Constant Time Weighted Frequency Estimation for Virtual Network Functionalities

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

Marcelo Caggiani Luizelli  
Federal University of Rio Grande do Sul (UFRGS) Brazil  
mcluizelli@inf.ufrgs.br

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

Abstract—Monitoring flow volumes is a fundamental capability in network measurement. Sampling is often used to cope with the line speed and the applied methods typically rely on uniform packet sampling. However, it is inaccurate when there is a large variance in packet sizes.

In this work we introduce Byte Uniform Sampling (BUS), a sampling method for estimating flow volumes. We show that BUS can be combined with existing unweighted estimation algorithms and that the result is a weighted algorithm. BUS enables an asymptotic update time improvement as existing weighted algorithms are slower. We formally analyze BUS and evaluate it on five Internet traces. Finally, we extend the DPDK version of Open vSwitch to support BUS and demonstrate similar throughput when compared to uniform packet samples.

I. INTRODUCTION

Network functionalities such as traffic engineering, load balancing, quality of service, caching and anomaly/intrusion detection [1]–[6] often rely on on-line per-flow measurements. Such measurements can focus on packet and byte counts and the desired metric differs from one application to another. For example, load balancers deal with the physical limitations of links and therefore require byte volume information.

In practice, performing measurement at line speed is a difficult challenge due to the large number of flows and the rapid transmission rates [7]–[11]. Sampling is a fundamental measurement technique as it only processes a small fraction of the traffic. Indeed, applied technologies such as sFlow [12] and NetFlow [13] offer sampling capabilities. Unfortunately, they only support random samples that are inefficient for estimating byte volumes. The problem escalates when packets’ size variance is large. Specifically, in some real Internet traces [14], [15] less than 1% of the packets constitute up to 25% of the overall byte traffic. Clearly, a load balancer that relies on packet counts may not be able to correctly balance the traffic due to inaccurate sample input.

Another key challenge of on-line measurements is the massive data scale. Modern network devices handle several million flows [16] at the same time, and as the measurement progresses the number of monitored flows increases. Algorithms for identifying the most frequent flows, which are sometimes called heavy hitters, reduce the required space [17]–[21].

The need for software-based on-line measurement is also at the core of emerging network paradigms such as Network Function Virtualization (NFV). NFV allows network functionalities to be executed in a virtual manner on commodity servers. That is, it reduces dependence on proprietary hardware, increases flexibility and lowers operation costs.

This work introduces a new sampling method called Byte Uniform Sampling (BUS) and combines it with existing streaming algorithms for finding frequent items. Intuitively, samples provide the required speed, while the streaming algorithm handles the space limitations. We show that the combined method supports weights and can monitor flow’s byte volume by counting the number of its sampled packets. This enables an asymptotic update time improvement by using faster algorithms that can only count packets. Our approach is illustrated in Figure 1.

Fig. 1: An example of the contribution in this paper. BUS samples packets with probability proportional their byte size and sampled packets are counted with a legacy measurement algorithm. The result is a weighted meta algorithm that in some cases asymptotically improves the state of the art.

A. Contributions

We introduce a new sampling method called Byte Uniform Sampling (BUS). We prove that BUS provides accuracy guarantees and unbiased volume estimations. We then explain how BUS can be combined with existing streaming algorithms and still provide strong accuracy guarantees. BUS is extensively evaluated using five Internet packet traces and we demonstrate its accuracy improvement over uniform packet samples. The degree of improvement depends on the packets’ size variance. In some real traffic traces, uniform packet sampling requires over 50 times as many samples to match the accuracy of BUS (see Figure 3a).
We show that BUS can be combined with legacy streaming algorithms to create a faster algorithm that supports weights. When the underlying algorithm is Space Saving [20], we asymptotically improve the update complexity for weighted inputs, from logarithmic to constant. The reason for that is that Space Saving implementations operate in constant time for unweighted inputs and we can use that implementation for byte traffic measurement. We analyze the accuracy of this mode of operation and show that it provides strong accuracy guarantees. We also evaluate it and demonstrate similar accuracy to the underlying Space Saving algorithm, even when sampling less than 1% of the traffic.

Finally, we implement BUS in a DPDK-enabled Open vSwitch (OVS) and evaluate its overheads on generated traffic. We show that BUS achieves similar throughput to commonly used uniform packet samples.

II. RELATED WORK

A. Streaming algorithms

Streaming algorithms reduce the space required for performing measurement at the expense of precision. That is, they provide approximate measurements in a tractable space. These algorithms typically process every packet in the stream and require logarithmic complexity to measure per-flow volume. Alternatively, some algorithms [20], [21] can count packets in constant time. Logarithmic runtime, as well as the need to process every packet in the stream, makes them insufficient for on-line measurements.

