Routing-Oblivious Network-Wide Measurements

Ran Ben-Basat, Gil Einziger, Shir Landau Feibish, Jalil Moraney, Bilal Tayh, Danny Raz

***Abstract*—The recent introduction of SDN allows deploying new centralized network algorithms that dramatically improve network operations. In such algorithms, the centralized controller obtains a network-wide view by merging measurement data from Network Measurement Points (NMPs). A fundamental challenge is that several NMPs may count the same packet, reducing the accuracy of the measurement. Existing solutions circumvent this problem by assuming that each packet traverses a single NMP or that the routing is fixed and known.**

This work suggests novel algorithms for three fundamental network-wide measurement problems without making any assumptions on the topology and routing and without modifying the underlying traffic. Specifically, this work introduces two algorithms for estimating the number of (distinct) packets or byte volume in the measurement, estimating per-flow packet and byte counts, and finding the heavy hitter flows. Our work includes formal accuracy guarantees and an extensive evaluation consisting of the realistic fat-tree topology and three real network traces. Our evaluation shows that our algorithms outperform existing works and provide accurate measurements within reasonable space parameters.

I. Introduction

Network algorithms such as routing, load balancing, quality of service enforcement, anomaly detection, and intrusion detection require knowledge about specific properties of the underlying network traffic [37], [41], [49], [21], [36], [29]. Furthermore, applications such as traffic engineering [21], detecting packet loss [44], [55], and identifying traffic patterns such as super-spreaders and port scans [53] require network-wide measurements. In such measurements [6], [8], [9], [38], [40], [28], [43], [26], [54], [14], [11], a centralized controller collects measurement data from multiple *Network Measurement Points (NMPs)* and merges that data into a coherent network-wide view of the traffic as illustrated in Figure 1.

Monitoring is challenging in multiple levels. First, we need to cope with the rapid line transmission rates within each NMP, and thus many works are optimized toward performing efficient single NMP measurements [46], [27], [33], [16], [17], [32], [18], [23], [51], [13], [12]. Network-wide measurement exposes new challenges such as the NMP placement problem. Namely, when considering the richness of SDN routing algorithms it is desirable to allow packets to (potentially) traverse multiple NMPs without double counting. Other challenges include optimizing the bandwidth between the control and the NMPs [38].

The majority of existing solutions avoid the NMP placement problem by assuming that packets are only monitored within a single NMP [6], [38], or by assuming a restrictive routing model [43]. Placing NMPs so that no packet traverses more than a single NMP requires knowledge regarding the routing and topology of the network. Furthermore, re-transmissions due to packet-loss may cause a single NMP to encounter the same packets twice. Worse yet, technologies such as multicast

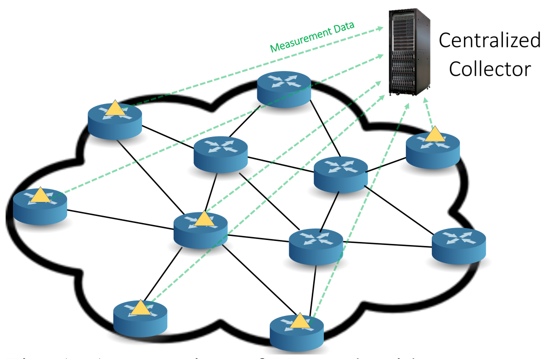


Fig. 1: An overview of Network-wide measurements. NMPs are placed in some network devices, and the centralized controller collects and merges their measurement into a network-wide view. When packets traverse multiple NMPs, they appear in the summaries of multiple NMPs, and the controller needs to avoid double counting them.

may cause a large portion of the traffic to traverse multiple NMPs. FlowRadar [43] assumes that each flow follows a single (and fixed) routing path. Therefore, it is not compatible with Multipath TCP that uses multiple paths to route a TCP flow.

The NMP placement problem was recently studied in [6]. In this work, the authors make use of unused bits in the IP header to mark packets in the first NMP they encounter. Such marking allows for other NMPs to avoid double counting by ignoring marked packets. However, there are significant limitations. First, an attacker can avoid detection by marking the unused bits of its traffic. Further, other entities are free to use the unused bits as they see fit, and thus some of the packets may be transmitted with set unused bits, and be ignored by all the NMPs. Therefore, a practical deployment requires actively clearing the unused bits of all packets that enter the network. Further, for good network citizenship, it is advised to clear the unused bits prior to egressing the network. However, clearing the unused bits once packets enter and leave the network complicates the deployment. Thus, a passive measurement solution that does not modify the network traffic is preferable.

A. Our Contribution

We suggest a novel network-wide algorithm for collecting a uniform sample from a network without double counting, and in a routing and topology oblivious manner. Based on this sample, we present algorithms for there fundamental monitoring problems: (i) estimating the total traffic byte volume or the number of (distinct) packets in the measurement. (ii) estimating per-flow packet or byte count, and (iii) reporting the heavy hitter flows (according to byte or packet count).

Within the more general context of count distinct algorithms, our work improves the state of the art for weighted count distinct algorithms. From another point of view, our work is the first to offer network-wide and routing oblivious algorithms for byte based measurements.

II. Preliminaries and Background

This section aims to grant the reader a brief introduction to the model, and problem definition in Section II-A. Then, Section II-B positions our work within the context of prior works in the field.

A. Model and Problem Definitions

Our data consists of a *stream* of packets $\mathcal{S} \in (\mathcal{U} \times \mathbb{N} \times \mathbb{N})^*$, where each packet $\langle x, i, w\rangle$ is associated with a *flow identifier* $x \in \mathcal{U}$, a *packet identifier* $i \in \mathbb{N}$, and a *weight* $w \in \mathbb{N}$. Flows can be any partition of the traffic to non-overlapping groups (e.g., they can refer to source or destination IP addresses, source-destination IP pairs, or 5-tuples). Packet weight may be the byte size (or the payload size) for byte volume measurements, and can be unit weights for packet based measurements.

We assume that each flow, and each packet have unique identifiers. For flows, 5-tuples are natural identifiers. However, packets have no obvious identifiers. For the TCP protocol, we can use the sequence number as a unique identifier, while for other protocols we can use previously suggested methods [30], [55]. We assume a bound $n \geq \max\{|\mathcal{S}|, |\mathcal{U}|\}$ on the stream size which is known in advance. We use a RAM machine with $\theta(\log n)$-sized words for the complexity analysis.

The network consists of *z Network Measurement Points (NMPs)* $R_1, \dots R_z$, each observes a subsequence of the stream $\mathcal{S}_i \subseteq \mathcal{S}$ such that $\cup_{i=1}^{z} \mathcal{S}_i = \mathcal{S}$. That is, packets may traverse multiple NMPs, but we assume that each packet traverses *at least a single* NMP. This is a more general model than the one studied in previous works [38], [40], [52]. Specifically, in such works no packet traverses multiple NMPs.

The *frequency* of a flow $x \in \mathcal{U}$ is the total weight of all packets with the same flow identifier: $f_x \triangleq \sum_{\langle x,i,w\rangle \in \mathcal{S}} w$. The total weight of all packets in the stream is denoted by $|\mathcal{S}|$. Given a threshold θ, the *heavy hitters* are all the flows whose frequency is larger than $\theta \cdot |\mathcal{S}|$. We summarize the notations used in this paper in Table I.

1) Problem Definitions: We consider the following network-wide problems:

- (ε, δ)-Distributed Volume Estimation: Return an estimator $\widehat{V}$ for the overall number of packets in the network. With probability $1-\delta$, $\widehat{V}$ is a $(1+\varepsilon)$ multiplicative approximation for $|\mathcal{S}|$.
- (ε, δ)-Distributed Frequency Estimation: given a query for flow $x \in \mathcal{U}$, return an estimator $\widehat{f_x}$ that satisfies: $\Pr\left[\left|\widehat{f_x} - f_x\right| > |\mathcal{S}|\varepsilon\right] \leq \delta$.
- (ε, δ)-Distributed Heavy Hitters: Upon a query with parameter θ, return a set $S \subseteq \mathcal{U}$ such that:
  (1) $\Pr\left[\exists x \mid f_x \geq |\mathcal{S}|\theta \wedge x \notin \mathcal{S}\right] \leq \delta$, and
  (2) for all $y \in \mathcal{U}$ such that $f_y < |\mathcal{S}|(\theta - \varepsilon)$ we have that $\Pr\left[y \in \mathcal{S}\right] \leq \delta\varepsilon$.

Symbol	Meaning
$\mathcal{S}$	The packet stream
n	An upper bound on the length of the stream ($n \geq \|\mathcal{S}\|$)
NMP	Network Measurement Point
$\langle x, i\rangle$	A packet from flow x with sequence number i
$\mathcal{U}$	The universe of flow identifiers
f_x	The frequency of flow $x \in \mathcal{U}$
z	The number of NMPs
R_i	Router number i ($i \in \{1, \dots, z\}$)
$\mathcal{S}_i$	The sub-stream seen by router R_i ($\cup_{i\in\{1,\dots,k\}} \mathcal{S}_i = \mathcal{S}$)
$\widehat{V}$	An estimate for $\|\mathcal{S}\|$
ε	The goal error parameter
δ	The goal error probability
$\widehat{f_x}$	An estimate for the frequency of flow x
θ	Heavy hitter threshold
$\mathbb{A}$	A Count Distinct algorithm with an $(\varepsilon_{\mathbb{A}}, \delta_{\mathbb{A}})$ guarantee
$S_{\varepsilon_{\mathbb{A}},\delta_{\mathbb{A}}}$	The space that $\mathbb{A}$ requires
K	Fat-tree parameter.
χ	The sample size required for an (ε, δ) approximation as in Lemma 2 ($\chi = 3\varepsilon^{-2} \log 2\delta^{-1}$)

TABLE I: A list of symbols and notations

B. Background

Here, we provide the technical background necessary to understand our work. In Section II-B1 we survey count distinct algorithms which lay the theoretical foundations to our work, then Section II-B2 surveys the Beta distribution that we use in order to extend our work for byte measurements. Finally, Section II-B3 explains priority sampling which is a sampling method that we adopt to network-wide byte measurements. Understanding how priority sampling works,is necessary to understand our Network-Wide Priority Sampling (NWPS) algorithm.

