Constant Time Updates in Hierarchical Heavy Hitters

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

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

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

Marcelo C. Luizelli
UFRGS
mcluizelli@inf.ufrgs.br

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

ABSTRACT

Monitoring tasks, such as anomaly and DDoS detection, require identifying frequent flow aggregates based on common IP prefixes. These are known as *hierarchical heavy hitters* (HHH), where the hierarchy is determined based on the type of prefixes of interest in a given application. The per packet complexity of existing HHH algorithms is proportional to the size of the hierarchy, imposing significant overheads.

In this paper, we propose a randomized constant time algorithm for HHH. We prove probabilistic precision bounds backed by an empirical evaluation. Using four real Internet packet traces, we demonstrate that our algorithm indeed obtains comparable accuracy and recall as previous works, while running up to 62 times faster. Finally, we extended Open vSwitch (OVS) with our algorithm and showed it is able to handle 13.8 million packets per second. In contrast, incorporating previous works in OVS only obtained 2.5 times lower throughput.

CCS CONCEPTS

• **Networks → Network measurement**;

KEYWORDS

Streaming, Heavy Hitters, Measurement, Monitoring

ACM Reference format:
Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C. Luizelli, and Erez Waisbard. 2017. Constant Time Updates in Hierarchical Heavy Hitters. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages.
https://doi.org/10.1145/3098822.3098832

1 INTRODUCTION

Network measurements are essential for a variety of network functionalities such as traffic engineering, load balancing, quality of service, caching, anomaly and intrusion detection [2, 3, 8, 16, 18, 22, 29, 45]. A major challenge in performing and maintaining network measurements comes from rapid line rates and the large number of active flows.

Previous works suggested identifying *Heavy Hitter* (HH) flows [44] that account for a large portion of the traffic. Indeed, approximate HH are used in many functionalities and can be captured quickly and efficiently [5–7, 20, 42]. However, applications such as anomaly detection and *Distributed Denial of Service* (DDoS) attack detection require more sophisticated measurements [41, 46]. In such attacks, each device generates a small portion of the traffic but their combined volume is overwhelming. HH measurement is therefore insufficient as each individual device is not a heavy hitter.

Hierarchical Heavy Hitters (HHH) account aggregates of flows that share certain IP prefixes. The structure of IP addresses implies a prefix based hierarchy as defined more precisely below. In the DDoS example, HHH can identify IP prefixes that are suddenly responsible for a large portion of traffic and such an anomaly may very well be a manifesting attack. Further, HHH can be collected in one dimension, e.g., a single source IP prefix hierarchy, or in multiple dimensions, e.g., a hierarchy based on both source and destination IP prefixes.

Previous works [14, 35] suggested deterministic algorithms whose update complexity is proportional to the hierarchy's size. These

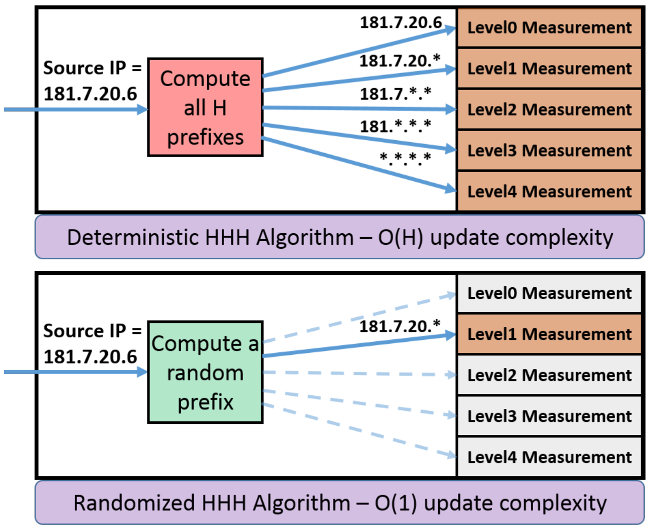


Figure 1: A high level overview of this work. Previous algorithms' update requires $\Omega(H)$ run time, while we perform at most a single $O(1)$ update.

algorithms are currently too slow to cope with line speeds. For example, a 100 Gbit link may deliver over 10 million packets per second, but previous HHH algorithms cannot cope with this line speed on existing hardware. The transition to IPv6 is expected to increase hierarchies' sizes and render existing approaches even slower.

Emerging networking trends such as *Network Function Virtualization* (NFV) enable virtual deployment of network functionalities. These are run on top of commodity servers rather than on custom made hardware, thereby improving the network's flexibility and reducing operation costs. These trends further motivate fast software based measurement algorithms.

1.1 Contributions

First, we define a probabilistic relaxation of the HHH problem. Second, we introduce *Randomized HHH* (a.k.a. RHHH), a novel randomized algorithm that solves probabilistic HHH over single and multi dimensional hierarchical domains. Third, we evaluate RHHH on four different real Internet traces and demonstrate a speedup of up to X62 while delivering similar accuracy and recall ratios. Fourth, we integrate RHHH with *Open vSwitch* (OVS) and demonstrate a capability of monitoring HHH at line speed, achieving a throughput of up to 13.8M packets per second. Our algorithm also achieves X2.5 better throughput than previous approaches. To the best of our knowledge, our work is the first to perform OVS multi dimensional HHH analysis in line speed.

Intuitively, our RHHH algorithm operates in the following way, as illustrated in Figure 1: We maintain an instance of a heavy-hitters detection algorithm for each level in the hierarchy, as is done in [35]. However, whenever a packet arrives, we randomly select only a single level to update using its respective instance of heavy-hitters rather than updating all levels (as was done in [35]). Since the update time of each individual level is $O(1)$, we obtain an $O(1)$ *worst case* update time. The main challenges that we address in this paper are in formally analyzing the accuracy of this scheme and exploring how well it works in practice with a concrete implementation.

The update time of previous approaches is $O(H)$, where H is the size of the hierarchy. An alternative idea could have been to simply sample each packet with probability $\frac{1}{H}$, and feed the sampled packets to previous solutions. However, such a solution only provides an $O(1)$ *amortized* running time. Bounding the worst case behavior to $O(1)$ is important when the counters are updated inside the data path. In such cases, performing an occasional very long operation could both delay the corresponding "victim" packet, and possibly cause buffers to overflow during the relevant long processing. Even in off-path processing, such as in an NFV setting, occasional very long processing creates an unbalanced workload, challenging schedulers and resource allocation schemes.

Roadmap. The rest of this paper is organized as follows: We survey related work on HHH in Section 2. We introduce the problem and our probabilistic algorithm in Section 3. For presentational reasons, we immediately move on to the performance evaluation in Section 4 followed by describing the implementation in OVS in Section 5. We then prove our algorithm and analyze its formal guarantees in Section 6. Finally, we conclude with a discussion in Section 7.

2 RELATED WORK

In one dimension, HHH were first defined by [12], which also introduced the first streaming algorithm to approximate them. Additionally, [28] offered a TCAM approximate HHH algorithm for one dimension. The HHH problem was also extended to multiple dimensions [13, 14, 23, 35, 46].

The work of [31] introduced a single dimension algorithm that requires $O\left(\frac{H^2}{\epsilon}\right)$space, where the symbol H denotes the size of the hierarchy and ϵ is the allowed relative estimation error for each single flow's frequency. Later, [43] introduced a two dimensions algorithm that requires $O\left(\frac{H^{3/2}}{\epsilon}\right)$space and update time[1]. In [14], the trie based Full Ancestry and Partial Ancestry algorithms were proposed. These use $O\left(\frac{H\log(N\epsilon)}{\epsilon}\right)$ space and requires $O\left(H\log(N\epsilon)\right)$ time per update.

The seminal work of [35] introduced and evaluated a simple multi dimensional HHH algorithm. Their algorithm uses a separate copy of Space Saving [34] for each lattice node and upon packet arrival, all lattice nodes are updated. Intuitively, the problem of finding hierarchical heavy hitters can be reduced to solving multiple non hierarchical heavy hitters problems, one for each possible query. This algorithm provides strong error and space guarantees and its update time does not depend on the stream length. Their algorithm requires $O\left(\frac{H}{\epsilon}\right)$ space and its update time for unitary inputs is $O\left(H\right)$ while for weighted inputs it is $O\left(H\log\frac{1}{\epsilon}\right)$.

The update time of existing methods is too slow to cope with modern line speeds and the problem escalates in NFV environments that require efficient software implementations. This limitation is both empirical and asymptotic as some settings require large hierarchies.

Our paper describes a novel algorithm that solves a probabilistic version of the hierarchical heavy hitters problem. We argue that in practice, our solution's quality is similar to previously suggested deterministic approaches while the runtime is dramatically improved. Formally, we improve the update time to $O(1)$, but require a minimal number of packets to provide accuracy guarantees. We argue that this trade off is attractive for many modern networks that route a continuously increasing number of packets.

3 RANDOMIZED HHH (RHHH)

We start with an intuitive introductory to the field as well as preliminary definitions and notations. Table 2 summarizes notations used in this work.

3.1 Basic terminology

We consider IP addresses to form a hierarchical domain with either bit or byte size granularity. *Fully specified* IP addresses are the lowest level of the hierarchy and can be generalized. We use $\mathcal{U}$ to denote the domain of fully specified items. For example, 181.7.20.6 is a fully specified IP address and 181.7.20.∗ generalizes it by a single byte. Similarly, 181.7.∗ generalizes it by two bytes and formally, a fully specified IP address is generalized by any of its prefixes. The *parent* of an item is the longest prefix that generalizes it.

[1]Notice that in two dimensions, H is a square of its counter-part in one dimension.

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

In two dimensions, we consider a tuple containing source and destination IP addresses. A fully specified item is fully specified in both dimensions. For example, $(\langle 181.7.20.6\rangle \rightarrow \langle 208.67.222.222\rangle)$ is fully specified. In two dimensional hierarchies, each item 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 both parents to $(\langle 181.7.20.6\rangle \rightarrow \langle 208.67.222.222\rangle)$.