Streaming algorithms come with two distinct flavors, sketch and counter based algorithms. Sketch based methods are composed of counter arrays, that are updated by multiple hash functions. Classical examples include Count Sketch (CS) [19] and Count Min Sketch (CMS) [18]. Sketches are simple to implement and are widely used in practice. While sketches are considered fast, increasing line speeds has motivated faster and more accurate off-line sketches. For example, Counter Braid and Counter Tree [10], [11], improve accuracy with off-line decode algorithms that reconstruct the counter values. Randomized Counter Sharing [22] updates a single counter and uses Maximum Likelihood Estimation to reconstruct frequencies.

Counter algorithms maintain a small flow cache and attempt to populate their cache with the largest or most frequent flows. They differ from each other in the cache’s maintenance policy. Examples include Lossy Counting [23] and its extensions [24], [25], Frequent [21] and Space Saving (SS) [17], [20]. Specifically, when a packet of an unmonitored flow arrives, Space Saving admits the newly arriving flow at the expense of the smallest monitored flow. Additionally, when a new flow is admitted to the cache, its estimated frequency starts from that of the evicted flow. Space Saving is often considered the state of the art [26]–[28].

B. Sampling techniques

Samples are a prominent approach for network measurement as they require to process only a small fraction of the traffic. The most common method is Packet Uniform Sampling (PUS) and is often the one used in practice [13], [12]. While conceptually simple, PUS can be ineffective for approximating byte counts since every packet is sampled with the same probability. As a result, large packets have a smaller per byte sample probability. The accuracy of PUS depends on the variance of the packets’ byte size and [29] dynamically adjust the sample probability to compensate for that variance. That is, when the variance is larger than normal, the sample probability is increased to compensate. Similar techniques are used by [30], [31] that also support constraints on resources. Similarly, [32] develops a sampled based billing service that identifies users that exceeded a certain quota without monitoring every user. Sampling can be used to measure diverse statistical features [33], and [34] even partition a sampling budget to satisfy multi-objective goals. In hardware, estimators [16], [35], [36] use samples to reduce the size of counters and fit more of them in expensive SRAM memory.

III. BYTE UNIFORM SAMPLING (BUS)

A. Preliminaries

Our measurement is modeled by a sequence of packets (S), which is initially empty and at each step an additional packet is added to (S). Packets are modeled as id and weight tuples (< id, w >). Where id ∈ U, w ∈ N \ {0} and U is the domain of ids. Packets that share the same id belong to the same flow and our goal is to estimate the total weight of a flow (B_{id}). M and A are the maximal and average packet size, M is assumed to be known in advance. N and B are the total number of packets and byte in S. Table I summarizes the notations used in this work.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Stream</td>
</tr>
<tr>
<td>N</td>
<td>Current number of packets (in all flows)</td>
</tr>
<tr>
<td>B</td>
<td>Current number of bytes (in all flows)</td>
</tr>
<tr>
<td>M</td>
<td>Maximal packet weight</td>
</tr>
<tr>
<td>A</td>
<td>Average packet weight</td>
</tr>
<tr>
<td>S_i</td>
<td>Random variable, S_i = 1 if x’s i’th packet is sampled.</td>
</tr>
<tr>
<td>S_x</td>
<td>the number of sampled packets from flow x.</td>
</tr>
<tr>
<td>S</td>
<td>Total number of sampled packets from all flows.</td>
</tr>
<tr>
<td>w_i</td>
<td>Weight of the i’th packet in bytes.</td>
</tr>
<tr>
<td>w_{ix}</td>
<td>Weight of the i’th packet in flow x (bytes).</td>
</tr>
<tr>
<td>U</td>
<td>Domain of ids.</td>
</tr>
<tr>
<td>p_i</td>
<td>i’th packet in the stream, p_i =&lt; d_i, w_i &gt;.</td>
</tr>
<tr>
<td>P_x</td>
<td>Total number of packets in flow x</td>
</tr>
<tr>
<td>P_e</td>
<td>An estimation of P_x</td>
</tr>
<tr>
<td>B_x</td>
<td>The total number of bytes in flow x</td>
</tr>
<tr>
<td>B_e</td>
<td>An estimation of B_e</td>
</tr>
<tr>
<td>τ</td>
<td>Accuracy parameters</td>
</tr>
<tr>
<td>δ, β</td>
<td>Confidence parameters</td>
</tr>
<tr>
<td>φ</td>
<td>Average packet sampling probability</td>
</tr>
<tr>
<td>β</td>
<td>Average byte sampling probability</td>
</tr>
</tbody>
</table>

TABLE I: List of symbols

Given a flow id (x), we denote by B_x ≜ ∑ w_{ix} the total byte traffic of x. Note that w_{ix} is the weight of the i’th packet
of flow $x$ (in bytes). Similarly, $P_x \triangleq \sum_{p_i \leq x, w_i > 1} 1$ is the packet traffic of $x$.