1) Counting Distinct Items: Our work adapts the K^{th} Minimum Value (KMV) count distinct algorithm of Bar Yossef et. al [10] to solve the (ε, δ)-Distributed Volume Estimation problem in distributed settings. The KMV algorithm is originally suggested for counting the number of distinct items in a stream. KMV assigns a hash value in the range of $[0, 1]$ for each item and maintains the smallest χ hash values. Let p be the largest of the χ smallest hash values. KMV estimates the number of distinct items as $\chi \cdot p^{-1}$. Intuitively, after 1,000 distinct items the tenth smallest hash value is expected to be 0.01. In general, a count distinct algorithm $\mathbb{A}$ provides a $(1 + \varepsilon_{\mathbb{A}})$-approximation of the number of distinct items with probability $\geq 1 - \delta_{\mathbb{A}}$.

Count distinct algorithms support three functions; $\mathbb{A}$.Add$(\cdot)$ that processes items, $\mathbb{A}$.Query$()$ that estimates the number of distinct items, and Merge$(\mathbb{A}_1, \mathbb{A}_2)$ that merges two instances $\mathbb{A}_1$ and $\mathbb{A}_2$ into a unified instance that monitors $S_1 \cup S_2$, where S_1, and S_2 are the streams processed by $\mathbb{A}_1$ and $\mathbb{A}_2$ respectively. This interface is satisfied by the KMV algorithm as well as many other count distinct algorithms [7], [25], [34], [35], [39], [42]. For example, recall that the KMV algorithm calculates a hash value for each item, and maintains the minimal k hash values. A merge operation accepts $2 \cdot k$ hash values (k from each instance), and retains the k minimal hash values (out of the 2k values). Notice that the same values are obtained by a single KMV algorithm that processed $S_1 \cup S_2$, which is a property that we later exploit to achieve unbiased

Problem	Space in Bits	Update Time	Query Time	Theorem #
DISTRIBUTED VOLUME ESTIMATION	$O\left(\varepsilon^{-2} \cdot \log \delta^{-1} \cdot \log n\right)$	$O(1)$ amortized	$O(1)$	-
DISTRIBUTED FREQUENCY ESTIMATION				5
DISTRIBUTED HEAVY HITTERS	$O\left(\varepsilon^{-2} \cdot \log(\varepsilon^{-1}\delta^{-1}) \cdot \log n\right)$		$O(\theta^{-1})$	6

TABLE II: Summary of our results for the UWRA algorithm. n is an upper bound on the number of packets/bytes ($n \geq |\mathcal{S}|$). The amortized constant time of the weighted variants of the DISTRIBUTED FREQUENCY ESTIMATION and DISTRIBUTED HEAVY HITTERS problem holds in expectation once the NMP sees enough packets, as explained in Theorem 3.

uniform samples.

2) The Beta Distribution: Our work also utilizes the Beta distribution to perform weighted (byte count based) network-wide measurements. The Beta distribution is a continuous probability distribution over the real interval $[0, 1]$, that allows us to calculate the minimum of r uniform variables directly without randomizing w variables. Specifically, the minimum of $X_1, \ldots X_r \sim U[0, 1]$ is distributed as $Beta(1, r)$. Fortunately, $Beta(1, r)$ can be efficiently calculated for any $r \geq 1$. Specifically, if U is a uniform variable $[0, 1]$, then $U^{1/r} \sim Beta(1, r)$ [25].

3) Priority Sampling: Priority Sampling (PS) [31] is a known method for sampling weighted items. Our work adapts and extends PS to network-wide measurements. Hence, we survey the method for completeness.

Given a packet $\langle x, i, w\rangle$ of weight w, PS first randomizes a uniform number $\sigma \sim U[0, 1]$, and assigns the packet with a priority of $p \triangleq \frac{w}{\sigma}$. PS maintains a table that stores for each packet: (i) the flow id (x), (ii) the packet id (i) , (iii) the priority (p), and (iv) the weight (w). PS maintains a list of the χ highest priority items. Denote by τ the smallest priority of any packet in the list (the χ-largest priority in the stream). The adjusted weight of a sampled packet $\langle x, i, w\rangle$ is the *maximum* between τ and w, and that of unsampled packets is 0. The adjusted weight is an unbiased estimator to the packets' weight. Intuitively, we always sample packets with weight $w \geq \tau$ (and their adjusted weight does not increase), the probability to sample packets whose weight (w) is smaller than τ, is w/τ and their adjusted weight is τ.

III. NETWORK-WIDE ALGORITHMS

In this section we define the *distributed uniform sampling (DUS)* algorithm for packet based measurements. Then Section III-E suggests two extensions of DUS to byte based measurements.

A. Distributed Measurement

We start by providing a sampling-based solution for packet based measurements (w=1). In a gist, we map each packet to a hashed value which is uniformly distributed in $[0, 1]$, and each NMP stores the χ minimal-hash packets (or their headers) using a heap data structure. Note that when a packet traverses multiple NMPs it receives the same hashed value in each NMP. We keep $O(\log n)$ bits for the hash and identifier, we have that the overall memory at each NMP i: $O\left(\varepsilon^{-2} \cdot \log \delta^{-1} \cdot \log n\right)$. We name this algorithm the *Distributed Uniform Sampling (DUS)* Algorithm [15].

B. Simple Controller Merge Algorithm

At the end of every epoch, each of the z NMPs sends its χ-sized sample to the centralized controller. The controller merges all the samples, and retains the χ globally minimal-hash packets in the entire measurement. Note that the χ globally minimal-hash packets, are the same regardless of the number of NMPs traversed by a single packet. Thus, since hash values are uniformly distributed, the merged sample is unbiased in the sense that each packet $p \in S$ has the same probability to be sampled. Our preliminary work [15] utilizes the Simple merge algorithm. However, the controller receives up to $z \cdot \chi$ samples, and only retains χ samples which implies that it potentially discards useful information. In Section III-C we introduce a better merge algorithm.

C. Improved Controller Merge Algorithm

We adapt the procedure of Cohen and Kaplan [24] for efficient merging to our settings. Their procedure, produces a sample of size $\chi' \geq \chi$ as follows: For $i \in \{1, \ldots, k\}$, let T_i be the χ'th smallest hash value measured in NMP i[1]. Let $T = \min\{T_i \mid i \in \{1, \ldots, k\}\}$ be the minimum across all the T_i's. Then, the controller selects all packets whose hash value is smaller than T. This process yields a merged sample which is larger or equal to χ as T is never smaller than the χ's globally minimal hash value. However, there may be more than χ hash values, which are smaller than T. Thus, the controller attains a larger and more accurate sample. Cohen and Kaplan showed that this merge algorithm is unbiased, which implies (in our case) that the obtained sample is unbiased. Specifically, one can verify that if we denote the size of the collected sample by $\chi' \geq \chi$, then the result of Cohen and Kaplan's algorithm is the same as merging KMV instances of size χ' using the simple merge algorithm.

D. Uniform Sample Analysis

1) Merge Complexity: In our suggestion, the samples are read from all switches at the end of each measurement epoch. As a result, the time for collecting the samples is linear in the number of switches and sample size. The overall aggregation time, which includes the controller merging is also linear in the number of samples received, using a linear-time percentile algorithm [22].

2) Network-wide Volume Estimation: In network-wide settings, it is not trivial to measure the total traffic volume as each NMP observes a substream and require the cardinally of the union of all streams without double counting packets that

[1]If NMP i sees less than χ packets, denote $T_i = 1$.

appear in multiple streams. We provide a controller algorithm to do just that by employing count distinct algorithms. Our solution tracks the overall volume of all distinct packets without assumptions on the network topology, routing, or the packets' order. Thus, we explicitly allow packets to be routed through multiple NMPs.

Once the controller merges the samples (either from the UWRA or from the NWPS algorithms) it can use the merged sample to estimate the volume of the measurement. For the UWRA, we observe that our merged sample can mimic the KMV count distinct algorithm [10] (that only requires the hash values). In NWPS, we estimate the volume to be the sum of the adjusted priority over all the sampled packets.

3) Network-wide Heavy Hitters: We find the heavy hitters by calculating the portion of all sampled flows in the measurement. In UWRA, we calculate their portion in the sample, compare it to the threshold (θ) and report flows whose portion in the sample is at least $\theta - \varepsilon$. In NWPS, we estimate measurement volume (V), and return the flows whose frequency is above the threshold of $V(\theta - \varepsilon)$.

4) Network-wide per flow Frequency Estimation: The above sampling techniques allow one to estimate the frequency of a flow if the effective sampling probability $p = \frac{\chi}{|\mathcal{S}|}$ is known (see Lemma 2). Unfortunately, while $\chi = 3\varepsilon^{-2}\log 2\delta^{-1}$ is known, $|\mathcal{S}|$ is not. Luckily, we can estimate $|\mathcal{S}|$, using the algorithm in Section III-D2. We need to carefully select the error parameters to solve the formal problem. We do so by setting the error parameters of the volume estimation to $\varepsilon_V = \varepsilon/3$ and $\delta_V = \delta/2$, and the parameters of the network-wide sample to $\varepsilon_s = \varepsilon/2$ and $\delta_s = \delta/2$ (i.e., each NMP tracks $\chi = 12\varepsilon^{-2}\log 4\delta^{-1}$ packets), we can solve the problem by returning $\widehat{f_x} \triangleq f'_x \widehat{V}/\chi$.