Definition 3.1 (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. A byte granularity two dimensional lattice is illustrated in Table 1. In the table, each lattice node is generalized by all nodes that are upper or more to the left. The most generalized node $(*, *)$ is called *fully general* and the most specified node $(s1.s2.s3.s4, d1.d2.d3.d4)$ is called *fully specified*. We denote H the hierarchy's size as the number of nodes in the lattice. For example, in IPv4, byte level one dimensional hierarchies imply $H = 5$ as each IP address is divided into four bytes and we also allow querying $*$.

Definition 3.2. Given a prefix p and a set of prefixes P, we define $G(p|P)$ as the set of prefixes:

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

Intuitively, $G(p|P)$ are the prefixes in P that are most closely generalized by p. E.g., let $p =< 142.14.* >$ and the set $P = \{< 142.14.13.* >, < 142.14.13.14 >\}$, then $G(p|P)$ only contains $< 142.14.13.* >$.

We consider a stream $\mathbb{S}$, where at each step a packet of an item e arrives. Packets belong to a hierarchical domain of size H, and can be generalized by multiple prefixes as explained above. Given a fully specified item e, f_e is the number of occurrences e has in $\mathbb{S}$. Definition 3.3 extends this notion to prefixes.

Definition 3.3. (Frequency) Given a prefix p, the frequency of p is:

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

Our implementation utilizes Space Saving [34], a popular (non hierarchical) heavy hitters algorithm, but other algorithms can also be used. Specifically, we can use any *counter algorithm* that satisfies Definition 3.4 below and can also find heavy hitters, such as [17, 30, 33]. We use Space Saving because it is believed to have an empirical edge over other algorithms [10, 11, 32].

Symbol	Meaning
$\mathbb{S}$	Stream
N	Current number of packets (in all flows)
H	Size of Hierarchy
V	Performance parameter, $V \geq H$
S_x^i	Variable for the i'th appearance of a prefix x.
S_x	Sampled prefixes with id x.
S	Sampled prefixes from all ids.
$\mathcal{U}$	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.
$C_{q\|P}$	Conditioned frequency of q with respect to P
$G(q\|P)$	Subset of P with the closest prefixes to q.
f_q	Frequency of prefix q
$\widehat{f_q^+}, \widehat{f_q^-}$	Upper,lower bound for f_q

Table 2: List of Symbols

The minimal requirements from an algorithm to be applicable to our work are defined in Definition 3.4. This is a weak definition and most counter algorithms satisfy it with $\delta = 0$. Sketches [9, 15, 19] can also be applicable here, but to use them, each sketch should also maintain a list of heavy hitter items (Definition 3.5).

Definition 3.4. An algorithm solves the (ϵ, δ) - Frequency Estimation problem if for any prefix (x), it provides $\widehat{f_x}$ s.t.:

$$\Pr\left[\left|f_x - \widehat{f_x}\right| \leq \varepsilon N\right] \geq 1 - \delta.$$

Definition 3.5 (Heavy hitter (HH)). Given a threshold (θ), a fully specified item (e) is a **heavy hitter** if its frequency (f_e) is above the threshold: $\theta \cdot N$, i.e., $f_e \geq \theta \cdot N$.

Our goal is to identify the hierarchical heavy hitter prefixes whose frequency is above the threshold ($\theta \cdot N$). However, if the frequency of a prefix exceeds the threshold then so is the frequency 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 definition of *conditioned frequency* ($C_{p|P}$). Intuitively, $C_{p|P}$ measures the **additional** traffic prefix p adds to a set of previously selected HHHs (P), and it is defined as follows.

Definition 3.6. (Conditioned frequency) The conditioned frequency of a prefix p with respect to a prefix set P is:

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

$C_{p|P}$ is derived by subtracting the frequency of fully specified items that are already generalized by items in P from p's frequency (f_p). In two dimensions, exclusion inclusion principles are used to avoid double counting.

We now continue and describe how exact hierarchical heavy hitters (with respect to $C_{p|P}$) are found. To that end, partition the hierarchy to levels as explained in Definition 3.7.

Definition 3.7 (Hierarchy Depth). Define L, the *depth of a hierarchy*, as follows: Given a fully specified element e, we consider a set of prefixes such that: $e \prec p_1 \prec p_2, .. \prec p_L$ where $e \neq p_1 \neq p_2 \neq ... \neq p_L$ and L is the maximal size of that set. We also define the function $level(p)$ that given a prefix p returns p's maximal location in the chain, i.e., the maximal chain of generalizations that ends in p.

To calculate exact heavy hitters, we go over fully specified items ($level0$) and add their heavy hitters to the set HHH_0. Using HHH_0, we calculate conditioned frequency for prefixes in $level1$ and if $C_{p|HHH_0} \geq \theta \cdot N$ we add p to HHH_1. We continue this process until the last level (L) and the exact heavy hitters are the set HHH_L. Next, we define HHH formally.

Definition 3.8 (Hierarchical HH (HHH)). The set HHH_0 contains the fully specified items e s.t. $f_e \geq \theta \cdot N$. Given a prefix p from level(l), $0 \leq l \leq L$, we define:

$$HHH_l = \\ HHH_{l-1} \cup \left\{p : \left(p \in level\,(l) \wedge C_{p|HHH_{l-1}} \geq \theta \cdot N\right)\right\}.$$

The set of exact hierarchical heavy hitters HHH is defined as the set HHH_L.

For example, consider the case where $\theta N = 100$ and assume that the following prefixes with their frequencies are the only ones above θN. $p_1 = (< 101.* >, 108)$ and $p_2 = (< 101.102.* >, 102)$. Clearly, both prefixes are heavy hitters according to Definition 3.5. However, the conditioned frequency of $p1$ is $108 - 102 = 6$ and that of p_2 is 102. Thus only p_2 is an HHH prefix.

Finding exact hierarchical heavy hitters requires plenty of space. Indeed, even finding exact (non hierarchical) heavy hitters requires linear space [37]. Such a memory requirement is prohibitively expensive and motivates finding approximate HHHs.

Definition 3.9 ((ϵ, θ)−approximate HHH). An algorithm solves (ϵ, θ) - Approximate Hierarchical Heavy Hitters if after processing any stream $\mathbb{S}$ of length N, it returns a set of prefixes (P) that satisfies the following conditions:

- **Accuracy:** for every prefix $p \in P$, $\left|f_p - \widehat{f_p}\right| \leq \varepsilon N$.
- **Coverage:** for every prefix $q \notin P$: $C_{q|P} < \theta N$.

Approximate HHH are a set of prefixes (P) that satisfies accuracy and coverage; there are many possible sets that satisfy both these properties. Unlike exact HHH, we do no require that for $p \in P$, $C_{p|P} \geq \theta N$. Unfortunately, if we add such a requirement then [23] proved a lower bound of $\Omega\left(\frac{1}{\theta^{d+1}}\right)$ space, where d is the number of dimensions. This is considerably more space than is used in our work ($\frac{H}{\epsilon}$) that when $\theta \propto \epsilon$ is also $\frac{H}{\theta}$.

Finally, Definition 3.10 defines the probabilistic approximate HHH problem that is solved in this paper.

Definition 3.10 ((δ, ϵ, θ)−approximate HHHs). An algorithm $\mathbb{A}$ solves (δ, ϵ, θ) - Approximate Hierarchical Heavy Hitters if after processing any stream $\mathbb{S}$ of length N, it returns a set of prefixes P that, for an arbitrary run of the algorithm, satisfies the following:

- **Accuracy:** for every prefix $p \in P$,
$$\Pr\left(\left|f_p - \widehat{f_p}\right| \leq \varepsilon N\right) \geq 1 - \delta.$$
- **Coverage:** given a prefix $q \notin P$,
$$\Pr\left(C_{q|P} < \theta N\right) \geq 1 - \delta.$$

Notice that this is a simple probabilistic relaxation of Definition 3.9. Our next step is to show how it enables the development of faster algorithms.

Algorithm 1 Randomized HHH algorithm

```
   Initialization: $\forall d \in [L] : HH[d] = $ HH_Alg $(\epsilon_a^{-1})$
1: function Update( $x$)
2:     $d = randomInt(0, V)$
3:     if $d < H$ then
4:         Prefix $p = x\&HH[d].mask$                    ▷ Bitwise AND
5:         $HH[d].INCREMENT(p)$
6:     end if
7: end function
8: function Output($\theta$)
9:     $P = \phi$
10:    for Level $l = |H|$ down to 0. do
11:        for each $p$ in level $l$ do
12:            $\widehat{C_{p|P}} = \widehat{f_p}^{+} + calcPred(p, P)$
13:            $\widehat{C_{p|P}} = \widehat{C_{p|P}} + 2Z_{1-\delta}\sqrt{NV}$
14:            if $\widehat{C_{p|P}} \geq \theta N$ then
15:                $P = P \cup \{p\}$                ▷ $p$ is an HHH candidate
16:                $print\left(p, \widehat{f_p}^{-}, \widehat{f_p}^{+}\right)$
17:            end if
18:        end for
19:    end for
20:    return $P$
21: end function
```

Algorithm 2 calcPred for one dimension

```
1: function CALCPRED(prefix $p$, set $P$)
2:     $R = 0$
3:     for each $h \in G(p|P)$ do
4:         $R = R - \widehat{f_h}^-$
5:     end for
6:     return $R$
7: end function
```

Algorithm 3 calcPred for two dimensions

```
1: function CALCPRED(prefix $p$, set $P$)
2:     $R = 0$
3:     for each $h \in G(p|P)$ do
4:         $R = R - \widehat{f_h}^-$
5:     end for
6:     for each pair $h, h' \in G(p|P)$ do
7:         $q = glb(h, h')$
8:         if $\nexists h_3 \neq h, h' \in G(p|P), q \preceq h_3$ then
9:             $R = R + \widehat{f_q}^+$
10:        end if
11:    end for
12:    return $R$
13: end function
```

3.2 Randomized HHH

Our work employs the data structures of [35]. That is, we use a matrix of H independent HH algorithms, and each node is responsible for a single prefix pattern.