1) Problem definition: This paper addresses the $(\epsilon, \delta)$-Frequency Estimation problem. That is, for each flow an algorithm is required to provide an estimator $\hat{B}_x$ that approximates the real byte count $B_x$ in the following manner:

$$\Pr \left[ \left| \hat{B}_x - B_x \right| \leq \epsilon B \right] \geq 1 - \delta.$$ 

Where $B$ is the total byte traffic from all flows.

B. Architecture and considered scenario

In our considered architecture a network device samples some of the traffic and transfers it to the measurement algorithm. The algorithm can either be implemented in a virtual machine or as part of the network device. There are two degrees of freedom, the sampling method used and the underlying measurement algorithm. For example, in our BUS-SS algorithm, BUS is the chosen sampling method and Space Saving (SS) is the chosen measurement algorithm. Different functionalities such as load balancing, traffic engineering, and anomaly detection can then use the measurement data during their operation. This scenario is illustrated in Figure 2.

![Fig. 2: A network device forwards a sample to a monitoring service that is used by higher level network functionalities.](image)

C. The Tap: PUS and BUS sample methods

The tap controls what part of the traffic is being measured. In principle, we can forward all traffic to the measurement service that is used by higher level network functionalities.

1) Legacy PUS method: For completeness, Definition III.1 formally defines the commonly used PUS method where all packets are sampled with equal probability. We suggest two byte estimation methods: BYTE-EST-1 that simply multiplies the packet count estimation with the average packet size and BYTE-EST-2 that sums the size of sampled packets and multiplies the result with the inverse sampling probability.

Definition III.1 ((PUS) Sampling method). 1. Each packet is sampled with probability $\tau$.

The number of packets ($P_x$) is estimated with the number of sampled packets ($S_x$) normalized by the inverse sampling probability. That is: $P_x \triangleq S_x \cdot \tau^{-1}$.

There are two ways to estimate the byte traffic of a flow ($B_x$), the first is to multiply the packet estimator with the average packet size.

3. BYTE-EST($x$)-1 returns $\hat{B}_x = \hat{P}_x \cdot A$.

Alternatively, we can sum the weight the packets and multiply by the inverse sample probability.

4. BYTE-EST($x$)-2 returns $\hat{B}_x = \tau^{-1} \sum_{i=1}^{S_x} w_i$.

2) BUS method: BUS samples packets with probability proportional to their size. That is, larger packets are sampled with higher probability than smaller ones. The number of sampled packets in BUS is proportional to the total byte traffic, and we use the parameter $\beta$ as the sample probability of a single byte. BUS is formally defined in Definition III.2.

Definition III.2 ((BUS) Sampling method). 1. Packets are sampled with probability proportional to their size: The sampling probability, depends on ($w$) in the following manner:

$$P(w) \triangleq \beta w$$

BUS estimates the number of bytes in the flow by multiplying the number of sampled packets $S_x$ with $\beta^{-1}$. That is: $\hat{B}_x = S_x \cdot \beta^{-1}$.

BUS is well defined (i.e., $P(w) \leq 1$) when $\beta \leq \frac{1}{M}$.

D. The BUS-SS algorithm

Samples contribute to the operation speed while streaming algorithms are used to reduce the amount of required space. BUS can be combined with many measurement algorithms and the result is a weighted algorithm. In BUS the number of sampled packets is proportional to the byte traffic volume. Hence, the underlying algorithm simply counts the number of sampled packets. This enables BUS-SS to asymptotically improve the update time for byte volume estimation from logarithmic to constant as packet counting algorithms are asymptotically faster.

Every algorithm that satisfies the frequency estimation problem can be used with BUS. Given a flow identifier ($x$) the algorithm provides an estimator ($\hat{S}_x$) for the number of sampled packets $S_x$. The estimated traffic volume of $x$, ($B_x$) is estimated in the following manner: $\hat{B}_x \triangleq \hat{S}_x \cdot \beta^{-1}$.

BUS-SS has the following merits. (i) It is simple to implement as it uses an unmodified standard algorithm. (ii) It operates on constant time and only processes a sample of the input. (iii) It solves the weighted volume estimation problem when the traffic volume is large enough.

IV. ANALYSIS

A. BUS sampling method analysis

We start by showing that BUS is unbiased. Formally, we need to show that:

$$\mathbb{E}(\hat{B}_x) = B_x,$$
where \( \hat{B}_x \) is defined in Definition III.2.