E. Byte Based Measurements

Byte volume measurements are often desired, and intuitively we can use the DUS algorithm unmodified for byte volume measurement by calculating a hash value for each byte rather than for each packet. Such a method is correct but impractical. Our *Unit Weight Reduction Algorithm (UWRA)* algorithm is a practical implementation of the above intuition as it treats a packet of weight w as if it is w distinct unit weight packets. Here, we leverage the Beta distribution for an efficient implementation of this concept. Alternatively, in *Network-wide Priority Sampling (NWPS)*, we extend the well-known PS method to network-wide settings.

1) Unit Weight Reduction Algorithm (UWRA): The UWRA algorithm treats a packet of weight w as w different packets of weight 1. We need to calculate w hash values for every single packet, which is very inefficient. Instead, we use the Beta distribution to compute the minimum of the w hashes directly and check if it is among the χ smallest entries. If it is among the χ smallest entries, we admit it to the sample and repeat the process by calculating the 2^{nd} smallest hash. Eventually, we discover that the i^{th} smallest hash value is not among the χ smallest and terminate. Intuitively, after seeing enough network traffic, the smallest value is unlikely to be sampled, and the amortized complexity becomes $O(1)$. However, during the start of the measurement, we may need to calculate $\min\{w, \chi\}$ entries to the table resulting in a worst-case per-packet complexity of $O(\min\{w, \chi\}\log\chi)$.

UWRA maintains the same heap data structure as the DUS algorithm, and each of the χ entries contains: (i) Packet identifier, (ii) flow identifier, and (iii) hash value. Our implementation requires a *Pseudo Random Number Generator (PRNG)* [50] to employ the Beta distribution. Intuitively, a PRNG is a deterministic mapping $\{0,1\}^\ell \to \{0,1\}^n$, for $\ell \ll n$, that computes an n-bit near random string from an ℓ-bit *seed*.

When processing packet $\langle x, i, w\rangle$, we initialize a PRNG with seed $\langle x, i\rangle$ and generate $\rho_0 \sim Beta(1, w)$. As explained in Section II-B2, the distribution of ρ_0 is identical to that of $\min\{h(\langle x,i,1\rangle), h(\langle x,i,2\rangle), \ldots, h(\langle x,i,w\rangle)\}$; using $\langle x, i\rangle$ as a seed for the PRNG ensures consistency and that the same ρ_0 value is computed by all NMPs that process the packet.

The next step is to check if ρ_0 is smaller than one of the χ stored elements. If not, then none of $\{h(\langle x,i,1\rangle), h(\langle x,i,2\rangle), \ldots, h(\langle x,i,w\rangle)\}$ are among the χ bytes with the minimal hash value and the processing terminates. Otherwise, we add $\langle x, i\rangle$ into the χ-sized heap with a hash value of ρ_0. We continue the process as multiple bytes from the packet may be among the minimal hash values. Next, we generate $\rho_1 \triangleq \rho_0 + (1-\rho_0)\cdot Beta(1, w-1)$. As discussed in Section IV, ρ_1 is distributed as the *second smallest* in $\{h(\langle x,i,1\rangle), h(\langle x,i,2\rangle), \ldots, h(\langle x,i,w\rangle)\}$, *conditioned on the minimum being* ρ_0. The algorithm checks whether ρ_1 should be inserted to the heap and if so proceeds. Intuitively, we reduced the packet processing time to $O(Z\log\chi)$, where Z is a random variable that denotes how many updates were performed. This result follows immediately as we generate $Z+1$ Beta variables, each of which requires $O(1)$ time (see Section II-B2), and make $O(Z)$ updates to the heap data structure. The pseudo-code of UWRA appears in Algorithm 1. As the algorithm simulates the unweighted algorithm precisely, we conclude its correctness and accuracy guarantees.

Algorithm 1 The UWRA algorithm

```
Initialization, at each router r: $M_r \leftarrow$ ($\chi$-sized max heap).$init()$
1: function ADD$_r$(Item $x$, sequence number $i$, weight $w$)
                                                   ▷ Runs at router $R_r$
2:     $P \leftarrow PRNG(\langle x,i\rangle)$        ▷ Initialize a PRNG with seed $\langle x,i\rangle$
3:     $\rho_0 \leftarrow P.generateBeta(1, w)$
4:     $i \leftarrow 0$
5:     while $(\rho_i < M_r.\max) \wedge (i < \min\{w,\chi\})$ do
6:         $M_r$.insert($\rho_i, \langle x,i\rangle$)      ▷ Insert the key $\rho_i$, delete the max
7:         $i \leftarrow i+1$
8:         $\rho_i \leftarrow \rho_{i-1} + (1-\rho_{i-1})\cdot P.generateBeta(1, w-i)$
```

Algorithm 2 The NWPS algorithm

```
Initialization, at each router r: $M_r \leftarrow$ ($\chi$-sized max heap).$init()$
1: function ADD$_r$(Item $x$, sequence number $i$, weight $w$)
                                                   ▷ Runs at router $R_r$
2:     $\sigma \leftarrow h(\langle x,i\rangle)$
3:     $p \leftarrow w/\sigma$
4:     if $(\rho_i < M_r.\max)$ then
5:         $M_r$.insert($p, \langle x,i,w\rangle$)     ▷ Insert the key $p$, delete the max
```

Our analysis shows that after seeing enough traffic, the

NMP reaches a constant amortized per-packet update time. Here, "enough" logarithmically depends on the average packet size in the measurement. UWRA does not assume the average packet size, and its value is only needed to analyze the complexity. In practice, packet sizes are bounded by the communication protocols (e.g., TCP/UDP packets' size cannot exceed 64KB). Further, a recent analysis of backbone and datacenter traces shows that the average packet size is $800 - 1500$ bytes [16].

2) Network-wide Priority Sampling (NWPS): Our next algorithm *Network-wide Priority Sampling (NWPS)* extends Priority Sampling [31] to network-wide measurements. In contrast to UWRA, we only require a single hash calculation per packet. Specifically, we maintain a χ sized table where each entry stores the packet id, weight, and priority.

As before, we use the same randomization to determine priority in all NMPs. That is, the same packet receives the same priority in all NMPs. Note that each packet takes at most a single place in the table and that the worst-case update complexity of NWPS is $O(\log(\chi))$ for maintaining a χ sized heap ordered according to priority. However, each entry in NWPS is slightly larger than in UWRA, as we also store weights in the table. Thus, for the same memory constraint, the UWRA algorithm can have slightly more entries than NWPS.

IV. Analysis

In this section, we provide the needed analysis and the proofs for the correctness of our algorithms.

We now analyze NWPS (Algorithm 2), which runs in constant worst case time per packet. Duffield et al. have shown that Priority Sampling is unbiased, in the sense that the expected corrected weight for each packet equals its original weight [31]. Observe that by using hash functions, the sample collected by the controller is (distributionally) equivalent to one reached by running Priority Sampling on $\mathcal{S}$, and therefore we remain unbiased. Recall that a flow size is defined as the sum of weights of its packets, and the overall stream volume is the sum of all packets. We can then use the linearity of expectation to conclude the following.

Theorem 1. *NWPS provides unbiased estimators to flow sizes and the overall volume.*

A. Complexity Analysis for UWRA

We now analyze the amortized complexity of UWRA. Intuitively, as it samples fewer and fewer bytes as the measurement progresses, its amortized processing time eventually becomes constant. We start with examining the number of updates made in UWRA for a given stream size $|\mathcal{S}|$. Later, we show that this allows us to prove a bound on the amortized processing time per-packet of our distributed weight sampling technique.

Theorem 2. *The expected number of updates made in UWRA by NMP (R_r) is bounded by $O(\chi \log(|\mathcal{S}_r|/\chi))$.*

Proof: Observe that the i'th item/byte (x_i) is added with a probability $\min\{1, \chi/i\}$. We denote this event by E_i. Thus, the expected number of updates to our algorithm is bounded by

$$\sum_{i \le |\mathcal{S}_r|} \Pr[E_i] \le \sum_{i \le |\mathcal{S}_r|} \min\{1, \chi/i\} \le \sum_{i=1}^{\chi} 1 + \sum_{\chi}^{|\mathcal{S}_r|} \chi/i$$
$$\le \chi(1 + \ln(|\mathcal{S}_r|/\chi) + O(1)) = O(\chi \log(|\mathcal{S}_r|/\chi)),$$

where the second equality follows from the fact that $H_z \triangleq \sum_{i=1}^{z} 1/i = \ln(z) + \Theta(1)$ is a known bound for the i'th harmonic number. ■

We now use the above lemma to show that if the number of packets/bytes processed by an NMP is large enough then its amortized update time drops to a constant.

Theorem 3. *Once an NMP (R_r) processes $n_r = \Omega(\chi \log \chi \max\{\log \chi, \log(A_r/\chi)\})$ packets, where $A_r = |\mathcal{S}_r|/n_r$ is the average packet weight, then its expected amortized processing time is $O(1)$ per packet.*

Proof: First, let us assume that

$$n_r \ge c\chi \log \chi \max\{\log \chi, \log(A_r/\chi)\},$$

for some $c = \Omega(1)$. According to Theorem 2, the expected number of bytes that are sampled from its $|\mathcal{S}_r|$-byte sub-stream is $O(\chi \log(|\mathcal{S}_r|/\chi))$. Each sampled byte incurs a $O(\log \chi)$ time for updating the heap. This means that the expected amortized processing time is:

$$\frac{\chi \log \chi \log(|\mathcal{S}_r|/\chi)}{n_r} = \frac{\chi \log \chi \log(n_r A_r/\chi)}{n_r}$$
$$= \frac{\chi \log \chi \log(n_r)}{n_r} + \frac{\chi \log \chi \log(A_r/\chi)}{n_r}$$
$$\le \frac{\chi \log \chi \log\left(c\chi \log^2 \chi\right)}{c\chi \log^2 \chi} + \frac{\chi \log \chi \log(A_r/\chi)}{c\chi \log \chi \log(A_r/\chi)}$$
$$= \frac{\log\left(c\chi \log^2 \chi\right)}{c \log \chi} + \frac{1}{c} = \frac{1}{c}\left(2 + \frac{\log c + \log\log \chi}{\log \chi}\right) = O(1).$$

■

B. Correctness analysis of UWRA

In this subsection, we show that the packets collected by UWRA form a uniform sample. We start with a lemma, due to Lurie and Hartley [45], that shows that our generation of ρ values in Algorithm 1 has the same distribution as $h(\langle x, i, 1\rangle), h(\langle x, i, 2\rangle), \ldots, h(\langle x, i, w\rangle)$, but they are generated in a descending order.

