INVEST: Flow-based Traffic Volume Estimation in Data-plane Programmable Networks

Damu Ding[1,2], Marco Savi[3], Federico Pederzolli[1], and Domenico Siracusa[1]
[1]Fondazione Bruno Kessler, Trento, Italy [2]University of Bologna, Bologna, Italy
[3]Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milano, Italy
{*ding, fpederzolli, dsiracusa*}*@fbk.eu*, *marco.savi@unimib.it*

***Abstract*—The emergence of programmable data planes in Software-Defined Networks enables the execution of various monitoring tasks directly in network devices, overcoming the need to deliver huge amounts of information to a controller that must then process it at scale. In this paper, we aim to solve a fundamental problem arising when exploiting programmable data planes for network-wide monitoring: how to estimate the overall number of packets in the network (i.e., the traffic volume), and the related number and size of flows, while avoiding packet double counting. Most existing works solve this problem by ensuring that each packet is counted only once on its path, which limits routing or requires coordination among devices. We propose a different approach, INVEST, a flow-based traffic volume estimator for P4-based switches, that relies on and can reuse commonly employed data structures while naturally solving the double-counting problem. We theoretically analyze and experimentally evaluate our solution, which we implemented in a real P4 carrier-grade switch, finding that it is accurate, memory-efficient, and can process packets at line rate.**

I. Introduction

Network monitoring plays a key role in modern network management, collecting data and inferring statistics that effectively indicate whether the infrastructure behaves as intended. In this context, the ability to precisely estimate the *traffic volume* [1] (i.e., number of distinct packets flowing in the network), and the related *number of distinct flows* and average *flow size* (i.e., average number of packets per flow) is necessary to support a broad range of monitoring tasks, including: (*i*) *heavy-hitter* [1][2][3] detection, where flows that carry a significant portion of total traffic require knowledge of said total traffic to be correctly identified; (*ii*) *heavy-changers* [4][5] detection, where large changes in the total number of packets of a flow (or overall) require knowledge of the total volume to be correctly estimated; (*iii*) *network traffic entropy estimation* [6], which indicates traffic distribution and can be used as part of attack detection strategies; (*iv*) *DDoS* victim [7] and superspreader [8] detection, where nodes contacting/being contacted by abnormally many destinations/sources are identified in relation to the total number of flows in the network. In general, whenever a metric requires to set some global, *network-wide* threshold, then an accurate estimation of the total traffic volume is of paramount importance.

Obviously, traffic volume can be deterministically computed as the sum of packet counts of all ingress interfaces of all border routers. However, it cannot be used to (easily) estimate flow count and size, also important for the tasks above.

With the emergence of Software-Defined Networks (SDNs) and recent advancements towards the design of programmable data planes [9], it is now possible to collect or estimate several network measurements directly inside switches, by programming them using domain-specific languages like P4 [9]: this enables a plethora of novel network monitoring and security functionalities, implementable in those *programmable switches*, for a timely diagnosis of performance issues.

Several network-wide monitoring solutions exploiting programmable data planes already exist in literature [2][4][7][6]. However, as remarked in [1], most assume that each packet is counted by a single programmable switch on its path through the network, otherwise the proposed strategies are not accurate, due to the *packet double counting* problem. Unfortunately, such assumption either strongly limits routing options or necessitates coordination between the programmable switches in the network, which makes such strategies either imprecise or impractical. Alternatively, one could mark counted packets, but such a solution is inherently insecure since an attacker could pre-mark its packets, avoiding detection [1].

This paper proposes INVEST (Improved Network traffic Volume ESTimation), an accurate and memory-efficient method for the estimation of network-wide traffic volume, number of distinct flows and average flow size at the controller, designed for Internet Service Provider (ISP) networks. It *(i)* adopts commonly used data structures allocated and updated in the data plane of a (potentially) reduced number of programmable switches and *(ii)* is inherently robust with respect to packet double counting. More precisely, INVEST relies on local packet counters and distinct flow counters based on HyperLogLog [10]. Using these, our strategy is able to estimate, at the controller, the number of distinct flows in the network and the average flow size, which can then be used for total traffic volume estimation. The advantages of our method include that it can work with only a few non-border routers (e.g. in the core network), exploits data structures that can be useful for a variety of flow-based monitoring tasks beyond traffic volume estimation, and can be feasibly deployed as a stand-alone module in existing network-wide measurement systems relying on programmable data planes [4][7][6].

We theoretically analyze and experimentally evaluate INVEST, using simulations, proving that it can estimate traffic volume accurately once tuning parameters (e.g. HyperLogLog register size and flow type) are properly set. We also im-

plement INVEST in a carrier-grade Tofino-based [11] P4-programmable switch and test its performance in a physical testbed; results show that the method can process packets at line-rate with only small hardware resource usage overhead.

II. Background on HyperLogLog

HyperLogLog (HLL) [10] is a sketch-based algorithm for the estimation of the cardinality of a data stream. In our context, it can be used to estimate the number of distinct flows (i.e., *flow cardinality*) crossing a switch using little memory. HLL uses an m-sized register M, where m indicates the number of counters (each allocating d bits) included in the register, which is *updated* as the data stream is processed and then *queried* to get an estimation of the cardinality.