Our solution, *Randomized HHH* (RHHH), updates **at most a single** randomly selected HH instance that operates in $O(1)$. In contrast, [35] updates **every** HH algorithm for each packet and thus operates in $O(H)$.

Specifically, for each packet, we randomize a number between 0 and V and if it is smaller than H, we update the corresponding HH algorithm. Otherwise, we ignore the packet. Clearly, V is a performance parameter: when $V = H$, every packet updates one of the HH algorithms whereas when $V \gg H$, most packets are ignored. Intuitively, each HH algorithm receives a *sample* of the stream. We need to prove that given enough traffic, hierarchical heavy hitters can still be extracted.

Pseudocode of RHHH is given in Algorithm 1. RHHH uses the same algorithm for both one and two dimensions. The differences between them are manifested in the *calcPred* method. Pseudocode of this method is found in Algorithm 2 for one dimension and in Algorithm 3 for two dimensions.

Definition 3.11. The underlying estimation provides us with upper and lower estimates for the number of times prefix p was updated (X_p). We denote: $\widehat{X^p}^+$ to be an upper bound for X_p and $\widehat{X^p}^-$ to be a lower bound. For simplicity of notations, we define the following:
$\widehat{f_p} \triangleq \widehat{X^p}V$ – an estimator for p's frequency.
$\widehat{f_p^+} \triangleq \widehat{X^p}^+V$ – an upper bound for p's frequency.
$\widehat{f_p^-} \triangleq \widehat{X^p}^-V$ – a lower bound for p's frequency.

Note these bounds ignore the sample error that is accounted separately in the analysis.

The output method of RHHH starts with fully specified items and if their frequency is above θN, it adds them to P. Then, RHHH iterates over their parent items and calculates a conservative estimation of their conditioned frequency with respect to P. Conditioned frequency is calculated by an upper estimate to (f_p^+) amended by the output of the *calcPred* method. In a single dimension, we reduce the lower bounds of p's closest predecessor HHHs. In two dimensions, we use inclusion and exclusion principles to avoid double counting. In addition, Algorithm 3 uses the notation of *greater lower bound (glb)* that is formally defined in Definition 3.12. Finally, we add a constant to the conditioned frequency to account for the sampling error.

Definition 3.12. 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.

In two dimensions, $C_{p|P}$ is first set to be the upper bound on p's frequency (Line 12, Algorithm 1). Then, we remove previously selected descendant heavy hitters (Line 4, Algorithm 3). Finally, we add back the common descendant (Line 9, Algorithm 3)).

Note that the work of [35] showed that their structure extends to higher dimensions, with only a slight modification to the Output method to ensure that it conservatively estimates the conditioned count of each prefix. As we use the same general structure, their extension applies in our case as well.

4 EVALUATION

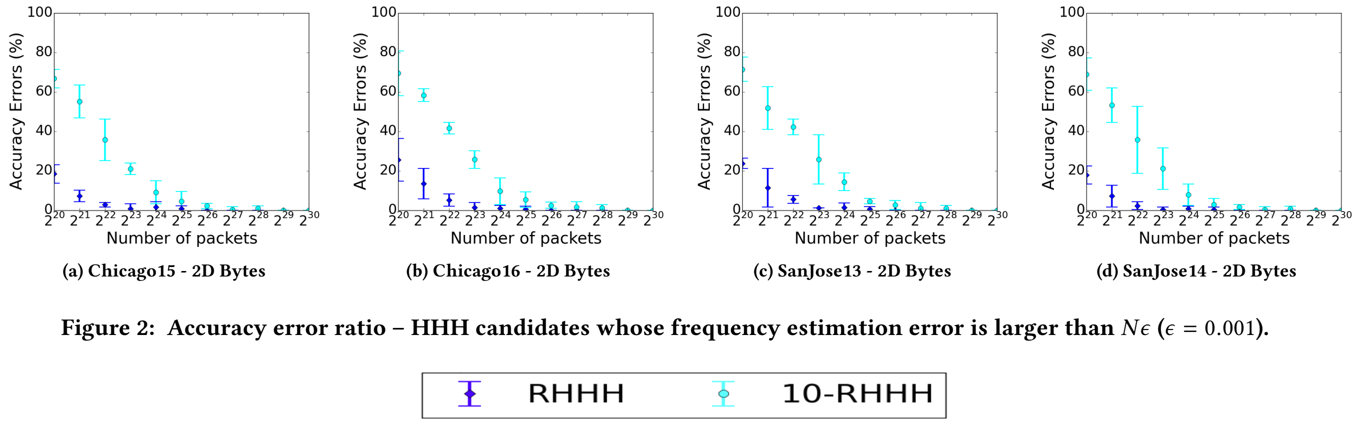
Our evaluation includes MST [35], the Partial and Full Ancestry [14] algorithms and two configurations of RHHH, one with $V = H$ (RHHH) and the other with $V = 10 \cdot H$ (10-RHHH). RHHH performs a single update operation per packet while 10-RHHH performs such an operation only for 10% of the packets. Thus, 10-RHHH is considerably faster than RHHH but requires more traffic to converge.

The evaluation was performed on a single Dell 730 server running Ubuntu 16.04.01 release. The server has 128GB of RAM and an Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz processor.

Our evaluation includes four datasets, each containing a mix of 1 billion UDP/TCP and ICMP packets collected from major backbone routers in both Chicago [26, 27] and San Jose [24, 25] during the years 2014-2016. We considered source hierarchies in byte (1D Bytes) and bit (1D Bits) granularities, as well as a source/destination byte hierarchy (2D Bytes). Such hierarchies were also used by [14, 35]. We ran each data point 5 times and used two-sided Student's t-test to determine 95% confidence intervals.

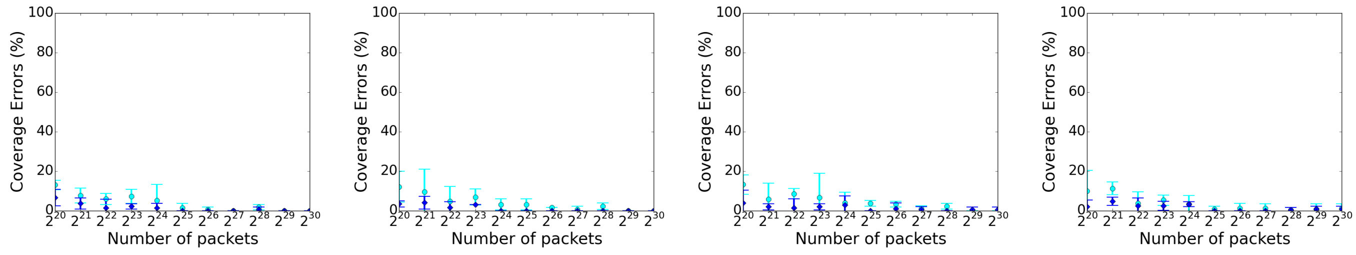
4.1 Accuracy and Coverage Errors

RHHH has a small probability of both accuracy and coverage errors that are not present in previous algorithms. Figure 2 quantifies the accuracy errors and Figure 3 quantifies the coverage errors. As can be seen, RHHH becomes more accurate as the trace progresses. Our theoretic bound (ψ as derived in Section 6 below) for these parameters is about 100 million packets for RHHH and about 1 billion packets for 10-RHHH. Indeed, these algorithms converge once they reach their theoretical bounds (see Theorem 6.19).



(a) Chicago15 - 2D Bytes (b) Chicago16 - 2D Bytes (c) SanJose13 - 2D Bytes (d) SanJose14 - 2D Bytes

Figure 2: Accuracy error ratio – HHH candidates whose frequency estimation error is larger than $N\epsilon$ ($\epsilon = 0.001$).



(a) Chicago15 - 2D Bytes (b) Chicago16 - 2D Bytes (c) SanJose13 - 2D Bytes (d) SanJose14 - 2D Bytes

Figure 3: The percentage of Coverage errors – elements q such that $q \notin P$ and $C_{q|P} \geq N\theta$ (false negatives).

4.2 False Positives

Approximate HHH algorithms find all the HHH prefixes but they also return non HHH prefixes. *False positives* measure the ratio non HHH prefixes pose out of the returned HHH set. Figure 4 shows a comparative measurement of false positive ratios in the Chicago 16 and San Jose 14 traces. Every point was measured for $\epsilon = 0.1\%$ and $\theta = 1\%$. As shown, for RHHH and 10-RHHH the false positive ratio is reduced as the trace progresses. Once the algorithms reach their theoretic grantees (ψ), the false positives are comparable to these of previous works. In some cases, RHHH and 10-RHHH even perform slightly better than the alternatives.

4.3 Operation Speed

Figure 5 shows a comparative evaluation of operation speed. Figure 5a, Figure 5b and Figure 5c show the results of the San Jose 14 trace for 1D byte hierarchy ($H = 5$), 1D bit hierarchy ($H = 33$) and 2D byte hierarchy ($H = 25$), respectively. Similarly, Figure 5d, Figure 5e and Figure 5f show results for the Chicago 16 trace on the same hierarchical domains. Each point is computed for $250M$ long packet traces. Clearly, the performance of RHHH and 10-RHHH is relatively similar for a wide range of ε values and for different data sets. Existing works depend on H and indeed run considerably slower for large H values.

Another interesting observation is that the Partial and Full Ancestry [14] algorithms improve when ε is small. This is because in that case there are few replacements in their trie based structure, as is directly evident by their $O(H \log(N\epsilon))$ update time, which is decreasing with ϵ. However, the effect is significantly lessened when H is large.