Lemma 1. *([45]) Let $X_1, \ldots, X_n$ be independent uniform variables. For $i \in \{1, \ldots, n\}$, let $X_{(i)}$ be the i'th order statistic – the i'th smallest variable among $X_1, \ldots, X_n$. Denote by $B_0, \ldots, B_{n-1}$ be independent Beta variables such that $B_i \sim Beta(n-i, 1)$. Let $\psi_i = \prod_{j=0}^{i} B_j$ for all $i \in \{0, 1, \ldots, n-1\}$. Then $\langle \psi_0, \ldots, \psi_{n-1}\rangle \overset{d}{=} \langle X_{(n)}, X_{(n-1)}, \ldots, X_{(1)}\rangle$.*

We can then use symmetry to reverse the process. That is, By setting $B_i' \sim Beta(1, n-i)$, $\rho_0 \triangleq B_0'$, and $\rho_i \triangleq \rho_{i-1} + (1 - \rho_{i-1}) \cdot B_i'$ we get that $\langle \rho_0, \ldots, \rho_{n-1}\rangle \overset{d}{=} \langle X_{(1)}, X_{(2)}, \ldots, X_{(n)}\rangle$. An equivalent algorithm can use the lemma directly by generating the ψ sequence instead of ρ and keeping a min-heap instead of a max heap.

We now claim that UWRA is the same as an hypothetical algorithm that adds w unit weight packets instead of each w-sized packet.

Theorem 4. *For any $\mathcal{S}$ and $\mathcal{S}_1, \dots, \mathcal{S}_r$ such that $\cup_{i=1}^{r} \mathcal{S}_i = \mathcal{S}$, Algorithm 1 allows the controller to attain a (weighted) uniform sample.*

Proof: The proof is by induction on $|\mathcal{S}|$. When $\mathcal{S} = \emptyset$ clearly no packet is sampled. Let $\mathcal{S}$ be a packet stream and let $\langle x, i, w \rangle$ be a new packet that is processed at a non-empty subset of $\mathcal{S}_1, \dots, \mathcal{S}_r$. Then UWRA computes $h(\langle x, i, 1\rangle), h(\langle x, i, 2\rangle), \dots, h(\langle x, i, w\rangle)$ at all NMPs that process the packet. Since for every packet all NMPs use a PRNG with the same seed, the process is equivalent to that obtained by using the unweighted algorithm and replacing each w-sized packet with w bytes. Finally, as long as $(\rho_i \geq M_r.\min)$ and $i < \chi'$ we generate that all packets ascending order of hash value are sampled as in Lemma 1. ∎

The remainder of the section deduces formal accuracy guarantees given a uniform traffic sample. Therefore, it applies to the UWRA algorithm but we were unable to prove this property for the NWPS algorithm in weighted streams.

C. Volume Estimation Correctness

The correctness of our algorithm for volume estimation follows immediately from the analysis of KMV [10].

D. Frequency Estimation Correctness

We start with the following lemma that determines the required sample size to perform frequency estimation. For simplicity, in this section we assume that the algorithm was able to collect exactly χ samples (which is the worst case for our merging procedure).

Lemma 2. *Denote $\chi \triangleq 3\varepsilon^{-2}\log 2\delta^{-1}$. Let $\mathcal{S}' \in \binom{\mathcal{S}}{\chi}$ be a random subset of size χ of $\mathcal{S}$ and let $p \triangleq \frac{\chi'}{|\mathcal{S}|}$ be the probability in which each packet is sampled. If we denote by f'_x the frequency of a flow $x \in \mathcal{U}$ in $\mathcal{S}'$, then* $\Pr\left[|p^{-1}f'_x - f_x| > |\mathcal{S}|\varepsilon\right] \leq \delta$.

Proof: We use the Chernoff Bound [48] that states that for all $0 < t, p' \leq 1$ and $n \in \mathbb{N}$, a binomial random variable $X \sim \text{Bin}(n, p')$ satisfies $\Pr\left[|X - np'| \geq t \cdot np'\right] \leq 2e^{-np't^2/3}$. Let X denote the number of appearances of x in $\mathcal{S}'$. Then $X \sim \text{Bin}(f_x, p)$ and according to the inequality

$$\begin{aligned}
\Pr\left[|p^{-1}f'_x - f_x| > |\mathcal{S}|\varepsilon\right] &= \Pr\left[|f'_x - pf_x| > p|\mathcal{S}|\varepsilon\right] \\
= \Pr\left[|f'_x - pf_x| > pf_x \cdot \frac{\varepsilon|\mathcal{S}|}{f_x}\right] &\leq 2e^{-pf_x\left(\frac{\varepsilon|\mathcal{S}|}{f_x}\right)^2/3} \\
= 2e^{-\frac{\chi'}{|\mathcal{S}|}f_x\left(\frac{\varepsilon|\mathcal{S}|}{f_x}\right)^2/3} &= 2e^{-\left(\frac{\chi'\varepsilon^2|\mathcal{S}|}{3f_x}\right)} \\
\leq 2e^{-\frac{|\mathcal{S}|\log 2\delta^{-1}}{f_x}} &\leq 2e^{-\log 2\delta^{-1}} = \delta.
\end{aligned}$$

∎

Theorem 5 utilizes Lemma 2 to solve the formal frequency estimation problem.

Theorem 5. *There exists an algorithm that requires: $O\left(\varepsilon^{-2} \cdot \log\delta^{-1} \cdot \log n\right)$ bits at each NMP, processes packets in $O(1)$ amortized case time, and solves (ε, δ)-DISTRIBUTED FREQUENCY ESTIMATION.*

Proof: Recall that our algorithm sets the error parameters of the volume estimation to $\varepsilon_V = \varepsilon/3$ and $\delta_V = \delta/2$ and the parameters of the distributed sampling to $\varepsilon_s = \varepsilon/2$ and $\delta_s = \delta/2$. This means that by the correctness of the DISTRIBUTED VOLUME ESTIMATION algorithm, we have that $\widehat{V}$ is a $(1 + \varepsilon_V)$ multiplicative approximation of $|\mathcal{S}|$, i.e., $\Pr\left[|\widehat{V} - |\mathcal{S}|| > |\mathcal{S}|\varepsilon_V\right] \leq \delta_V$. Recall that we observe the sampled frequency of the queried item f'_x, and that according to Lemma 2: $\Pr\left[\left|\frac{|\mathcal{S}|}{\chi}f'_x - f_x\right| > |\mathcal{S}|\varepsilon_s\right] = \Pr\left[|p^{-1}f'_x - f_x| > |\mathcal{S}|\varepsilon_s\right] \leq \delta_s$. Since $\delta_V + \delta_s = \delta$, we have that with probability $1 - \delta$ $\left|\frac{|\mathcal{S}|}{\chi}f'_x - f_x\right| \leq |\mathcal{S}|\varepsilon_s$ *and* $|\widehat{V} - \mathcal{S}| \leq |\mathcal{S}|\varepsilon_V$. Recall that our estimator is $\widehat{f_x} = f'_x\widehat{V}/\chi$. Therefore, since $0 \leq f_x \leq |\mathcal{S}|$ and $\varepsilon < 1$, we get: $\widehat{f_x} - f_x = f'_x\widehat{V}/\chi - f_x \leq (1+\varepsilon_V)f'_x|\mathcal{S}|/\chi - f_x$ $\leq (1 + \varepsilon_V)\left(f'_x|\mathcal{S}|/\chi - f_x\right) + |\mathcal{S}|\varepsilon_V$ $\leq (1 + \varepsilon_V)|\mathcal{S}|\varepsilon_s + |\mathcal{S}|\varepsilon_V = |\mathcal{S}|(\varepsilon_V + \varepsilon_s + \varepsilon_V\varepsilon_s) = |\mathcal{S}|\varepsilon$. Similarly:

$$\begin{aligned}
\widehat{f_x} - f_x = f'_x\widehat{V}/\chi - f_x &\geq (1 - \varepsilon_V)f'_x|\mathcal{S}|/\chi - f_x \\
&\geq (1 - \varepsilon_V)\left(f'_x|\mathcal{S}|/\chi - f_x\right) - |\mathcal{S}|\varepsilon_V \\
\geq -(1-\varepsilon_V)|\mathcal{S}|\varepsilon_s - |\mathcal{S}|\varepsilon_V &= -|\mathcal{S}|(\varepsilon_V + \varepsilon_s - \varepsilon_V\varepsilon_s) \geq -|\mathcal{S}|\varepsilon.
\end{aligned}$$

We thus conclude that $\Pr\left[\left|\widehat{f_x} - f_x\right| \geq |\mathcal{S}|\varepsilon\right] \leq \delta_V + \delta_s = \delta$. Finally, since $\varepsilon_V = \varepsilon_s = \Theta(\varepsilon)$ and $\delta_V = \delta_s = \Theta(\delta)$, the memory per NMP is $O\left(\varepsilon^{-2} \cdot \log\delta^{-1} \cdot \log n\right)$. ∎

E. Heavy Hitters Correctness

The following theorem shows that our Heavy Hitters algorithm is correct.