**Theorem IV.1.** BUS is unbiased. Namely, \( \mathbb{E}(\hat{B}_x) = B_x. \)

*Proof.* Recall that \( \hat{B}_x = S_x \cdot \beta^{-1} \) and therefore it is sufficient to show that \( \mathbb{E}(S_x) = \beta B_x. \)

\[
\mathbb{E}(S_x) = \mathbb{E}\left( \sum S^i_x \right) = \sum \mathbb{E}(S^i_x)
\]

For each \( S^i_x \), the sampling probability is \( P(w^i_x) = \beta \cdot w^i_x. \) Therefore:

\[
\sum \mathbb{E}(S^i_x) = \sum \beta \cdot w^i_x = \beta \cdot \left( \sum w^i_x \right) = \beta \cdot B_x
\]

Therefore, \( \mathbb{E}(S_x) = \beta B_x \) and we get that:

\[
\mathbb{E}(\hat{B}_x) = \mathbb{E}(S_x \cdot \beta^{-1}) = \mathbb{E}(S_x) \cdot \beta^{-1} = B_x \beta^{-1} = B_x
\]

\( \Box \)

Theorem IV.1 shows that BUS is unbiased. Our next step is to be able to compare BUS and PUS on the same number of sampled packets. Toward that end, we need to establish that the average packet sample probability (\( \tau \)) has the same meaning for both methods. Formally for BUS we need to find a \( \tau \) s.t. \( \mathbb{E}(S) = \tau \cdot N \), where \( S \) is the number of sampled packets from all flows.

**Theorem IV.2.** Let \( \tau \triangleq \beta \cdot A \), then \( \mathbb{E}(S) = \tau N \)

*Proof.* Theorem IV.1 shows that:

\[
\mathbb{E}(\hat{B}) = \mathbb{E}(S \cdot \beta^{-1}) = B
\]

Since \( B = N \cdot A \), we get:

\[
\mathbb{E}(S) = \beta B = \frac{\tau}{A} B = \frac{\tau}{A} N A = \tau N.
\]

\( \Box \)

Theorem IV.2 allows us to compare BUS to PUS on equal size samples, if we also know the average packet size (\( A \)). Note that the knowledge of \( A \) is only required to evaluate PUS and BUS on equal terms. Specifically, BUS does not require any prior knowledge of \( A \).

Intuitively, there is a complex relation between the required accuracy (\( \varepsilon \)), the required confidence (\( \delta \)), the sample probability (\( \beta \)), and the total byte volume (\( B \)). Therefore, if we set \( \varepsilon, \delta \) and \( \beta \) we can derive an upper bound on the minimal stream volume that is required to solve \( (\varepsilon, \delta) \) - FREQUENCY ESTIMATION. For simplicity, we use result 4.9 in [37]: In this result, our goal is to estimate the unknown average success probability \( p \) of a random variable \( S_x \) that is the sum of multiple independent Poisson trails. The trails \( S^i_x \) only need to be independent and do not need to have the same success probability. In our case, BUS sample packets independently of each other and therefore \( S^i_x \) are independent of each other. The result states that:

\[
\Pr (p \notin [\hat{p} - \varepsilon, \hat{p} + \varepsilon]) \leq e^{-\frac{\varepsilon^2}{2B}} + e^{-\frac{\delta^2}{2}}.
\]  

(1)

We now explain how BUS matches the terms of Equation 1:

**Definition IV.1** (BUS adaptation of Eq. 1).

- \( p \triangleq \frac{\mathbb{E}(S_x)}{B_x} = \beta \), the probability that a byte belongs to a sampled packet is \( \beta \).
- \( \hat{p} = \frac{\hat{B}_x}{B_x} \) is the measured byte sample probability.

Thus, the sampling probability is \( P(w_x) = \beta \cdot w_x \). Hence, for \( \hat{p} = \frac{\hat{B}_x}{B_x} \), the sample probability is \( \hat{p} \). We get that:

\[
\Pr (\hat{p} \notin \left[ \frac{1}{\hat{p}} - \varepsilon, \frac{1}{\hat{p}} + \varepsilon \right]) \leq e^{-\frac{\varepsilon^2}{2B_x}} + e^{-\frac{\delta^2}{2}}.
\]

(2)

We now derive the required number of packets and byte samples for BUS.