RHHH and 10-RHHH achieve speedup for a wide range of ε values, while 10-RHHH is the fastest algorithm overall. For one dimensional byte level hierarchies, the achieved speedup is up to X3.5 for RHHH and up to X10 for 10-RHHH. For one dimensional bit level hierarchies, the achieved speedup is up to X21 for RHHH and up to X62 for 10-RHHH. Finally, for 2 dimensional byte hierarchies, the achieved speedup is up to X20 for RHHH and up to X60 for 10-RHHH. Evaluation on Chicago15 and SanJose13 yielded similar results, which are omitted due to lack of space.

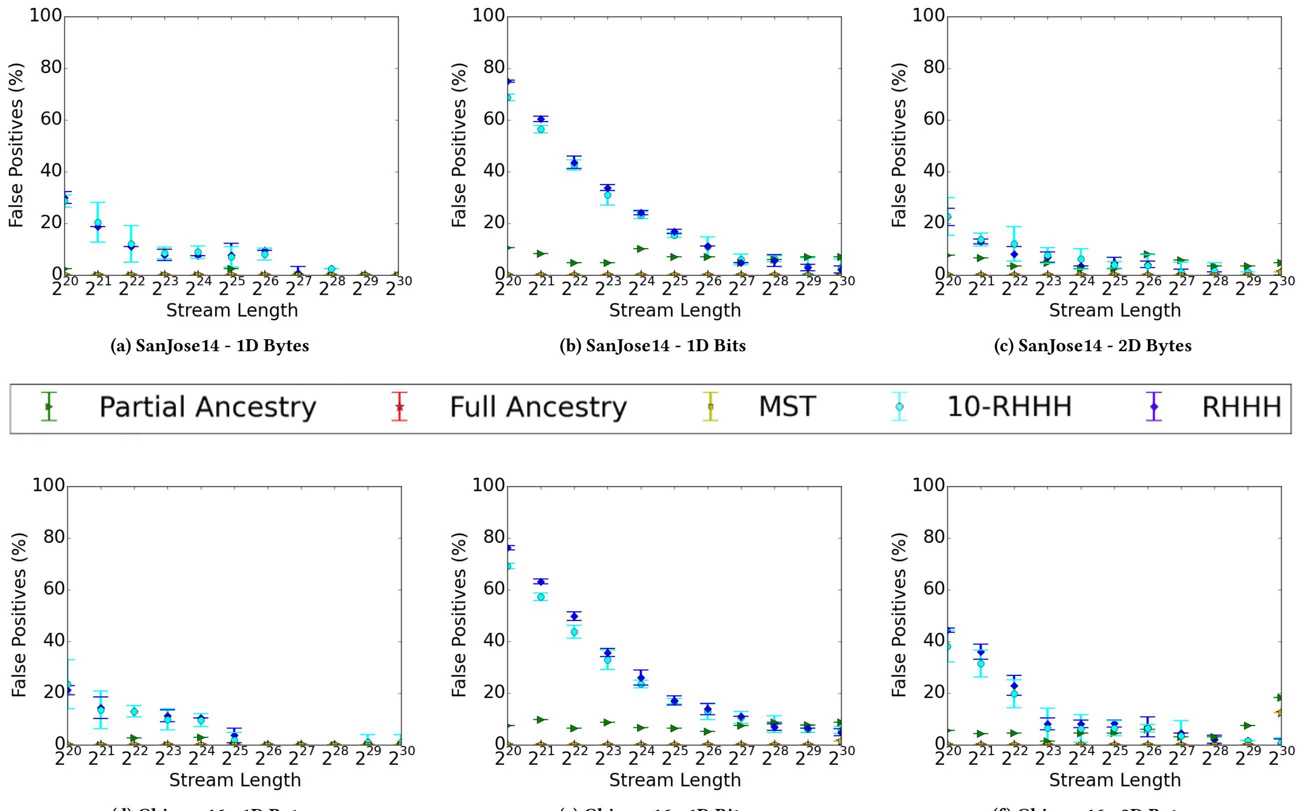
5 VIRTUAL SWITCH INTEGRATION

This section describes how we extended Open vSwitch (OVS) to include approximate HHH monitoring capabilities. For completeness, we start with a short overview of OVS and then continue with our evaluation.

5.1 Open vSwitch Overview

Virtual switching is a key building block in NFV environments, as it enables interconnecting multiple *Virtual Network Functions* (VNFs) in service chains and enables the use of other routing technologies such as SDN. In practice, virtual switches rely on sophisticated optimizations to cope with the line rate.

Specifically, we target the DPDK version of OVS that enables the entire packet processing to be performed in user space. It mitigates overheads such as interrupts required to move from user space to



(a) SanJose14 - 1D Bytes

(b) SanJose14 - 1D Bits

(c) SanJose14 - 2D Bytes

(d) Chicago16 - 1D Bytes

(e) Chicago16 - 1D Bits

(f) Chicago16 - 2D Bytes

Figure 4: False Positive Rate for different stream lengths.

kernel space. In addition, DPDK enables user space packet processing and provides direct access to NIC buffers without unnecessary memory copy. The DPDK library received significant engagement from the NFV industry [1].

The architectural design of OVS is composed of two main components: ovs-vswitchd and ovsdb-server. Due to space constraints, we only describe the vswitchd component. The interested reader is referred to [39] for additional information. The DPDK-version of the vswitchd module implements control and data planes in user space. Network packets ingress the datapath (dpif or dpif-netdev) either from a physical port connected to the physical NIC or from a virtual port connected to a remote host (e.g., a VNF). The datapath then parses the headers and determines the set of actions to be applied (e.g., forwarding or rewrite a specific header).

5.2 Open vSwitch Evaluation

We examined two integration methods: First, HHH measurement can be performed as part of the OVS dataplane. That is, OVS updates each packet as part of its processing stage. Second, HHH measurement can be performed in a separate virtual machine. In that case, OVS forwards the relevant traffic to the virtual machine. When RHHH operates with $V > H$, we only forward the sampled packets and thus reduce overheads.

5.2.1 OVS Environment Setup

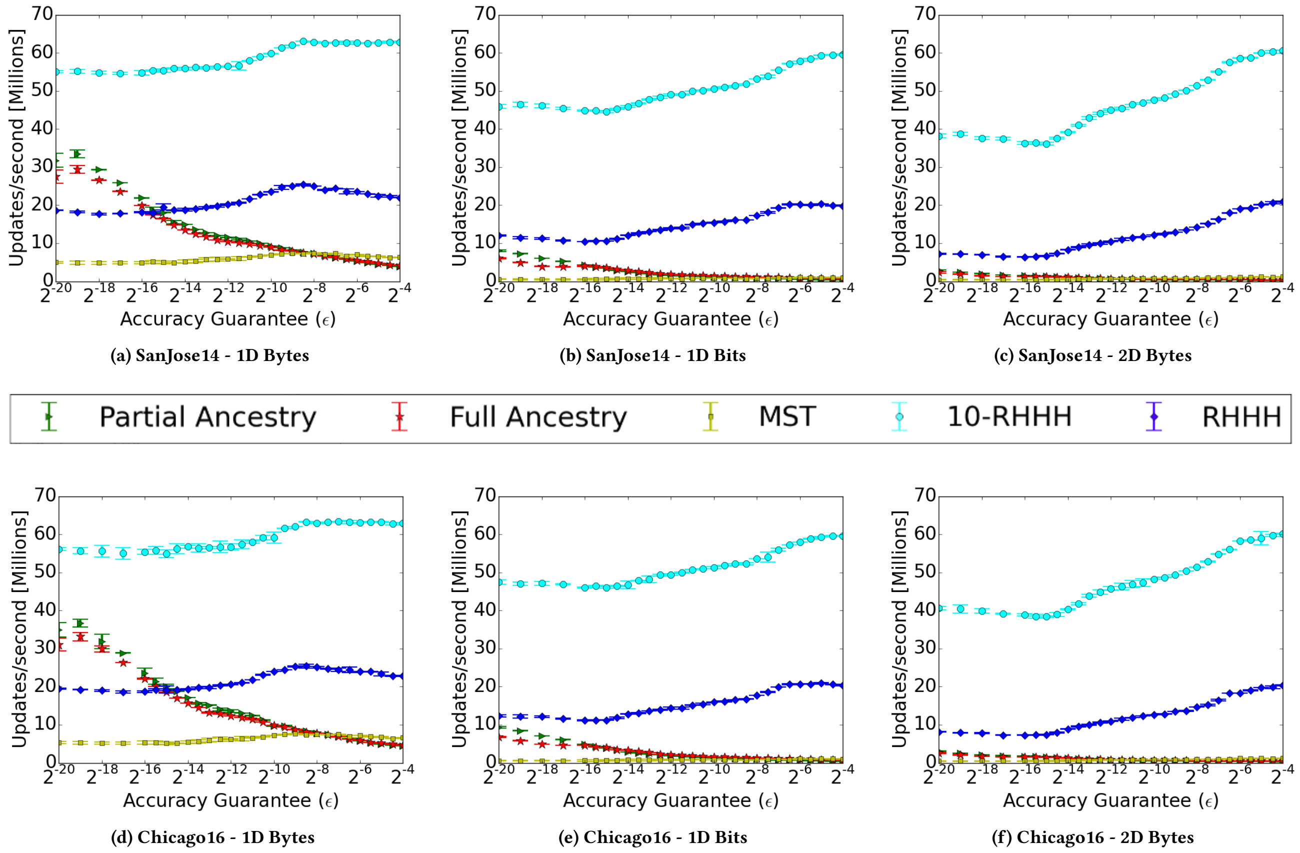
Our evaluation settings consist of two identical HP ProLiant servers with an Intel Xeon E3-1220v2 processor running at 3.1 Ghz with 8 GB RAM, an Intel 82599ES 10 Gbit/s network card and CentOS 7.2.1511 with Linux kernel 3.10.0 operating system. The servers are directly connected through two physical interfaces. We used 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 *Design Under Test (DUT)*. Placed on the DUT, OVS receives packets on one network interface and then forwards them to the second one. Traffic is generated using MoonGen traffic generator [21], and we generate 1 billion UDP packets but preserve the source and destination IP as in the original dataset. We also adjust the payload size to 64 bytes and reach 14.88 million packets per second (Mpps).