Lemma 3. *Let $\varepsilon, \delta, \theta > 0$ and $\chi' \triangleq \lceil 9\varepsilon^{-2}\log(2\delta^{-1}\varepsilon^{-1})\rceil$. Let $\mathcal{S}' \in \binom{\mathcal{S}}{\chi'}$ be a random subset of size χ' of $\mathcal{S}$. If we denote by f'_x the frequency of a flow $x \in \mathcal{U}$ in $\mathcal{S}'$, then $S \triangleq \{x \in \mathcal{S}' \mid f'_x \geq (\theta - \varepsilon/2)\chi'\}$ solves (ε, δ)-DISTRIBUTED HEAVY HITTERS.*

Proof: Denote by $H \triangleq \{x \in \mathcal{U} \mid f_x \geq |\mathcal{S}|\theta\}$ be the set of true heavy hitters, and let $x \in H$. We apply Lemma 2 with $\delta' = \delta\varepsilon$ and $\varepsilon' = \varepsilon/2$ to get

$$\begin{aligned}
&\Pr\left[f'_x < (\theta - \varepsilon/2)\chi'\right] = \Pr\left[f'_x < (\theta - \varepsilon')\chi'\right] \\
&= \Pr\left[\frac{|\mathcal{S}|}{\chi'}f'_x < |\mathcal{S}|(\theta - \varepsilon')\right] = \Pr\left[\frac{|\mathcal{S}|}{\chi'}f'_x - f_x < |\mathcal{S}|(\theta - \varepsilon') - f_x\right] \\
&\leq \Pr\left[\frac{|\mathcal{S}|}{\chi'}f'_x - f_x < |\mathcal{S}|(\theta - \varepsilon') - |\mathcal{S}|\theta\right] \\
&= \Pr\left[\frac{|\mathcal{S}|}{\chi'}f'_x - f_x < -|\mathcal{S}|\varepsilon'\right] \leq \Pr\left[\left|\frac{|\mathcal{S}|}{\chi'}f'_x - f_x\right| > |\mathcal{S}|\varepsilon'\right] \leq \delta'.
\end{aligned}$$

Thus, the probability of each $x \in H$ to not be reported is at most δ'. We then use the Union bound to conclude that the probability *all* of H is successfully reported in H is at least $1 - |H|\delta' \geq 1 - \theta^{-1}\delta' \geq 1 - \varepsilon^{-1}\delta' \geq 1 - \delta$. Next, let $NH \triangleq \{y \in \mathcal{U} \mid f_y < |\mathcal{S}|(\theta - \varepsilon)\}$ be the set of non-heavy flows and let $y \in NH$. Then, we use Lemma 2 for y to obtain

$$\begin{aligned}
\Pr\left[f'_y > (\theta - \varepsilon/2)\chi'\right] &= \Pr\left[\frac{|\mathcal{S}|}{\chi'} f'_y > |\mathcal{S}|(\theta - \varepsilon')\right] \\
&= \Pr\left[\frac{|\mathcal{S}|}{\chi'} f'_y - f_y > |\mathcal{S}|(\theta - \varepsilon') - f_y\right] \\
&\leq \Pr\left[\frac{|\mathcal{S}|}{\chi'} f'_y - f_y > |\mathcal{S}|(\theta - \varepsilon') - |\mathcal{S}|(\theta - \varepsilon)\right] \\
= \Pr\left[\frac{|\mathcal{S}|}{\chi'} f'_y - f_y > |\mathcal{S}|\varepsilon'\right] &\leq \Pr\left[\left|\frac{|\mathcal{S}|}{\chi'} f'_y - f_y\right| > |\mathcal{S}|\varepsilon'\right] \leq \delta' = \delta\varepsilon.
\end{aligned}$$

We thus conclude that our method of examining just the sampled set $\mathcal{S}'$ solves DISTRIBUTED HEAVY HITTERS. ■

Theorem 6 is the main result of the heavy hitter problem.

Theorem 6. *There exists a network-wide heavy hitter algorithm that requires $O\left(\varepsilon^{-2}\log\left(\delta^{-1}\varepsilon^{-1}\right)\log n\right)$ bits at each NMP, processes packets in $O(1)$ amortized time, answers queries in $O(\theta^{-1})$ time, and solves (ε, δ)-DISTRIBUTED HEAVY HITTERS.*

Based on the correctness of Lemma 3, the space requirement stated in Theorem 6 follows immediately. For answering queries in $O(\theta^{-1})$ time. To do so, the controller keeps the sample $\mathcal{S}'$ sorted by the observed frequencies $\{f'_x\}$.

V. EVALUATION

In this section, we evaluate UWRA and NWPS on real network traces for packet and byte volume measurements.

Evaluated Algorithms: When applicable, we compare UWRA with existing approaches. The notation DUS denotes the earliest version of our algorithm [15] that does not support weights and uses the simple controller merge algorithm from Section III-B. DUS also uses the HyperLogLog algorithm (HLL) for volume estimation, so we compare directly with HLL in that problem. In weighted volume estimation, we compare our work with the CKY estimator that generalize HLL to the weighted count distinct problem [25]. Since there are no previously suggested algorithms for network-wide and routing oblivious byte volume based per-flow frequency estimation, and heavy hitters detection then we compare our own UWRA and NWPS that utilize the advanced merge algorithm from Section III-C, to UWRA simple, and NWPS simple that use the simple merge algorithm from SectionIII-B. We vary the number of samples stored per NMP χ, which affects the memory consumption. As a point of reference, modern switches have 50-100MB of SRAM space [47] and support 400Gbps ports for a total throughput of 12.8Tbps [5]. We note that the switch memory is shared across all of its functionalities and thus it is important to check the performance when only a fraction of it is allocated for the measurement.

Implementation: We used our C++ prototypes for UWRA and NWPS [4], and for DUS [3], an open source project for HLL [4], and our own implementation of CKY.

Datasets: We used the following datasets:

1) The CAIDA Internet Trace 2016 [1], from the Equinix-Chicago high-speed monitor, denoted Chicago.
2) A data center trace [20], denoted by Univ.

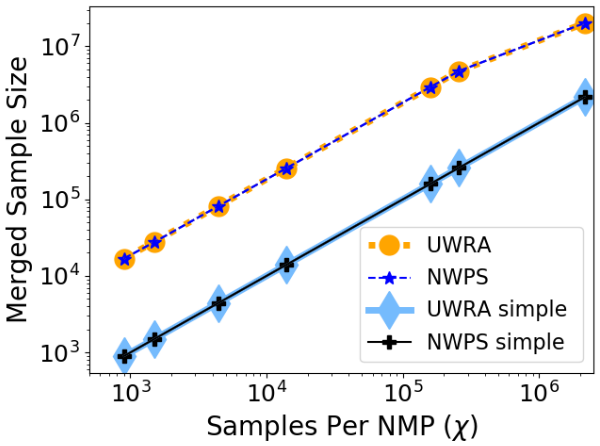


Fig. 2: Merged sample size for packet based measurements on fat-tree topology with parameter $K = 8$.

3) The CAIDA Internet Trace 2018 from New York City, denoted by New York [2].

We used the sequence number as a unique identifier for TCP packets. For the few UDP packets, we artificially assigned identifiers but in principle [30] shows a method to generate identifiers to such packets.

Metrics: We consider the following performance metrics:

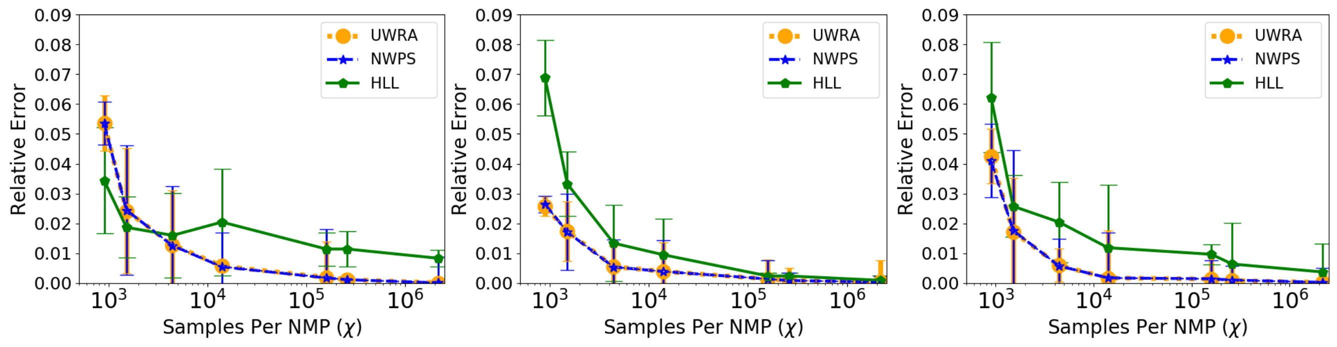
1) Mean Square Relative Error (MSRE): Measures the average of the squares of the relative errors, i.e., given the estimations $(a_1, a_2, ..., a_n)$ and true values $t_1, t_2, ..., t_n$, the MSRE is: $\frac{1}{n}\sum_{i=1}^{i=n}(\frac{a_i - t_i}{t_i})^2$
2) Root Mean Square Error (RMSE): Measures the differences between predicted values of an estimator to actual values. Formally, for each flow x the estimated frequency is $\widehat{f_x}$ and real frequency is f_x. RMSE is calculated as: $\sqrt{\frac{1}{N}\sum_x(\widehat{f_x} - f_x)^2}$.
3) F1-score: is a common way to estimate the quality of heavy hitters, where higher values are better. It is the harmonic mean of (1-FPR) and (1-FNR), where FPR and FNR are the False Positive Ratio and False Negative Ratio respectively. Formally, the F1 score of an heavy hitter set is: $2 \cdot \frac{(1-FPR)\cdot(1-FNR)}{(1-FPR)+(1-FNR)}$.