**Theorem IV.3.** BUS solves \( (\varepsilon, \delta_x) \) - FREQUENCY ESTIMATION for \( B > \frac{6}{\delta_x^2} \frac{\ln(\frac{1}{\delta_x})}{\varepsilon_x^2 \beta^{-1}}. \)

*Proof.* We first need to bring the Equation 1 to match our formal problem. We apply the adaptation in Definition IV.1 of Equation 1 and get:

\[
\Pr (B_x - \hat{B} \notin [-\varepsilon_x B, \varepsilon_x B]) \leq \Pr (B_x - \hat{B} \geq \varepsilon_x B)
\]

Now that the left-hand side of Equation 1 is phrased correctly we can apply Definition IV.1 to the right-hand side and get:

\[
\Pr (B_x - \hat{B} \geq \varepsilon_x B) \leq e^{\frac{-\varepsilon_x^2 \beta B}{2}} + e^{\frac{-\varepsilon_x^2 \beta B}{3}}.
\]

(2)

Equation 2 requires us to know \( B_x \), (or an upper bound on \( B_x \)). We use the trivial observation that \( B_x < B \) to continue.

\[
\Pr (B_x - \hat{B} \geq \varepsilon_x B) \leq e^{\frac{-\varepsilon_x^2 \beta B}{2}} + e^{\frac{-\varepsilon_x^2 \beta B}{3}}.
\]

We denote \( \delta_x = e^{\frac{-\varepsilon_x^2 \beta B}{2}} + e^{\frac{-\varepsilon_x^2 \beta B}{3}} \) and solve for \( B \).

\[
\ln \left( \frac{1}{\delta_x} \right) = \frac{\varepsilon_x^2 \beta B}{2} + \frac{\varepsilon_x^2 \beta B}{3}.
\]

\[
B = \frac{6}{\varepsilon_x^2 \beta} \ln \left( \frac{1}{\delta_x} \right) = \frac{1}{\delta_x^2} \frac{\ln(\frac{1}{\delta_x})}{\varepsilon_x^2 \beta^{-1}}.
\]

(3)

Therefore, for byte traffic volumes larger than \( B \) we have:

\[
\Pr (B_x - \hat{B} \geq \varepsilon_x B) \leq \delta_x, \text{ and therefore:}
\]

\[
\Pr (B_x - \hat{B} \leq \varepsilon_x B) \geq 1 - \delta_x, \text{ completing the proof.}
\]

\( \Box \)

In some cases, the measurement interval is known in advance, and for such measurements it may be useful to derive an upper bound on the minimal \( \beta \) that enables BUS to solve the \( (\varepsilon, \delta) \) - FREQUENCY ESTIMATION problem for the given measurement size.

**Corollary IV.4.** Given \( \varepsilon_x, \delta_x \) and \( B \), BUS solves \( (\varepsilon, \delta_x) \) - FREQUENCY ESTIMATION for \( \beta > \frac{6}{\delta_x^2} \frac{\ln(\frac{1}{\delta_x})}{\varepsilon_x^2 \beta^{-1}}. \)

*Proof.* The proof is derived by extracting \( \beta \) as function of \( B \), in Equation 3.

\( \Box \)
B. BUS-SS analysis

We now analyze BUS-SS, where the Space Saving algorithm (SS) receives a BUS sample instead of the original stream. In BUS-SS, part of the error comes from BUS \((\varepsilon_a)\) and the rest from SS \((\varepsilon_a)\). We name the sampling error \(\varepsilon_a\) and the approximation error \(\varepsilon_a\).

The error of streaming algorithms depends on the number of sampled packets and hence, we first need to bound the size of the entire BUS sample.

Corollary IV.5. Consider a BUS sample with probability \(\beta\), and stream of volume \(B \geq \frac{6}{5} \ln \left(\frac{1}{\delta}\right) \varepsilon_a^2 \cdot 2^{-1}\), then

\[
\Pr(S \leq \beta B (1 + \varepsilon_a)) \geq 1 - \delta.
\]

Proof. Note that the expected number of sampled packets is \(E(S) = \beta \cdot B\), and that \(B \geq \frac{6}{5} \ln \left(\frac{1}{\delta}\right) \varepsilon_a^2 \cdot 2^{-1}\). Therefore, we use Theorem IV.3 for the entire sample to complete the proof.

Using Corollary IV.5 we can guarantee that although the number of sampled packets fluctuates, the actual sample size is bounded. Given an error parameter \(\varepsilon_a\) and an algorithm \(\hat{A}\) that solves the \((\varepsilon_a, \delta_a)\) - Frequency Estimation problem, we define:

\[
\varepsilon_a = \frac{\varepsilon_a^*}{1 + \varepsilon_a}.
\]

According to Corollary IV.5, with probability \(1 - \delta\) the number of sampled packets is at most \((1 + \varepsilon_a)\beta B\) and the accuracy is as required.