5.2.2 OVS Throughput Evaluation

Figure 6 exhibits the throughput of OVS for dataplane implementations. It includes our own 10-RHHH (with V=10H) and RHHH (with V=H), as well as MST and Partial Ancestry. Since we only have 10 Gbit/s links, the maximum achievable packet rate is 14.88 Mpps.

As can be seen, 10-RHHH processes 13.8 Mpps, only 4% lower than unmodified OVS. RHHH achieves 10.6 Mpps, while the fastest competition is Partial Ancestry that delivers 5.6 Mpps. Note that a



(a) SanJose14 - 1D Bytes
(b) SanJose14 - 1D Bits
(c) SanJose14 - 2D Bytes
(d) Chicago16 - 1D Bytes
(e) Chicago16 - 1D Bits
(f) Chicago16 - 2D Bytes

Figure 5: Update speed comparison for different hierarchical structures and workloads

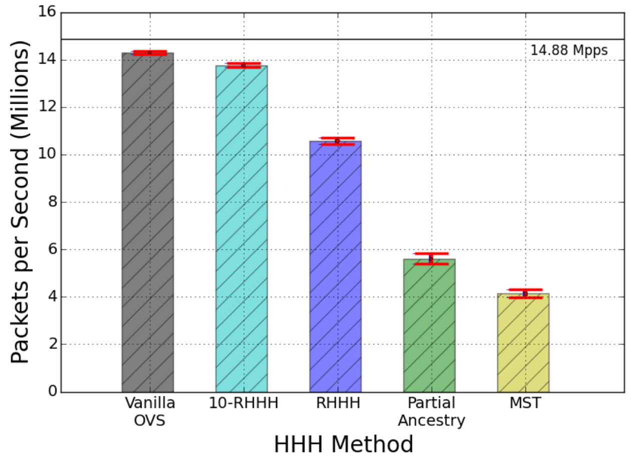


Figure 6: Throughput of dataplane implementations (ϵ = 0.001, δ = 0.001, 2D Bytes, Chicago 16).

100 Gbit/s link delivering packets whose average size is 1KB only delivers ≈ 8.33 Mpps. Thus, 10-RHHH and RHHH can cope with the line speed.

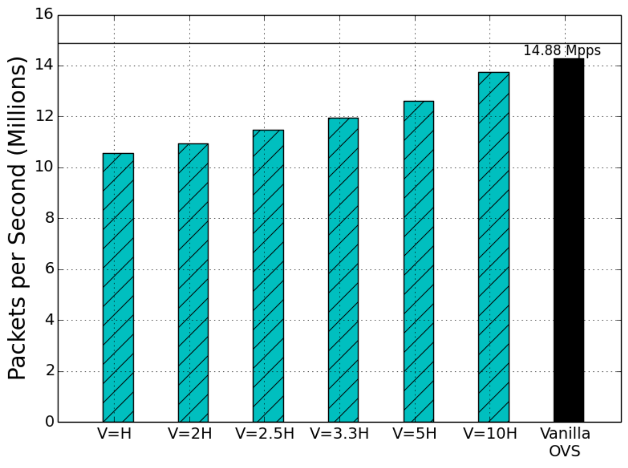
In Figure 7, we evaluate the throughput for different V values, from $V = H = 25$ (RHHH) to $V = 10 \cdot H = 250$ (10-RHHH). Figure 7a evaluates the dataplane implementation while Figure 7b evaluates the distributed implementation. In both figures, performance improves for larger V value. In the distributed implementation, this speedup means that fewer packets are forwarded to the VM whereas in the dataplane implementation, it is linked to fewer processed packets.

Note that while the distributed implementation is somewhat slower, it enables the measurement machine to process traffic from multiple sources.

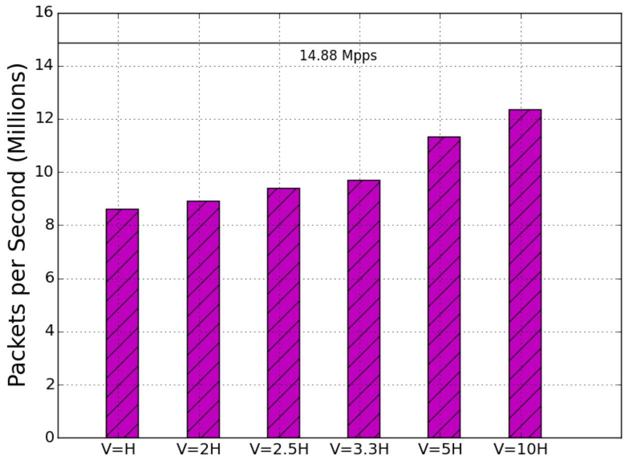
6 ANALYSIS

This section aims to prove that RHHH solves the $(\delta, \epsilon, \theta)-$APPROXIMATE HHH problem (Definition 3.10) for one and two dimensional hierarchies. Toward that end, Section 6.1 proves the accuracy requirement while Section 6.2 proves coverage. Section 6.3 proves that RHHH solves the $(\delta, \epsilon, \theta)-$APPROXIMATE HHH problem as well as its memory and update complexity.

We model the update procedure of RHHH as a balls and bins experiment where there are V bins and N balls. Prior to each packet



(a) Dataplane implementation



(b) Distributed implementation

Figure 7: Measured throughput in both dataplane and distributed implementations.

arrival, we place the ball in a bin that is selected uniformly at random. The first H bins contain an HH update action while the next $V-H$ bins are void. When a ball is assigned to a bin, we either update the underlying HH algorithm with a prefix obtained from the packet's headers or ignore the packet if the bin is void. Our first goal is to derive confidence intervals around the number of balls in a bin.

Definition 6.1. We define X_i^K to be the random variable representing the number of balls from set K in bin i, e.g., K can be all packets that share a certain prefix, or a combination of multiple prefixes with a certain characteristic. When the set K contains all packets, we use the notation X_i.

Random variables representing the number of balls in a bin are dependent on each other. Therefore, we cannot apply common methods to create confidence intervals. Formally, the dependence is manifested as:
$\sum_1^V X_i = N$. This means that the number of balls in a certain bin is determined by the number of balls in all other bins.

Our approach is to approximate the balls and bins experiment with the corresponding Poisson one. That is, analyze the Poisson case and derive confidence intervals and then use Lemma 6.2 to derive a (weaker) result for the original balls and bins case.

We now formally define the corresponding Poisson model. Let $Y_1^K, ..., Y_V^K$ s.t. $\{Y_i^K\} \sim Poisson\left(\frac{K}{V}\right)$ be **independent** Poisson random variables representing the number of balls in each bin from a set of balls K. That is: $\{Y_i^K\} \sim Poisson\left(\frac{K}{V}\right)$.

Lemma 6.2 (Corollary 5.11, page 103 of [36]). *Let $\mathfrak{E}$ be an event whose probability is either monotonically increasing or decreasing with the number of balls. If $\mathfrak{E}$ has probability p in the Poisson case then $\mathfrak{E}$ has probability at most $2p$ in the exact case.*

6.1 Accuracy Analysis

We now tackle the accuracy requirement from Definition 3.10. That is, for every HHH prefix (p), we need to prove:

$$\Pr\left(\left|f_p - \widehat{f_p}\right| \le \varepsilon N\right) \ge 1 - \delta.$$

In RHHH, there are two distinct origins of error. Some of the error comes from fluctuations in the number of balls per bin while the approximate HH algorithm is another source of error.

We start by quantifying the balls and bins error. Let Y_i^p be the Poisson variable corresponding to prefix p. That is, the set p contains all packets that are generalized by prefix p. Recall that f_p is the number of packets generalized by p and therefore: $E(Y_i^p) = \frac{f_p}{V}$.

We need to show that with probability $1-\delta_s$, Y_i^p is within $\epsilon_s N$ from $E(Y_i^p)$. Fortunately, confidence intervals for Poisson variables are a well studied [38] and we use the method of [40] that is quoted in Lemma 6.3.

Lemma 6.3. *Let X be a Poisson random variable, then*

$$\Pr\left(|X - E(X)| \ge Z_{1-\delta}\sqrt{E(X)}\right) \le \delta,$$

where Z_α is the z value that satisfies $\phi(z) = \alpha$ and $\phi(z)$ is the density function of the normal distribution with mean 0 and standard deviation of 1.

Lemma 6.3, provides us with a confidence interval for Poisson variables, and enables us to tackle the main accuracy result.