Network Topology Our evaluation uses a fat-tree topology with parameter $K = 8$, consisting of 80 switches. Each network flow is routed through of a single path from a leaf node to one of the core switches, the paths are selected by applying a hash function to the flow identifier. We use the fat-tree topology because it is common in data-center networks.

Statistical Methodology We used the first 200 million packets of each Caida trace (the Chicago trace contains 197.62GB, while the New-York trace contains 226GB), and 10 million packets from the (shorter) Univ trace (0.55 GB). Each data point was run 10 times, and we used Student's t-test to calculate the 95% confidence interval.

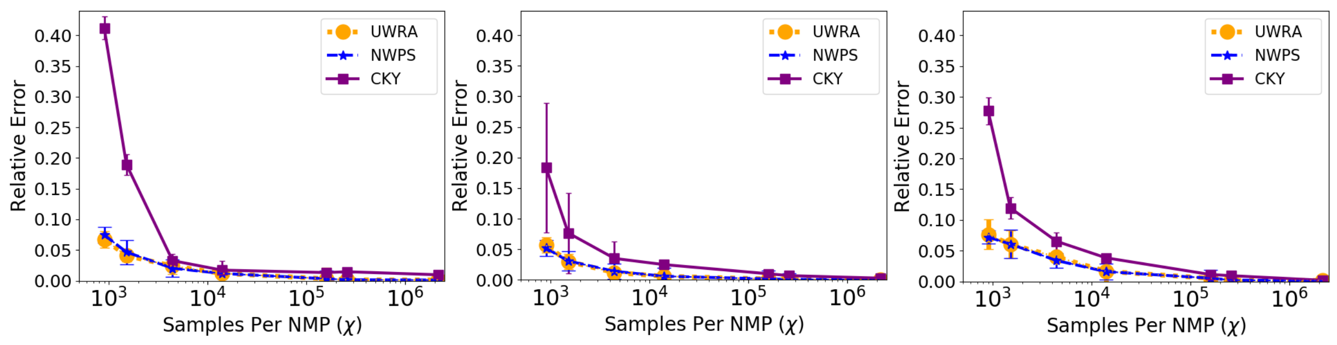
A. Merged Sample size

We begin by evaluating our improved merge algorithm compared to the simple merge algorithm [15]. Our results for packet based measurements on the Chicago trace are in



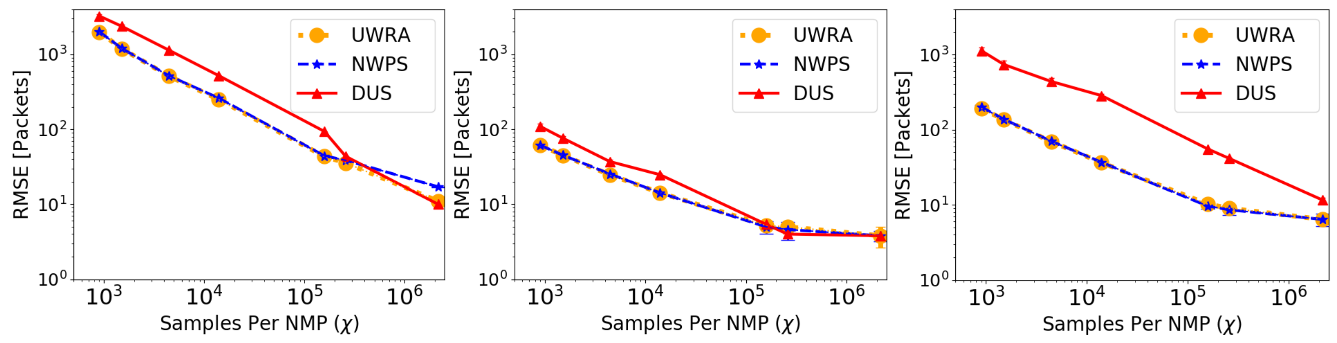
(a) Chicago (b) Univ (c) New York

Fig. 3: Relative error in estimating the total number of packets in fat-tree with parameter $K = 8$.



(a) Chicago (b) Univ (c) New York

Fig. 4: Relative error in estimating the total byte volume in fat-tree with parameter $K = 8$.



(a) Chicago (b) Univ (c) New York

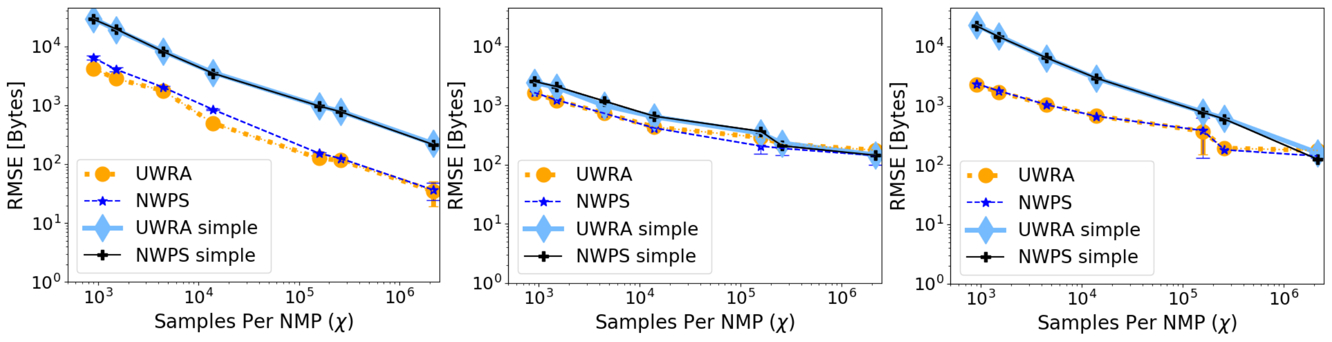
Fig. 5: Root mean square error (RMSE), for per flow packet counting in a fat-tree topology with $K = 8$.

Figure 2. Observe that both UWRA and NWPS attain almost the same sized merged sample and that such sample is ≈10-20 times bigger than the one obtained by the simple merge algorithm. The increased sample size is the root cause for the accuracy improvements, which we later show for various measurement tasks. We note that repeating this experiment for different traces as well as for byte counting yields almost identical results.

B. Volume Estimation

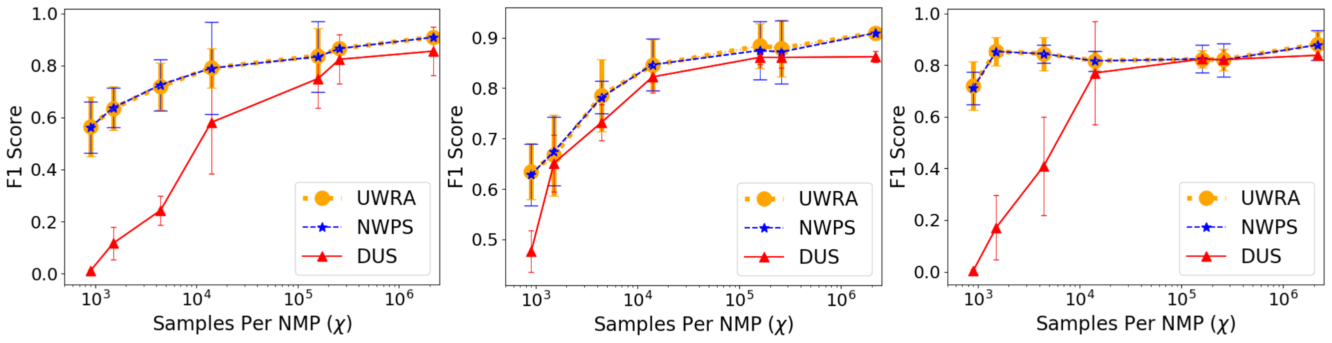
We start with the volume estimation task, where the goal is to estimate the total number of packets (or the total byte volume) in the measurement. Figure 3 shows the results for estimating the total number of packets in the measurement. As can be observed, the relative error decreases as the number of samples per NMP increases, achieving very accurate estimations for acceptable memory overheads. Also, in this case, since UWRA and NWPS devolve to the same algorithm, their accuracy is the same up to random noise. Finally, UWRA and NWPS are more accurate than the HLL algorithm since they support the advanced merge method from Section III-C. It turns out that the KMV estimator (used by our approach) is superior to the HLL estimator in network-wide measurements as HLL only supports the simple merge procedure.

Figure 4 shows the results for estimating the total number of bytes in the same topology and workloads. Surprisingly, the difference between UWRA and NWPS is very small and statistically insignificant. Therefore, the decision between them boils down to the priorities of the user. Recall that NWPS offers worst-case update complexity while UWRA adheres to formal analysis. Also, observe that UWRA and NWPS are more accurate than the CKY estimator [25] as it is



(a) Chicago (b) Univ (c) New York

Fig. 6: Root mean square erorr (RMSE), for per flow byte counting, in a fat-tree topology with $K = 8$.



(a) Chicago (b) Univ (c) New York

Fig. 7: Packet based heavy hitters on a fat-tree topology with parameter $K = 8$.

based on the HLL algorithm and only supports simple merge. Therefore, our algorithms constitute a contribution even within the broader context of weighted count distinct algorithms.

C. *Frequency Estimation*

Next, we evaluate the task of estimating per-flow frequency. Figure 5 shows results for packet-based frequency estimation. As usual, the topology is a fat-tree topology with $K = 8$. As can be observed, UWRA and NWPS are identical for packet-based measurements, and they improve the DUS algorithm. In packet-based measurement, the only difference between DUS and UWRA/NWPS is the controller merge procedure; DUS utilizes the simple merge procedure (Section III-B) while UWRA and NWPS utilize the improved merge procedure (Section III-C). In general, our algorithms attain very low RMSE within adequate space.