Theorem IV.6. Consider BUS-SS where an algorithm \((\hat{A})\) that solves the \((\varepsilon, 0)\) - Frequency Estimation problem receives a BUS sample as input. If \(B \geq \frac{6}{5} \ln \left(\frac{1}{\delta}\right) \varepsilon_a^2 \cdot 2^{-1}\), then

\[
\Pr \left( \left| B_x - \hat{B}_x \right| \leq \varepsilon B \right) \geq 1 - \delta, \text{ for } \varepsilon = \varepsilon_a + \varepsilon_a \text{ and } \delta = 2\delta_a.
\]

That is, BUS-SS solves \((\varepsilon, \delta)\) - Frequency Estimation.

Proof. Algorithm \(\hat{A}\) solves the \(\varepsilon_a\) frequency estimation problem. That is, it provides an estimator \(\hat{S}_x\) for \(S_x\), the number sampled packets from flow \(x\). \(\hat{A}\) is deterministic and hence:

\[
\Pr \left( \left| S_x - \hat{S}_x \right| \geq \varepsilon_a \beta B \right) = 0.
\] (4)

\[
B \geq \frac{6}{5} \ln \left(\frac{1}{\delta}\right) \varepsilon_a^2 \cdot 2^{-1}\quad \text{and according to Theorem IV.3:}
\]

\[
\Pr \left( \left| B_x - \hat{B}_x \right| \geq \varepsilon_a B \right) \leq \delta_a.
\] (5)

According to definition III.2: \(\hat{B}_x = S_x \cdot \beta^{-1}\). We need to show that:

\[
\Pr \left( \left| B_x - \hat{S}_x \beta^{-1} \right| \geq \varepsilon B \right) \leq \delta_a.
\]

In order to do so, we rewrite the above as:

\[
= \Pr \left( \left| B_x - \hat{B}_x + \hat{B}_x - \hat{S}_x \beta^{-1} \right| \geq \varepsilon B \right)
\]

\[
\leq \Pr \left( \left| B_x - \hat{B}_x \right| + \left| \hat{B}_x - \hat{S}_x \beta^{-1} \right| \geq \varepsilon B \right)
\]

Since \(\hat{B}_x = S_x \beta^{-1}\) we can write the above as:

\[
= \Pr \left( \left| B_x - \hat{B}_x \right| + \left| S_x \beta^{-1} - \hat{S}_x \beta^{-1} \right| \geq \varepsilon B \right)
\] (6)

Note that: \(\Pr \left( \left| S_x \beta^{-1} - \hat{S}_x \beta^{-1} \right| \geq \varepsilon_a B \right) = 0\)

Due to Equation 4. \(\Pr \left( \left| S_x - \hat{S}_x \right| \geq \varepsilon_a B \right) = 0\). We can reason about Equation 6 in the following manner:

\[
\leq \Pr \left( \left| B_x - \hat{B}_x \right| \geq \varepsilon B \right) + 0 \leq \delta_a
\] (7)

That is, we showed that: \(\Pr \left( \left| B_x - \hat{S}_x \beta^{-1} \right| \geq \varepsilon B \right) \leq \delta_s\) and therefore: \(\Pr \left( \left| B_x - \hat{S}_x \beta^{-1} \right| \leq \varepsilon B \right) \geq 1 - \delta_s\), accounting for over sample and applying union bound yields

\[
\Pr \left( \left| B_x - \hat{S}_x \beta^{-1} \right| \leq \varepsilon B \right) \geq 1 - \delta \quad \text{completing the proof.}
\]

Theorem IV.6 shows the correctness of BUS-SS.

V. EXPERIMENTAL EVALUATION

A. Metrics and datasets

Our evaluation includes four datasets of a mix of UDP/TCP and ICMP packets collected from major backbone routers in both Chicago [38], [39] and San Jose [14], [15] in the years 2014-2016. We also include a trace of TCP packets from the UCLA Computer Science department [40] in the year 2001. In these datasets, the percentage and volume of large TCP packets varies; in the San Jose traces [14], [15] less than 1% of the packets account for up to 25% of the traffic volume. In the Chicago datasets large TCP packets are insignificant, yet in both datasets these packets become increasingly common. In the UCLA trace, there are no large tcp packets probably because the technological state of 2001.

On-Arrival volume estimation

We use the On-Arrival error model that is also used by [17], [41]. That is, a query is performed before every packet arrival. We then measure the Root Mean Square Error (RMSE) of the algorithm, i.e.,

\[
RMSE(Alg) \triangleq \sqrt{\frac{1}{|S|} \sum_{t=1}^{S} (\hat{B}_{x_t} - B_{x_t})^2},
\]

where \(x_t\) is the identifier of the \(t\)th packet.