Theorem 6.4. *If $N \ge Z_{1-\frac{\delta_s}{2}} V \varepsilon_s{}^{-2}$ then*

$$\Pr\left(\left|X_i{}^p H - f_p\right| \ge \varepsilon_s N\right) \le \delta_s.$$

Proof. We use Lemma 6.3 for $\frac{\delta_s}{2}$ and get:

$$\Pr\left(\left|Y_i{}^p - \frac{f_p}{V}\right| \ge Z_{1-\frac{\delta_s}{2}}\sqrt{\frac{f_p}{V}}\right) \le \frac{\delta_s}{2}.$$

To make this useful, we trivially bind $f_p \le N$ and get

$$\Pr\left(\left|Y_i{}^p - \frac{f_p}{V}\right| \ge Z_{1-\frac{\delta_s}{2}}\sqrt{\frac{N}{V}}\right) \le \frac{\delta_s}{2}.$$

However, we require error of the form $\frac{\epsilon_s \cdot N}{V}$.

$$\varepsilon_s N V^{-1} \geq Z_{1-\frac{\delta_s}{2}} V^{-0.5} N^{0.5}$$
$$N^{0.5} \geq Z_{1-\frac{\delta_s}{2}} V^{0.5} \varepsilon_s{}^{-1}$$
$$N \geq Z_{1-\frac{\delta_s}{2}} V \varepsilon_s{}^{-2}.$$

Therefore, when $N \geq Z_{1-\frac{\delta_s}{2}} V \varepsilon_s{}^{-2}$, we have that:

$$\Pr\left(\left|Y_i{}^p - \frac{f_p}{V}\right| \geq \frac{\varepsilon_s N}{V}\right) \leq \frac{\delta_s}{2}.$$

We multiply by V and get:

$$\Pr\left(\left|Y_i{}^p V - f_p\right| \geq \varepsilon_s N\right) \leq \frac{\delta_s}{2}.$$

Finally, since Y_i^p is monotonically increasing with the number of balls (f_p), we apply Lemma 6.2 to conclude that

$$\Pr\left(\left|X_i{}^p V - f_p\right| \geq \varepsilon_s N\right) \leq \delta_s.$$

□

To reduce clutter, we denote $\psi \triangleq Z_{1-\frac{\delta_s}{2}} V \varepsilon_s{}^{-2}$. Theorem 6.4 proves that the desired sample accuracy is achieved once $N > \psi$.

It is sometimes useful to know what happens when $N < \psi$. For this case, we have Corollary 6.5, which is easily derived from Theorem 6.4. We use the notation $\varepsilon_s(N)$ to define the actual sampling error after N packets. Thus, it assures us that when $N < \psi$, $\varepsilon_s(N) > \varepsilon_s$. It also shows that $\varepsilon_s(N) < \varepsilon_s$ when $N > \psi$. Another application of Corollary 6.5 is that given a measurement interval N, we can derive a value for ε_s that assures correctness. For simplicity, we continue with the notion of ε_s.

Corollary 6.5. $\varepsilon_s(N) \geq \sqrt{\frac{Z_{1-\frac{\delta_s}{2}} V}{N}}$.

The error of approximate HH algorithms is proportional to the number of updates. Therefore, our next step is to provide a bound on the number of updates of an arbitrary HH algorithm. Given such a bound, we configure the algorithm to compensate so that the accumulated error remains within the guarantee even if the number of updates is larger than average.

Corollary 6.6. *Consider the number of updates for a certain lattice node (X_i). If $N > \psi$, then*

$$\Pr\left(X_i \leq \frac{N}{V}(1+\varepsilon_s)\right) \geq 1-\delta_s.$$

Proof. We use Theorem 6.4 and get:
$\Pr\left(\left|X_i - \frac{N}{V}\right| \geq \varepsilon_s N\right) \leq \delta_s$. This implies that:
$\Pr\left(X_i \leq \frac{N}{V}(1+\varepsilon_s)\right) \geq 1-\delta_s$, completing the proof. □

We explain now how to configure our algorithm to defend against situations in which a given approximate HH algorithm might get too many updates, a phenomenon we call *over sample*. Corollary 6.6 bounds the probability for such an occurrence, and hence we can slightly increase the accuracy so that in the case of an over sample, we are still within the desired limit. We use an algorithm ($\mathbb{A}$) that solves the $(\varepsilon_a, \delta_a)$ - Frequency Estimation problem. We define $\varepsilon_a' \triangleq \frac{\varepsilon_a}{1+\varepsilon_s}$. According to Corollary 6.6, with probability $1-\delta_s$, the number of sampled packets is at most $(1+\varepsilon_s)\frac{N}{V}$. By using the union bound and with probability $1-\delta_a-\delta_s$ we get:

$$\left|X^p - \widehat{X^p}\right| \leq \varepsilon_{a'}(1+\varepsilon_s)\frac{N}{V} = \frac{\varepsilon_a(1+\varepsilon_s)}{1+\varepsilon_s}\frac{N}{V} = \varepsilon_a\frac{N}{V}.$$

For example, Space Saving requires 1,000 counters for $\epsilon_a = 0.001$. If we set $\epsilon_s = 0.001$, we now require 1001 counters. Hereafter, we assume that the algorithm is configured to accommodate these over samples.

Theorem 6.7. *Consider an algorithm ($\mathbb{A}$) that solves the (ϵ_a, δ_a) - Frequency Estimation problem. If $N > \psi$, then for $\delta \geq \delta_a + 2\cdot\delta_s$ and $\epsilon \geq \epsilon_a + \epsilon_s$, $\mathbb{A}$ solves (ϵ, δ) - Frequency Estimation.*

Proof. As $N > \psi$, we use Theorem 6.4. That is, the input solves (ϵ, δ) - Frequency Estimation.

$$\Pr\left[\left|f_p - X_p V\right| \geq \varepsilon_s N\right] \leq \delta_s. \tag{1}$$

$\mathbb{A}$ solves the (ϵ_a, δ_a) - Frequency Estimation problem and provides us with an estimator $\widehat{X^p}$ that approximates X^p – the number of updates for prefix p. According to Corollary 6.6:

$$\Pr\left(\left|X^p - \widehat{X^p}\right| \leq \frac{\varepsilon_a N}{V}\right) \geq 1-\delta_a-\delta_s,$$

and multiplying both sides by V gives us:

$$\Pr\left(\left|X^p V - \widehat{X^p}V\right| \geq \varepsilon_a N\right) \leq \delta_a + \delta_s. \tag{2}$$

We need to prove that: $\Pr\left(\left|f_p - \widehat{X^p}V\right| \leq \varepsilon N\right) \geq 1-\delta$. Recall that: $f_p = E(X^p)V$ and that $\widehat{f_p} = \widehat{X^p}V$ is the estimated frequency of p. Thus,

$$\begin{aligned} &\Pr\left(\left|f_p - \widehat{f_p}\right| \geq \varepsilon N\right) = \Pr\left(\left|f_p - \widehat{X^p}V\right| \geq \varepsilon N\right) \\ &= \Pr\left(\left|f_p + (X^p V - X^p V) - V\widehat{X^p}\right| \geq (\epsilon_a + \epsilon_s)N\right) \\ &\leq \Pr\left(\left[\left|f_p - X^p V\right| \geq \varepsilon_s N\right] \vee \left[\left|X^p V - \widehat{X^p}V\right| \geq \varepsilon_a N\right]\right), \end{aligned} \tag{3}$$

where the last inequality follows from the fact that in order for the error of (3) to exceed ϵN, at least one of the events has to occur. We bound this expression using the Union bound.

$$\begin{aligned} &\Pr\left(\left|f_p - \widehat{f_p}\right| \geq \varepsilon N\right) \leq \\ &\Pr\left(\left|f_p - X^p V\right| \geq \varepsilon_s N\right) + \Pr\left(\left|X^p V - \widehat{X^p}H\right| \geq \varepsilon_a N\right) \\ &\leq \delta_a + 2\delta_s, \end{aligned}$$

where the last inequality is due to equations 1 and 2. □

An immediate observation is that Theorem 6.7 implies accuracy, as it guarantees that with probability $1-\delta$ the estimated frequency of any prefix is within εN of the real frequency while the accuracy requirement only requires it for prefixes that are selected as HHH.

Lemma 6.8. *If $N > \psi$, then Algorithm 1 satisfies the accuracy constraint for $\delta = \delta_a + 2\delta_s$ and $\epsilon = \epsilon_a + \epsilon_s$.*

Proof. The proof follows from Theorem 6.7, as the frequency estimation of a prefix depends on a single HH algorithm. □

Multiple Updates

One might consider how RHHH behaves if instead of updating at most 1 HH instance, we update r independent instances. This implies that we may update the same instance more than once per packet. Such an extension is easy to do and still provides the required guarantees. Intuitively, this variant of the algorithm is what one would get if each packet is duplicated r times. The following corollary shows that this makes RHHH converge r times faster.

Corollary 6.9. *Consider an algorithm similar to RHHH with $V = H$, but for each packet we perform r independent update operations. If $N > \frac{\psi}{r}$, then this algorithm satisfies the accuracy constraint for $\delta = \delta_a + 2\delta_s$ and $\epsilon = \epsilon_a + \epsilon_s$.*

Proof. Observe that the new algorithm is identical to running RHHH on a stream ($\mathcal{S}'$) where each packet in $\mathcal{S}$ is replaced by r consecutive packets. Thus, Lemma 6.8 guarantees that accuracy is achieved for $\mathcal{S}'$ after ψ packets are processed. That is, it is achieved for the original stream ($\mathcal{S}$) after $N > \frac{\psi}{r}$ packets. □

6.2 Coverage Analysis

Our goal is to prove the coverage property of Definition 3.10. That is: $\Pr\left(\widehat{C_{q|P}} \geq C_{q|P}\right) \geq 1-\delta$. Conditioned frequencies are calculated in a different manner for one and two dimensions. Thus, Section 6.2.1 deals with one dimension and Section 6.2.2 with two.

We now present a common definition of the best generalized prefixes in a set.

Definition 6.10 (Best generalization). Define $G(q|P)$ as the set $\{p : p \in P, p \prec q, \neg\exists p' \in P : q \prec p' \prec p\}$. Intuitively, $G(q|P)$ is the set of prefixes that are best generalized by q. That is, q does not generalize any prefix that generalizes one of the prefixes in $G(q|P)$.

6.2.1 One Dimension

We use the following lemma for bounding the error of our conditioned count estimates.

Lemma 6.11. *([35]) In one dimension,*

$$C_{q|P} = f_q - \sum\nolimits_{h \in G(q|P)} f_h.$$

Using Lemma 6.11, it is easier to establish that the conditioned frequency estimates calculated by Algorithm 1 are conservative.