Figure 6 show results for estimating the number of bytes per flow. In this case, observe that UWRA is slightly better than NWPS on the Chicago workload, is slightly worse on the Univ workload, and slightly better on the New York workload. In general, the differences are very small, and UWRA/NWPS offer similar empirical performance. Since we are unaware of competing network-wide and routing oblivious algorithms for this measurement, we compare UWRA and NWPS with UWRA simple and NWPS simple that utilize the simple merge procedure to demonstrate the effectiveness of the smarter merge procedure. As can be observed, the procedure is very effective, and the simple merger requires almost an order of magnitude more memory for the same accuracy.

D. *Heavy Hitters*

We continue with the measurement task of finding the heavy hitter flows using $\theta = 0.001$ [2] (we continue with the fat-tree topology with parameter $K = 8$). Figure 7 shows results for packet-based heavy hitters identification in terms of the F1 score of the returned list of heavy hitter flows (higher is better). Please observe that UWRA and NWPS are identical and that their F1 score is notably higher than DUS. Such an improvement is due to the improved merge algorithm of UWRA and NWPS.

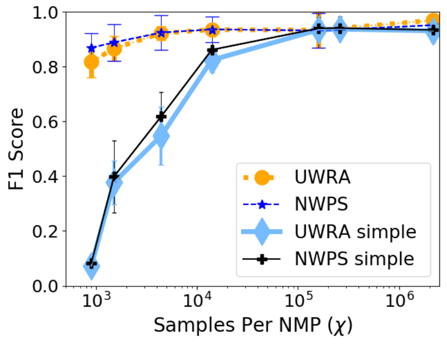
Figure 8 repeats this experiment for finding the heavy hitter flows according to byte counts. In this case, we are unaware of competing network-wide and routing oblivious algorithms, so we compare UWRA and NWPS with UWRA simple and NWPS simple that utilize the simple merge procedure.

As can be observed, our UWRA and NWPS achieve nearly identical F1 score (despite being different algorithms), and their accuracy is comparable to that of packet-based heavy hitters. They are considerably better than UWRA simple, and NWPS simple sometimes by almost three orders of magnitude. All in all, our algorithms demonstrate the ability to identify the heavy hitters accurately flows within acceptable memory constraints.

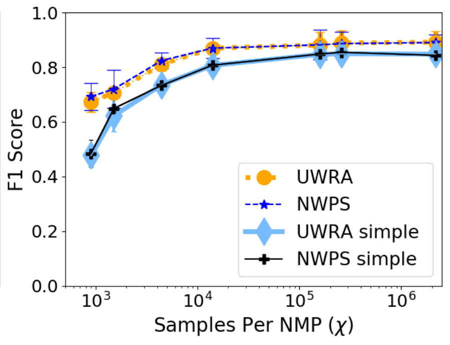
E. *Updates Over Time*

Figure 9 shows the number of updates over time for packet counting, and Figure 10 for byte counting. As can be observed,

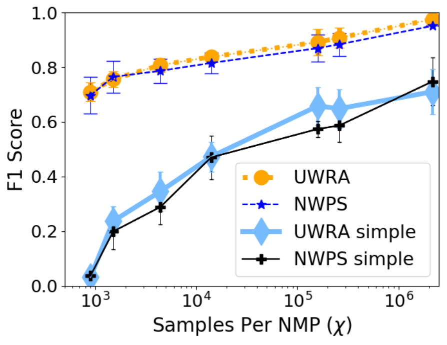
[2]We get similar results for a variety of θ values.



(a) Chicago

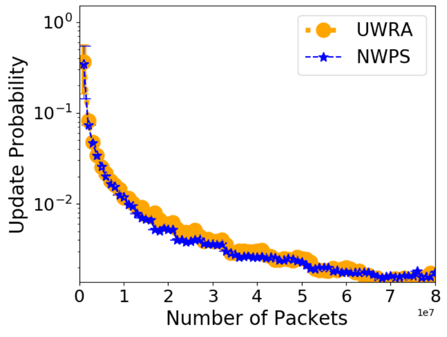


(b) Univ

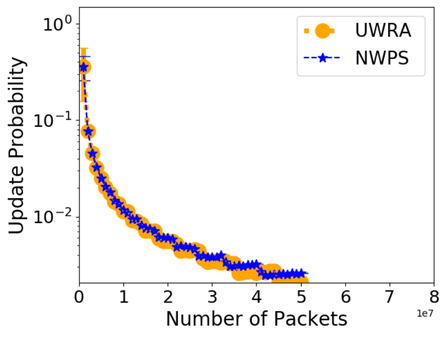


(c) New York

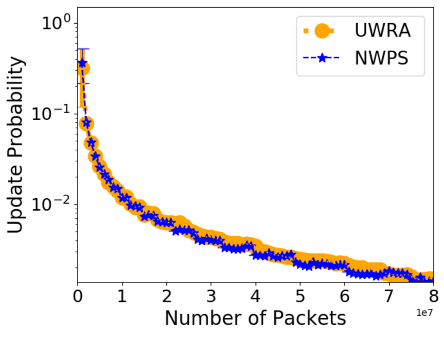
Fig. 8: Byte volume based heavy hitters on a fat-tree topology with parameter $K = 8$.



(a) Chicago

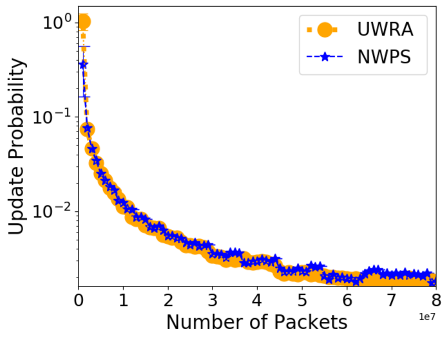


(b) Univ

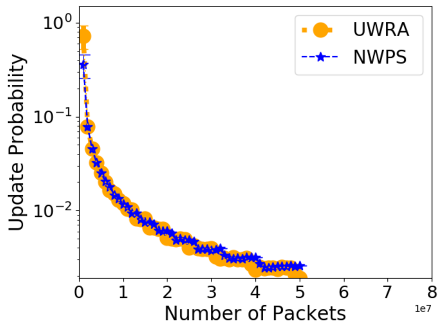


(c) New York

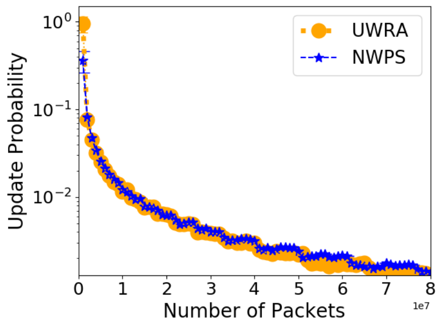
Fig. 9: Update probability for packet based measurements on a fat-tree topology ($K = 8$ χ=10000).



(a) Chicago

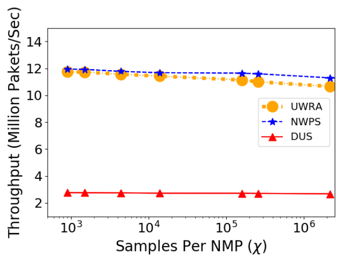


(b) Univ

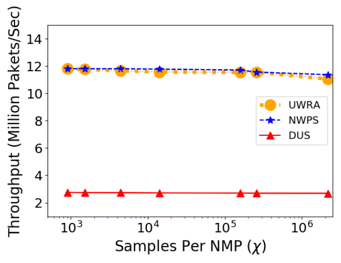


(c) New York

Fig. 10: Update probability for byte based measurements on a fat-tree ($K = 8$ χ=10000).



(a) Chicago



(b) New York

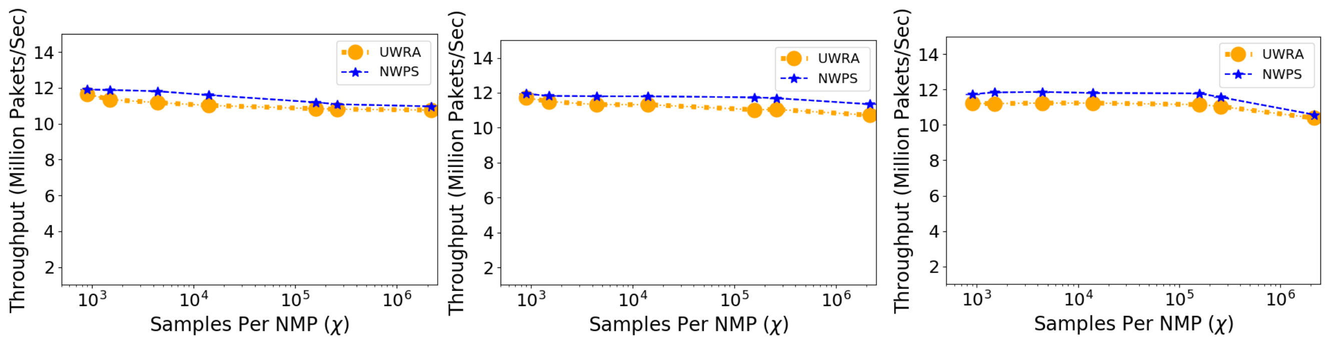
Fig. 11: Throughput (million packets/sec) of various algorithms for packet based measurement.

the rate of updates declines as the traces prolong. Our analysis indicates this (see Theorem 2) that states that the asymptotic number of updates is logarithmic in the stream length. Observe that for packet counting (unit sized weights), UWRA performs fewer updates than NWPS. However, for byte counting, UWRA performs more updates than NWPS since it sometimes adds multiple entries for the same packet.