B. BUS sampling method evaluation

We start with the evaluation of BUS and compare it to the commonly available PUS. For a fair comparison, we measured the average packet size (A) in each trace, and used Theorem IV.2 to assure that both BUS and PUS sample on average the same number of packets. We compare BUS to both methods suggested in Definition III.1. PUS1 utilizes BYTE-EST-1, and multiplies the packet estimation with A. While PUS2, utilizes the method BYTE-EST-2 that maintains
Fig. 3: The effect of the packet sampling probability \( (\tau) \) on estimation error.

The results in Figure 4 show that for all datasets except UCLA01, BUS-SS behaves very similar to Vanilla SS. When the packet sample probability is too low, the error of BUS-SS is dominated by the sample error and is considerably higher than that of Vanilla SS. It is encouraging to observe that the 'break-away' point still allows for sampling rates of about \( 2^{-9} \) in the San Jose traces and \( 2^{-8} \) in the Chicago and UCLA traces.

VI. Open vSwitch Implementation

This section describes our implementation of BUS and PUS in DPDK enabled Open vSwitch (OVS). We start by providing a high level overview for the OVS architecture and continue by describing our implementation and the performed evaluation.

A. Background and overview

Virtual switching is a key building block of any virtualized environment which enables flexible interconnection of virtual hosts. Such flexibility is fundamental to NFV as it allows chaining virtual network functionalities together. Operation speed is the key bottleneck of virtual switching as it competes with dedicated and custom made hardware.

Context switches between user and kernel spaces cause performance bottlenecks. Specifically, the routing of each arriving packet requires a context switch to reach the physical network interface card (NIC) and its data needs to be physically copied from the NIC’s buffer to a user space buffer. In practice, high-speed virtual switching is enabled by many software and
hardware acceleration technologies such as NetMap [42] and Intel’s Data Plane Developer Kit (DPDK) [43] that reduce the number of context switches and the amount of copied data.

DPDK allows OVS to perform the entire packet processing pipeline in user space and thereby avoiding the aforementioned overheads. It consists of a set of data plane libraries that optimize packet processing. DPDK takes advantage of Poll Mode Drivers (PMD) for user space packet processing and direct access to NIC buffers.

Therefore, our implementation targets the DPDK version of OVS. That version is designed as a multilayer software switching implementation composed of two main components: ovs-vswitchd\(^1\) and ovsdb-server. Due to space constraints, we focus on vswitchd component and refer the interested reader to [44] for additional information.

The vswitchd module implements both the control and the data plane 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 virtual network function). Then, for each packet, the datapath parses the headers and determines the set of actions to be applied (e.g., forwarding or rewriting a specific header).

B. BUS OVS implementation

Currently the OVS project does not have datapath sampling capabilities. The existing sampling mechanisms, such as sFlow and NetFlow, are not designed to communicate with physical and virtual interfaces. Thus, sampled traffic is sent to a user space application that is unable to send traffic back to OVS\(^2\). Therefore, we had to implement such capabilities for both physical and virtual ports. Our implementation is realized through a simple sampled-based forwarding action.

Our BUS implementation extends the DPDK accelerated datapath capabilities and is limited to the dpif-netdev component. As packets are processed through the datapath, we retain easy access to their metadata and specifically to the payload size field that is required by BUS. Our BUS implementation redefines a forwarding action as a ‘branch’ of the datapath. Sampled packets are actually forwarded and unsampled packets are dropped. Hence, our core implementation is simple and only requires few lines of code in dpif-netdev.

C. OVS Evaluation

1) Environment setup: We evaluate BUS on an NFV-based environment. Our evaluation consists of two HP ProLiant servers. Each is equipped with an Intel Xeon E3-1220v2

\(^1\)Note that the datapath is not integrated with vswitchd in the Kernel OVS.

\(^2\)OVS DPDK owns the management of all interface attached to it. Additionally, current implementation does not allow to attach kernel-owned interfaces to OVS DPDK – which would allow user space application to communicate with DPDK interfaces.
processor, with 4 physical cores at 3.1 Ghz. The servers have 8 GB RAM, and a DPDK-enabled Intel 82599ES 10 Gbit/s network device with two interfaces (physical ports) that are directly interconnected. The servers are installed with CentOS version 7.2.1511, Linux kernel version 3.10.0.

One server is used as Design Under Test (DUT), and the other is used as a traffic generator. We use Open vSwitch 2.6 compiled with Intel’s DPDK 16.07. All VNFs are configured with Fedora 22 operating system, Linux kernel version 4.0.4 – running on top of qemu version 2.6 and are configured to have a single virtual CPU pinned to a specific physical core and 512MB of RAM. Packet generation is performed on the traffic generator server that sends traffic through one network interface and receives processed traffic from the DUT server on the other network interface. We generate UDP network traffic with packet sizes that are determined by the real packet traces, the traffic is generated with MoonGen traffic generator [45].

