduction.

VRHHH is an important step forward in realizing volumetric HHH measurement in networking devices. Looking into the future, we plan to pursue autonomous DDoS attack mitigation at the switch level which would enable future networks to better cope with such attacks. Looking into the future, we plan to extend VRHHH to cloud settings where multiple measurement points report to an SDN controller that forms a network-wide view of the traffic [37]–[39].

REFERENCES

[1] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient Datacenter Load Balancing in the Wild," in *ACM SIGCOMM*, 2017.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *ACM SIGCOMM*, 2011.

[3] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers," in *IEEE HOTI*, 2010.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *ACM CoNEXT*, 2011.

[5] L. Ying, R. Srikant, and X. Kang, "The Power of Slightly More than One Sample in Randomized Load Balancing," in *IEEE INFOCOM*, 2015.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," in *ACM SIGCOMM*, 2014.

[7] G. Einziger, R. Friedman, and B. Manes, "TinyLFU: A Highly Efficient Cache Admission Policy," *ACM Transactions on Storage (ToS)*, vol. 13, no. 4, 2017.

[8] O. Tilmans, T. Bühler, I. Poese, S. Vissicchio, and L. Vanbever, "Stroboscope: Declarative network monitoring on a budget," in *USENIX NSDI*, 2018.

[9] D. P. Woodruff, "New Algorithms for Heavy Hitters in Data Streams (Invited Talk)," in *ICDT*, 2016.

[10] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *ACM SOSR*, 2017.

[11] G. Einziger, M. C. Luizelli, and E. Waisbard, "Constant time weighted frequency estimation for virtual network functionalities," in *IEEE ICCCN*, 2017.

[12] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy Hitters in Streams and Sliding Windows," in *IEEE INFOCOM*, 2016.

[13] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *IEEE INFOCOM*, 2017.

[14] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Randomized admission policy for efficient top-k and frequency estimation," in *IEEE INFOCOM*, 2017.

[15] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications," in *ACM IMC*, 2004.

[16] V. Sekar, N. Duffield, O. Spatscheck, J. van der Merwe, and H. Zhang, "LADS: Large-scale Automated DDOS Detection System," in *USENIX ATEC*, 2006, pp. 16–16.

[17] M. Mitzenmacher, T. Steinke, and J. Thaler, "Hierarchical Heavy Hitters with the Space Saving Algorithm," in *ALENEX*, 2012.

[18] R. B. Basat, G. Einziger, and R. Friedman, "Fast Flow Volume Estimation," in *ACM ICDCN*, 2018.

[19] L. Jose and M. Yu, "Online measurement of large traffic aggregates on commodity switches," in *USENIX Hot-ICE*, 2011.

[20] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding Hierarchical Heavy Hitters in Streaming Data," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 4, Feb. 2008.

[21] K. Nyalkalkar, S. Sinhay, M. Bailey, and F. Jahanian, "A comparative study of two network-based anomaly detection methods," in *IEEE INFOCOM*, 2011.

[22] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system." in *USENIX ATC*, 2006.

[23] R. Ben Basat, G. Einziger, and R. Friedman, "Fast Flow Volume Estimation," in *ACM ICDCN*, 2018.

[24] R. Ben-Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *ACM SIGCOMM*, 2017.

[25] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient Computation of Frequent and Top-k Elements in Data Streams," in *ICDT*, 2005.

[26] G. Cormode and M. Hadjieleftheriou, "Methods for Finding Frequent Items in Data Streams," *J. VLDB*, vol. 19, no. 1, 2010.

[27] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding Hierarchical Heavy Hitters in Data Streams," in *VLDB*, 2003.

[28] ——, "Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-dimensional Data," in *SIGMOD*, 2004.

[29] J. Hershberger, N. Shrivastava, S. Suri, and C. D. Tóth, "Space Complexity of Hierarchical Heavy Hitters in Multi-dimensional Data Streams," in *ACM PODS*, 2005.

[30] D. Anderson, P. Bevan, K. Lang, E. Liberty, L. Rhodes, and J. Thaler, "A high-performance algorithm for identifying frequent items in data streams," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, pp. 268–282. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131407

[31] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive random sampling for load change detection," *ACM SIGMETRICS*, 2002.

[32] P. Hick, "CAIDA Anonymized 2013-2016 Internet Traces, equinix-chicago and equinix-sanjose, Direction A."

[33] "Intel DPDK, http://dpdk.org/."
[34] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *USENIX NSDI*, May 2015.
[35] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *ACM IMC*, 2015.
[36] T. Republic, "Ddos attacks increased 91% in 2017 thanks to iot," https://www.techrepublic.com/article/ddos-attacks-increased-91-in-2017-thanks-to-iot/.
[37] R. Ben Basat, G. Einziger, J. Moraney, and D. Raz, "Network-wide routing oblivious heavy hitters," in *ACM/IEEE ANCS*, 2018.
[38] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *SOSR*. ACM, 2018.
[39] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. L, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *ACM SIGCOMM*, 2018.