F. Throughput Evaluation

Next, we implemented our algorithms on top of the new q-max data-structure [19] that maintains the q maximal (or minimal) items with a constant update complexity, at the expense of increased memory. In our implementation, we used a performance parameter of $\gamma = 0.25$, which means that the q-max requires 25% more memory than a heap. Our previous implantation (DUS) utilizes a heap with logarithmic complexity, which is slightly more space-efficient.

Figure 11 shows the results for packet counting, notice that NWPS and UWRA process about ≈ 10-12 million packets per



(a) Chicago (b) Univ (c) New York

Fig. 12: Throughput (million packets/sec) of various algorithms for byte based measurement.

second while DUS processes less than 3 million packets per second. Further, NWPS is marginally faster than UWRA, especially at the beginning of the measurement. Thus, choosing the new q-max structure [19] improves the processing speed by more than x3, yielding over 10M packets per seconds, which is sufficient for virtual switching as it not far from the maximal possible packet rate on a 10G link. Further, the processing speed increases as the measurement prolong as fewer packets update the q-max. Figure 12 shows that we attain very similar results for byte based measurements. That is, the optimizations we suggest incur very low processing overheads in practice.

VI. Conclusion

Our work shows the feasibility of attaining network-wide measurements without traffic manipulations, or assumptions about routing protocols, and network topology. This routing oblivious approach allows for flexible positioning on network measurement points (NMPs) without worrying about double-counting packets. In comparison, the prior work either makes assumptions on the routing protocol or restrict the placement of NMPs to avoid double counting. That is, our work is the first to study network-wide measurement problems in this routing oblivious model.

We suggested the UWRA and NWPS algorithms that attain a network-wide sample of the traffic by cleverly leveraging count distinct algorithms. We showed that this sample solves three fundamental measurement tasks; (i) total volume within a network, (ii) providing per-flow frequency estimations, and (iii) finding the heavy hitters. For all these tasks, we show formal correctness proofs, as well as an extensive evaluation. The evaluation indicates that UWRA and NWPS attain high accuracy within moderate space and that their performance is good enough for virtual switch deployment. We open-sourced our code [3], [4] to benefit the community.

Acknowledgements: This work is partially sponsored by the Technion HPI research school, the Eric and Wendy Schmidt Fund for Strategic Innovation, the Cyber Security Research Center at and the Lynne and William Frankel Center for Computing Science at Ben-Gurion University, by the Zuckerman Institute, the Technion Hiroshi Fujiwara Cyber Security Research Center, and the Israel Cyber Directorate.

References

[1] The caida ucsd anonymized internet traces 2016 - june. 20st.
[2] The caida ucsd anonymized internet traces 2018 - march. 21st.
[3] Network-wide routing-oblivious heavy hitters - available: https://github.com/jalilm/TDHH.
[4] Network-wide routing-oblivious measurements - available: https://github.com/Bilal-Tayh/RONWM.
[5] Tofino V2.
[6] Yehuda Afek, Anat Bremler-Barr, Shir Landau Feibish, and Liron Schiff. Detecting heavy flows in the SDN match and action model. *Computer Networks*, 2018.
[7] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *ACM PODS*, 2012.
[8] Daniel Anderson, Pryce Bevan, Kevin Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. A high-performance algorithm for identifying frequent items in data streams. In *ACM IMC*, 2017.
[9] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. Snap: Stateful network-wide abstractions for packet processing. In *ACM SIGCOMM*, 2016.
[10] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM*, 2002.
[11] R. B. Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu. Routing oblivious measurement analytics. In *2020 IFIP Networking Conference (Networking)*, 2020.
[12] R. B. Basat, G. Einziger, M. C. Luizelli, and E. Waisbard. A black-box method for accelerating measurement algorithms with accuracy guarantees. In *IFIP Networking*, 2019.
[13] R. B. Basat, G. Einziger, M. Mitzenmacher, and S. Vargaftik. Faster and more accurate measurement through additive-error counters. In *IEEE INFOCOM*, 2020.
[14] R. B. Basat, G. Einziger, and B. Tayh. Cooperative network-wide flow selection. In *IEEE ICNP*, 2020.
[15] Ran Ben Basat, Gil Einziger, Shir Landau Feibish, Jalil Moraney, and Danny Raz. Network-wide routing-oblivious heavy hitters. In *ACM ANCS*, 2018.
[16] Ran Ben Basat, Gil Einziger, and Roy Friedman. Fast flow volume estimation. *Pervasive and Mobile Computing*, 2018.
[17] Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Optimal elephant flow detection. In *IEEE INFOCOM*. IEEE, 2017.
[18] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. *ACM SIGCOMM*, 2017.
[19] Ran Ben Basat, Gil Einziger, Junzhi Gong, Jalil Moraney, and Danny Raz. Q-max: A unified scheme for improving network measurement throughput. In *ACM IMC*, 2019.
[20] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.
[21] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *ACM CoNEXT*, 2011.
[22] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *J. of Computer and System Sciences*, 1973.
[23] Min Chen, Shigang Chen, and Zhiping Cai. Counter tree: A scalable counter architecture for per-flow traffic measurement. *IEEE/ACM TON*, 2017.
[24] Edith Cohen and Haim Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates. In *ACM SIGMETRICS*, 2009.

[25] Reuven Cohen, Liran Katzir, and Aviv Yehezkel. A unified scheme for generalizing cardinality estimators to sum aggregation. *Inf. Process. Lett.*, 2015.

[26] Graham Cormode. Continuous distributed monitoring: A short survey. In *AlMoDEP*, 2011.

[27] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proc. VLDB Endow.*, 2008. Code: www.research.att.com/ marioh/frequent-items.html.

[28] Xenofontas Dimitropoulos, Paul Hurley, and Andreas Kind. Probabilistic lossy counting: An efficient algorithm for finding heavy hitters. *SIGCOMM CCR*, 2008.

[29] Gero Dittmann and Andreas Herkersdorf. Network processor load balancing for high-speed links. In *Proc. of the 2002 Int. Symp. on Performance Evaluation of Computer and Telecommunication Systems*.

[30] N. G. Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw.*, 2001.

[31] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 2007.

[32] Gil Einziger, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time weighted frequency estimation for virtual network functionalities. In *IEEE ICCCN*, 2017.

[33] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *SIGCOMM CCR*, 2002.

[34] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 1985.

[35] Philippe Flajolet, ric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *AOFA*, 2007.

[36] Pedro Garcia-Teodoro, Jess E. Daz-Verdejo, Gabriel Maci-Fernndez, and E. Vzquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 2009.

[37] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *ACM SIGCOMM*, 2018.

[38] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-wide heavy hitter detection with commodity switches. In *ACM SOSR*, 2018.

[39] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *ACM EDBT*, 2013.

[40] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. Sketchvisor: Robust network measurement for software packet processing. In *ACM SIGCOMM*, 2017.

[41] Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *IEEE HOTI*, 2010.

[42] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *ACM PODS*, 2010.

[43] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *USENIX NSDI*, 2016.

[44] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Lossradar: Fast detection of lost packets in data center networks. In *ACM CoNEXT*, 2016.

[45] D Lurie and HO Hartley. Machine-generation of order statistics for monte carlo computations. *The American Statistician*, 1972.

[46] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.

[47] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *ACM SIGCOMM*, 2017.

[48] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[49] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *Network, IEEE*, 1994.

[50] N. Nisan. Psuedorandom generators for space-bounded computation. In *ACM STOC*, 1990.

[51] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *ACM SOSR*, 2017.

[52] Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 2013.

[53] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *USENIX NSDI*, 2013.

[54] Haiquan Zhao, Ashwin Lall, Mitsunori Ogihara, and Jun Xu. Global iceberg detection over distributed data streams. In *IEEE ICDE*, 2010.

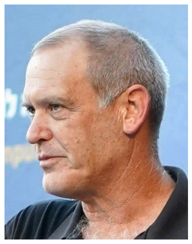
[55] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM*, 2015.

Ran Ben Basat is a Lecturer (Assistant Professor) in the Computer Science department of University College London. Previously, he was a postdoctoral fellow at Harvard University. Ran received his B.Sc, M.Sc, and Ph.D. from the CS department of Technion. He is primarily interested in algorithms for networking systems and measurement.

Gil Einziger is an assistant professor at the department of Computer Science of Ben Gurion University of the Negev, Beer Sheva, Israel. He holds a B.Sc and a Ph.D in computer science from the Technion. Earlier, Gil worked as a researcher in Nokia Bell Labs, and as a Postdoctoral Research Fellow in the Polytechnic university of Turin, Italy. He is interested in networked systems, algorithms, and security.

Shir Landau Feibish is a Senior Lecturer (Assistant Professor) at The Open University of Israel. Previously she was a postdoctoral researcher at Princeton University. Shir holds a Ph.D. and B.Sc. in Computer Science from Tel Aviv University, and an M.Sc. in Computer Science from Bar-Ilan University. Her main research interests are monitoring and management of computer networks, programmable networks and network security.

Jalil Moraney is a P.hD. Student at the Computer Science department at Technion under the supervision of Prof. Danny Raz. His main research interests are on the practicality of efficient monitoring and resource-constrained monitoring. Jalil received his B.Sc. and M.Sc.from the Computer Science department at Technion.



Danny Raz is a professor and the Hewlett-Packard Chair in Computer Engineering in the Computer Science Department, Technion, Israel Institute of Technology. Before joining the CS faculty, he was a member of the Technical staff, at the Networking Research Laboratory at Bel labs, Lucent Technologies. In 2008, he spent a year on a Sabbatical at Google in Mountain View, CA. and on 2015-2016 He spent a Sabbatical as the director of the first Bell Labs branch in Israel. His main research interests include the theory and applications of efficient network and system management, in particular, concentrating on cloud resource management, NFV, SDN, TE, and network aware services.