Lemma 6.12. *The conditioned frequency estimation of Algorithm 1 is:*

$$\widehat{C_{q|P}} = \widehat{f_q}^{+} - \sum\nolimits_{h \in G(q|P)} \widehat{f_h}^{-} + 2Z_{1-\delta}\sqrt{NV}.$$

Proof. Looking at Line 12 in Algorithm 1, we get that:

$$\widehat{C_{q|P}} = \widehat{f_q}^{+} + calcPred(q, P).$$

That is, we need to verify that the return value $calcPred(q, P)$ in one dimension (Algorithm 2) is $\sum_{h \in G(q|P)} \widehat{f_h}^{-}$. This follows naturally from that algorithm. Finally, the addition of $2Z_{1-\delta}\sqrt{NV}$ is due to line 13. □

In deterministic settings, $\widehat{f_q}^{+} - \sum_{h \in G(q|P)} \widehat{f_h}^{-}$ is a conservative estimate since $\widehat{f_q}^{+} \geq f_q$ and $f_h < \widehat{f_h}^{-}$. In our case, these are only true with regard to the sampled sub-stream and the addition of $2Z_{1-\delta}\sqrt{NV}$ is intended to compensate for the randomized process.

Our goal is to show that $\Pr\left(\widehat{C_{q|P}} > C_{q|P}\right) \geq 1-\delta$. That is, the conditioned frequency estimation of Algorithm 1 is probabilistically conservative.

Theorem 6.13. $\Pr\left(\widehat{C_{q|P}} \geq C_{q|P}\right) \geq 1-\delta$.

Proof. Recall that:

$$\widehat{C_{q|P}} = \widehat{f}_q^{+} - \sum_{h \in G(q|P)} \widehat{f}_h^{-} + 2Z_{1-\frac{\delta}{8}}\sqrt{NV}.$$

We denote by K the set of packets that may affect $\widehat{C_{q|P}}$. We split K into two sets: K^{+} contains the packets that may positively impact $\widehat{C_{q|P}}$ and K^{-} contains the packets that may negatively impact it.

We use K^{+} to estimate the sample error in $\widehat{f_q}$ and K^{-} to estimate the sample error in $\sum\limits_{h \in G(q|P)} \widehat{f}_h^{-}$. The positive part is easy to estimate. In the negative, we do not know exactly how many bins affect the sum. However, we know for sure that there are at most N. We define the random variable Y_{+}^{K} that indicates the number of balls included in the positive sum. We invoke Lemma 6.3 on Y_{+}^{K}. For the negative part, the conditioned frequency is positive so $E\left(Y_{-}^{K}\right)$ is at most $\frac{N}{V}$. Hence, $\Pr\left(\left|Y_{K}^{+} - E\left(Y_{K}^{+}\right)\right| \geq Z_{1-\frac{\delta}{8}}\sqrt{\frac{N}{V}}\right) \leq \frac{\delta}{4}$. Similarly, we use Lemma 6.3 to bound the error of Y_{K}^{-}:

$$\Pr\left(\left|Y_{K}^{-} - E\left(Y_{K}^{-}\right)\right| \geq Z_{1-\frac{\delta}{8}}\sqrt{\frac{N}{V}}\right) \leq \frac{\delta}{4}.$$

Y_{+}^{K} is monotonically increasing with any ball and Y_{K}^{-} is monotonically decreasing with any ball. Therefore, we can apply Lemma 6.2 on each of them and conclude:

$$\begin{aligned}&\Pr\left(\widehat{C_{q|P}} \geq C_{q|P}\right) \leq \\ &2\Pr\left(H\left(Y_{K}^{-} + Y_{K}^{+}\right) \geq VE\left(Y_{K}^{-} + Y_{K}^{+}\right) + 2Z_{1-\frac{\delta}{8}}\sqrt{NV}\right) \\ &\leq 1 - 2\frac{\delta}{2} = 1-\delta.\end{aligned}$$

□

Theorem 6.14. *If $N > \psi$, Algorithm 1 solves the $(\delta, \varepsilon, \theta)$ - Approximate HHH problem for $\delta = \delta_a + 2\delta_s$ and $\varepsilon = \varepsilon_s + \varepsilon_a$.*

Proof. We need to show that the accuracy and coverage guarantees hold. Accuracy follows from Lemma 6.8 and coverage follows from Theorem 6.13 that implies that for every non heavy hitter prefix (q), $\widehat{C_{q|P}} < \theta N$ and thus:

$$\Pr\left(C_{q|P} < \theta N\right) \geq 1-\delta.$$

□

6.2.2 Two Dimensions

Conditioned frequency is calculated differently for two dimensions, as we use inclusion/exclusion principles and we need to show that these calculations are sound too. We start by stating the following lemma:

Lemma 6.15. *([35]) In two dimensions,*

$$C_{q|P} = f_q - \sum_{h \in G(q|P)} f_h + \sum_{h, h' \in G(q|P)} f_{\mathrm{glb}(h, h')}.$$

In contrast, Algorithm 1 estimates the conditioned frequency as:

Lemma 6.16. *In two dimensions, Algorithm 1 calculates conditioned frequency in the following manner:*

$$\widehat{C_{q|P}} = \hat{f}_q^+ - \sum_{h \in G(q|P)} \hat{f}_h^- + \sum_{h,h' \in G(q|P)} \hat{f}^+_{\text{glb}(h,h')} + 2Z_{1-\frac{\delta}{8}}\sqrt{NV}.$$

Proof. The proof follows from Algorithm 1. Line 12 is responsible for the first element $\widehat{f_q^+}$ while Line 13 is responsible for the last element. The rest is due to the function calcPredecessors in Algorithm 3. □

Theorem 6.17. $\Pr\left(\widehat{C_{q|P}} \geq C_{q|P}\right) \geq 1-\delta.$

Proof. Observe Lemma 6.15 and notice that in deterministic settings, as shown in [35],

$$\widehat{f_q^+} - \sum_{h \in G(q|P)} \widehat{f_h^-} + \sum_{h,h' \in G(q|P)} \widehat{f^+_{\text{glb}(h,h')}}$$

is a conservative estimate for $C_{q|P}$. Therefore, we need to account for the randomization error and verify that with probability $1-\delta$ it is less than $2Z_{1-\frac{\delta}{8}}\sqrt{NV}$.

We denote by K the packets that may affect $C_{q|P}$. Since the expression of $\widehat{C_{q|P}}$ is not monotonic, we split it into two sets: K^+ are packets that affect $\widehat{C_{q|P}}$ positively and K^- affect it negatively. Similarly, we define $\{Y_i^K\}$ to be Poisson random variables that represent how many of the packets of K are in each bin.

We do not know how many bins affect the sum, but we know for sure that there are no more than N balls. We define the random variable Y_+^K that defines the number of packets from K that fell in the corresponding bins to have a positive impact on $\widehat{C_{q|P}}$. Invoking Lemma 6.3 on Y_+^K yields that:

$$\Pr\left(\left|Y_K^+ - E\left(Y_K^+\right)\right| \geq Z_{1-\frac{\delta}{8}}\sqrt{\frac{N}{V}}\right) \leq \frac{\delta}{4}.$$

Similarly, we define Y_K^- to be the number of packets from K that fell into the corresponding buckets to create a negative impact on $\widehat{C_{q|P}}$ and Lemma 6.3 results in:

$$\Pr\left(\left|Y_K^- - E\left(Y_K{}^-\right)\right| \geq Z_{1-\frac{\delta}{8}}\sqrt{\frac{N}{V}}\right) \leq \frac{\delta}{4}.$$

Y_K^+ is monotonically increasing with the number of balls and Y_K^- is monotonically decreasing with the number of balls. We can apply Lemma 6.2 and conclude that:

$$\begin{aligned}&\Pr\left(\widehat{C_{q|P}} \geq C_{q|P}\right) \leq \\ &2\Pr\left(V\left(Y_K^- + Y_K^+\right) \geq \left(VE\left(Y_K^- + Y_K^+\right) + 2Z_{1-\frac{\delta}{8}}\sqrt{NV}\right)\right) \\ &\leq 1 - 2\tfrac{\delta}{2} = 1-\delta,\end{aligned}$$

completing the proof. □

6.2.3 Putting It All Together

We can now prove the coverage property for one and two dimensions.

Corollary 6.18. *If $N > \psi$ then RHHH satisfies coverage. That is, given a prefix $q \notin P$, where P is the set of HHH returned by RHHH,*

$$\Pr\left(C_{q|P} < \theta N\right) > 1-\delta.$$

Proof. The proof follows form Theorem 6.13 in one dimension, or Theorem 6.17 in two, that guarantee that in both cases: $\Pr\left(C_{q|P} < \widehat{C_{q|P}}\right) > 1-\delta$.

The only case where $q \notin P$ is if $\widehat{C_{q|P}} < \theta N$. Otherwise, Algorithm 1 would have added it to P. However, with probability $1-\delta$, $C_{q|P} < \widehat{C_{q|P}}$, and therefore $C_{q|P} < \theta N$ as well. □

6.3 RHHH Properties Analysis

Finally, we can prove the main result of our analysis. It establishes that if the number of packets is large enough, *RHHH* is correct.

Theorem 6.19. *If $N > \psi$, then RHHH solves $(\delta, \epsilon, \theta)$ - Approximate Hierarchical Heavy Hitters.*

Proof. The theorem is proved by combining Lemma 6.8 and Corollary 6.18. □

Note that $\psi \triangleq Z_{1-\frac{\delta_s}{2}} V \varepsilon_s{}^{-2}$ contains the parameter V in it. When the minimal measurement interval is known in advance, the parameter V can be set to satisfy correctness at the end of the measurement. For short measurements, we may need to use $V = H$, while longer measurements justify using $V \gg H$ and achieve better performance. When considering modern line speed and emerging new transmission technologies, this speedup capability is crucial because faster lines deliver more packets in a given amount of time and thus justify a larger value of V for the same measurement interval.