2) Scenario: Our use case is an NFV-based service chaining deployment with a monitoring VNF. The service chain has two VNFs in line which are forwarding the received network traffic back to the traffic generator. Additionally, the traffic sent out by the second VNF is sampled to a third monitoring service. Note that this evaluation measures OVS throughput and not the performance of the monitoring service. Figure 5 depicts the aforementioned scenario.

3) Results: Figure 6 shows the average end-to-end throughput for the SanJose14 trace, for PUS and BUS compared to the two baselines. The baseline on the left shows throughout when all the packets are sampled and the one to the right shows the vanilla performance when no sampling takes place. As can be observed, BUS and PUS achieve similar throughput for all sampled probabilities; further we achieve 20% better throughput in comparison to the baseline where all the traffic is sampled. Additionally, our implementation still has non negligible overheads as performance is improved by 9% when not sampling at all. This indicates that there is still room for optimizations. Similar results were obtained on other traces.

![Fig. 5: Our evaluated scenario: Traffic passes through two VNFs and a portion of it is also sent to the monitoring service.](image)

![Fig. 6: Average measured throughput in our NFV use case.](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space</th>
<th>Query</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Saving [26]</td>
<td>$O\left(\frac{1}{\epsilon}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\log\frac{1}{\epsilon}\right)$</td>
</tr>
<tr>
<td>Frequent [46]</td>
<td>$O\left(\frac{1}{\epsilon}\right)$</td>
<td>$O(1)$</td>
<td>$O\left(\log\frac{1}{\epsilon}\right)$</td>
</tr>
<tr>
<td>CM Sketch [18]</td>
<td>$O\left(\frac{1}{\epsilon}\cdot\log\frac{1}{\delta}\right)$</td>
<td>$O\left(\log\frac{1}{\delta}\right)$</td>
<td>$O\left(\log\frac{1}{\delta}\right)$</td>
</tr>
<tr>
<td>Count Sketch [19]</td>
<td>$O\left(\frac{1}{\epsilon}\cdot\log\frac{1}{\delta}\right)$</td>
<td>$O\left(\log\frac{1}{\delta}\right)$</td>
<td>$O\left(\log\frac{1}{\delta}\right)$</td>
</tr>
<tr>
<td>BUS - SS</td>
<td>$O\left(\frac{1}{\epsilon}\right)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

TABLE II: Positioning of BUS-SS among other popular flow volume estimation algorithms. Space Saving and Frequent are deterministic, while CM Sketch, Count Sketch and BUS-SS are probabilistic.

VII. CONCLUSIONS

This work is about estimating per-flow byte volume with samples. Such a capability is needed by network functionalities but the applied uniform sampling method is sometimes inefficient due to large variation in packets’ size. We showed that this is indeed a problem in recent Internet traces.

We introduced an alternative and sampling method called BUS that samples packets with probability proportional to their byte size. We also introduced a measurement architecture where a network device transmits a BUS sample to a virtual machine that perform per-flow measurement. We showed that BUS enables the underlying algorithm simply count the number of sampled packets and still provide reliable estimations for per-flow byte volume. This enables the use of asymptotically faster algorithms. Specifically, when the underlying measurement algorithm is Space Saving the combined algorithm (BUS-SS) achieves constant update time. This improves existing works that require logarithmic update time. Table II compares BUS-SS to previous works.

We rigorously analyzed BUS and BUS-SS and proved strong accuracy guarantees. Our algorithms require a minimal amount of traffic to be included in the measurement. This is a minor limitation as modern link technologies dramatically increase the throughput. Such enhancements can be leveraged to enable shorter measurements or more aggressive sampling.

We evaluated BUS and BUS-SS using five real Internet
traces. We showed that for the same sample size BUS is always more accurate than uniform sampling. Specifically, the latter requires up to x50 times as many packets (and bytes) to match the accuracy of BUS as shown in Figure 3a. We then showed that the accuracy of BUS-SS is similar to that of its underlying algorithm even when sampling less than 1% of the traffic.

Finally, we implemented BUS in the DPDK-enabled Open vSwitch (OVS), a real open source virtual switch. Our implementation allows sampled traffic to be transmitted directly to virtual machines and thus eases the use of virtual network functionalities. We evaluated a distributed implementation of BUS-SS where Space Saving is placed in a virtual machine and OVS samples packets according to BUS and forwards them to the measurement machine. We showed that BUS is indeed practical as OVS achieves similar throughput with BUS and with the commonly used uniform packet samples. BUS is released as branch of the Open vSwitch project, and is open sourced as well [47].

ACKNOWLEDGEMENT

The authors would like to thank Ran Ben Basat for his insightful comments in early stages of this work.

REFERENCES