For completeness, we prove the following.

Theorem 6.20. *RHHH's update complexity is $O(1)$.*

Proof. Observe Algorithm 1. For each update, we randomize a number between 0 and $V-1$, which can be done in $O(1)$. Then, if the number is smaller than H, we also update a Space Saving instance, which can be done in $O(1)$ as well [34]. □

Finally, we note that our space requirement is similar to that of [35].

Theorem 6.21. *The space complexity of RHHH is $O\left(\frac{H}{\varepsilon_a}\right)$ flow table entries.*

Proof. RHHH utilizes H separate instances of Space Saving, each using $\frac{1}{\epsilon_a}$ table entries. There are no other space significant data structures. □

7 DISCUSSION

This work is about realizing hierarchical heavy hitters measurement in virtual network devices. Existing HHH algorithms are too slow to cope with current improvements in network technology. Therefore, we define a probabilistic relaxation of the problem and introduce a matching randomized algorithm called RHHH. Our algorithm leverages the massive traffic in modern networks to perform simpler update operations. Intuitively, the algorithm replaces the traditional approach of computing all prefixes for each incoming packets by sampling (if $V > H$) and then choosing one *random* prefix to be updated. While similar convergence guarantees can be derived for the simpler approach of updating all prefixes for each sampled packet, our solution has the clear advantage of processing elements in $O(1)$ worst case time.

We evaluated RHHH on four real Internet packet traces, consisting over 1 billion packets each and achieved a speedup of up to X62 compared to previous works. Additionally, we showed that the solution quality of RHHH is comparable to that of previous work. RHHH performs updates in constant time, an asymptotic improvement from previous works whose complexity is proportional to the hierarchy's size. This is especially important in the two dimensional case as well as for IPv6 traffic that requires larger hierarchies.

Finally, we integrated RHHH into a DPDK enabled Open vSwitch and evaluated its performance as well as the alternative algorithms. We provided a dataplane implementation where HHH measurement is performed as part of the per packet routing tasks. In a dataplane implementation, RHHH is capable of handling up to 13.8 Mpps, 4% less than an unmodified DPDK OVS (that does not perform HHH measurement). We showed a throughput improvement of X2.5 compared to the fastest dataplane implementations of previous works.

Alternatively, we evaluated a distributed implementation where RHHH is realized in a virtual machine that can be deployed in the cloud and the virtual switch only sends the sampled traffic to RHHH. Our distributed implementation can process up to 12.3 Mpps. It is less intrusive to the switch, and offers greater flexibility in virtual machine placement. Most importantly, our distributed implementation is capable of analyzing data from multiple network devices.

Notice the performance improvement gap between our direct implementation – X62, compared to the performance improvement when running over OVS – X2.5. In the case of the OVS experiments, we were running over a 10Gbps link, and were bound by that line speed – the throughput obtained by our implementation was only 4% lower than the unmodified OVS baseline (that does nothing). In contrast, previous works were clearly bounded by their computational overhead. Thus, one can anticipate that once we deploy the OVS implementation on faster links, or in a setting that combines traffic from multiple links, the performance boost compared to previous work will be closer to the improvement we obtained in the direct implementation.

A downside of RHHH is that it requires some minimal number of packets in order to converge to the desired formal accuracy guarantees. In practice, this is a minor limitation as busy links deliver many millions of packets every second. For example, in the settings reported in Section 4.1, RHHH requires up to 100 millions packets to fully converge, yet even after as little as 8 millions packets, the error reduces to around 1%. With a modern switch that can serve 10 million packets per second, this translates into a 10 seconds delay for complete convergence and around 1% error after 1 second. As line rates will continue to improve, these delays would become even shorter accordingly. The code used in this work is open sourced [4]

Acknowledgments. We thank Ori Rottenstreich for his insightful comments and Ohad Eytan for helping with the code release. We would also like to thank the anonymous reviewers and our shepherd, Michael Mitzenmacher, for helping us improve this work.

This work was partially funded by the Israeli Science Foundation grant #1505/16 and the Technion-HPI research school.

Marcelo Caggiani Luizelli is supported by the research fellowship program funded by CNPq (201798/2015-8).

REFERENCES

[1] Intel DPDK, http://dpdk.org/.
[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *ACM SIGCOMM*. 503–514.
[3] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: Minimal Near-optimal Datacenter Transport. *ACM SIGCOMM* (2013), 435–446.
[4] Ran Ben Basat. RHHH code. Available: https://github.com/ranbenbasat/RHHH.
[5] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2016. Heavy Hitters in Streams and Sliding Windows. In *IEEE INFOCOM*.
[6] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2017. Optimal elephant flow detection. In *IEEE INFOCOM*.
[7] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2017. Randomized Admission Policy for Efficient Top-k and Frequency Estimation. In *IEEE INFOCOM*.
[8] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT*.
[9] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *EATCS ICALP*. 693–703.
[10] Graham Cormode and Marios Hadjieleftheriou. 2008. Finding Frequent Items in Data Streams. *VLDB* 1, 2 (Aug. 2008), 1530–1541.
[11] Graham Cormode and Marios Hadjieleftheriou. 2010. Methods for Finding Frequent Items in Data Streams. *J. VLDB* 19, 1 (2010), 3–20.
[12] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. 2003. Finding Hierarchical Heavy Hitters in Data Streams. In *VLDB*. 464–475.
[13] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. 2004. Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-dimensional Data. In *SIGMOD*. 155–166.
[14] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. 2008. Finding Hierarchical Heavy Hitters in Streaming Data. *ACM Trans. Knowl. Discov. Data* 1, 4 (Feb. 2008), 2:1–2:48.
[15] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* (2005), 18.
[16] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: Scaling Flow Management for High-performance Networks. In *ACM SIGCOMM*. 254–265.
[17] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. 2002. Frequency Estimation of Internet Packet Streams with Limited Space. In *EATCS ESA*.
[18] Gil Einziger and Roy Friedman. 2014. TinyLFU: A Highly Efficient Cache Admission Policy. In *Euromicro PDP*. 146–153.
[19] Gil Einziger and Roy Friedman. 2016. Counting with TinyTable: Every Bit Counts!. In *ACM ICDCN*.
[20] Gil Einziger, Marcelo Caggiani Luizelli, and Erez Waisbard. 2017. Constant Time Weighted Frequency Estimation for Virtual Network Functionalities. In *IEEE ICCCN*.
[21] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *ACM IMC*. 275–287.
[22] Pedro Garcia-Teodoro, Jesus E. Diaz-Verdejo, Gabriel Macia-Fernandez, and E. Vazquez. 2009. Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers and Security* (2009), 18–28.
[23] John Hershberger, Nisheeth Shrivastava, Subhash Suri, and Csaba D. Tóth. 2005. Space Complexity of Hierarchical Heavy Hitters in Multi-dimensional Data Streams. In *ACM PODS*. 338–347.
[24] Paul Hick. CAIDA Anonymized 2013 Internet Trace, equinix-sanjose 2013-12-19 13:00-13:05 UTC, Direction B.
[25] Paul Hick. CAIDA Anonymized 2014 Internet Trace, equinix-sanjose 2013-06-19 13:00-13:05 UTC, Direction B.
[26] Paul Hick. CAIDA Anonymized 2015 Internet Trace, equinix-chicago 2015-12-17 13:00-13:05 UTC, Direction A.
[27] Paul Hick. CAIDA Anonymized 2016 Internet Trace, equinix-chicago 2016-02-18 13:00-13:05 UTC, Direction A.
[28] Lavanya Jose and Minlan Yu. 2011. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *USENIX Hot-ICE*.
[29] Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. 2010. AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers. In *IEEE HOTI*. 58–65.
[30] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. 2003. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions Database Systems* 28, 1 (March 2003).
[31] Yuan Lin and Hongyan Liu. 2007. Separator: Sifting Hierarchical Heavy Hitters Accurately from Data Streams. In *ADMA (ADMA)*. 170–182.
[32] Nishad Manerikar and Themis Palpanas. 2009. Frequent Items in Streaming Data: An Experimental Evaluation of the State-of-the-art. *Data Knowl. Eng.* (2009), 415–430.

[33] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate Frequency Counts over Data Streams. In *VLDB*.

[34] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *ICDT*.

[35] M. Mitzenmacher, T. Steinke, and J. Thaler. 2012. Hierarchical Heavy Hitters with the Space Saving Algorithm. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments (ALENEX)*. 160–174.

[36] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA.

[37] S. Muthukrishnan. 2005. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science* 1, 2 (2005), 117–236.

[38] V.V. Patil and H.V. Kulkarni. 2012. Comparison of confidence intervals for the Poisson mean: Some new aspects. *REVSTAT Statistical Journal* 10, 2 (June 2012), 211–227.

[39] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *USENIX NSDI*. 117–130.

[40] Neil C. Schwertman and Ricardo A. Martinez. 1994. Approximate poisson confidence limits. *Communications in Statistics - Theory and Methods* 23, 5 (1994), 1507–1529.

[41] Vyas Sekar, Nick Duffield, Oliver Spatscheck, Jacobus van der Merwe, and Hui Zhang. 2006. LADS: Large-scale Automated DDOS Detection System. In *USENIX ATEC*. 16–16.

[42] 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 (ACM SOSR)*. 164–176.

[43] P. Truong and F. Guillemin. 2009. Identification of heavyweight address prefix pairs in IP traffic. In *ITC*. 1–8.

[44] David P. Woodruff. 2016. New Algorithms for Heavy Hitters in Data Streams (Invited Talk). In *ICDT*.

[45] L. Ying, R. Srikant, and X. Kang. 2015. The Power of Slightly More than One Sample in Randomized Load Balancing. In *IEEE INFOCOM*. 1131–1139.

[46] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. 2004. Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications. In *ACM IMC*. 101–114.