The *HLL_Update* operation works as follows. When an incoming packet with flow key id arrives at the switch, HLL applies a hash function with an os-bit output to id (with $os \geq \log_2 m + 2^d$): the resulting os-bit binary string H is denoted by $H = [0 : os - 1]$, where 0 is the index of the leftmost bit, while $os - 1$ that of the rightmost one. HLL then updates register M. Let $bucket$ be the leftmost $\log_2 m$ bits of H and x the remaining bits, i.e., $bucket = H[0 : \log_2 m - 1]$ and $x = H[\log_2 m : os - 1]$; M is updated following the rule: $M[bucket] = max(M[bucket], value)$, where $value$ is the index of the rigthtmost 1 of x plus one.

The *HLL_Query* operation is used to estimate the flow cardinality $\hat{n}_M$: it is computed as a harmonic mean of the m counters: $\hat{n}_M = \alpha_m \cdot m^2 \cdot (\sum_{bucket=0}^{m-1} 2^{-M[bucket]})^{-1}$, where α_m is a bias correction parameter. The standard error of HLL has been proven to be $\frac{1.04}{\sqrt{m}}$ [10].

The *union property* of HLL, leveraged in this work, ensures that multiple HLL registers, e.g. M_m and M_n, can be merged into a single register $M_{mn} = M_m \cup M_n$ to count the flow cardinality of the packet streams that have independently updated M_m and M_n, i.e., $\hat{n}_{M_m \cup M_n}$, avoiding any double counting.

III. Estimation of traffic volume

A. Problem definition

We start by formally defining the problem. **Given:**

- A time interval T_{int};
- A packet stream S in T_{int};
- The total number of packets $|S_i|$ that have traversed each switch i at the end of T_{int};
- The updated HLL register M_i (size m) for each switch i at the end of T_{int};
- The number q of programmable switches in the network;

Return an estimation of the traffic volume (i.e., the overall number of packets from stream S), denoted by $|\hat{S}_{tot}|$, that have crossed the network in T_{int}, and related $\hat{n}_{tot}$ (i.e., overall number of distinct flows) and $\hat{R}_{tot}$ (i.e., average flow size).

B. INVEST estimation method

INVEST consists of two operations, *INVEST_Update* and *INVEST_Query* (see Fig. 1). INVEST_Update is autonomously performed during T_{int} by each programmable switch's data plane every time it is crossed by a packet, while INVEST_Query is executed by the controller at the end of T_{int} using information made available by the switches. In the following, we describe those operations.

1) INVEST_Update: Each time a packet crosses a switch, its data plane updates the counter $|S_i|$ and the HLL register M_i. $|S_i|$ is simply increased by one, while M_i is updated using the flow key id of the packet, as specified by the *HLL_Update* operation [10].

2) INVEST_Query: At the end of T_{int}, $|S_i|$ and M_i $\forall i \in \{1, \ldots, q\}$ are retrieved by the controller. For each of the switches, the controller estimates the number of distinct flows $\hat{n}_i$[1] obtained by querying the HLL register M_i as specified by the *HLL_Query* operation [10]:

$$\hat{n}_i = HLL\_Query(M_i) \quad \forall i \in \{1, \ldots, q\}$$

Due to the union property of HLL, the overall number of distinct flows $\hat{n}_{tot}$ in the network can instead be estimated as:

$$\hat{n}_{tot} = HLL\_Query(M_1 \cup M_2 \cup \cdots \cup M_q)$$

Once $\hat{n}_i$ $\forall i \in \{1, \ldots, q\}$ and $\hat{n}_{tot}$ have been estimated, the controller picks the *top-k* largest $\hat{n}_i \in \mathcal{N} = \{\hat{n}_1, \cdots, \hat{n}_q\}$ with k being the minimum value that satisfies $HLL\_Query(M_1 \cup M_2 \cup \cdots \cup M_k) = \hat{n}_{tot}$[2] (see Section III-C). Clearly, $k \leq q$ and the more flows are concentrated on a small number of switches (i.e., the traffic load is strongly unbalanced), the lower k is.

Then, the average number of packets per flow, denoted by $\hat{R}_i$, is computed considering the switches belonging to the *top-k* set in the following way:

$$\hat{R}_i = \frac{|S_i|}{\hat{n}_i} \quad \forall i \in \{1, \ldots, k\}$$

The average flow size $\hat{R}_{tot}$ in the network is estimated as an average of the average number of packets per flow per switch $\hat{R}_i$, by only considering the *top-k* switches:

$$\hat{R}_{tot} = \frac{1}{k} \sum_{i=1}^{k} \hat{R}_i = \frac{1}{k} \sum_{i=1}^{k} \frac{|S_i|}{\hat{n}_i} \tag{1}$$

Finally, the traffic volume $|S_{tot}|$ is estimated as:

$$|\hat{S}_{tot}| = \hat{n}_{tot} \hat{R}_{tot} = \frac{\hat{n}_{tot}}{k} \sum_{i=1}^{k} \frac{|S_i|}{\hat{n}_i}$$

Note that the accuracy of the estimation $|\hat{S}_{tot}|$ depends on two aspects: 1) how accurate HLL is on estimating the exact values of n_i and n_{tot} and 2) how accurate the estimation $\hat{R}_{tot}$ is with respect to its exact value $R_{tot} = \frac{|S_{tot}|}{n_{tot}}$. Taking this into account, in the next subsection we prove that INVEST does indeed converge to the desired values in realistic scenarios.

[1] In this paper, the *cap* symbol identifies *estimated* values, such as $\hat{n}_i$, $\hat{R}_i$, $|\hat{S}_{tot}|$. Their cap-less counterparts n_i, R_i, $|S_{tot}|$ indicate instead *exact* values.

[2] From now on the index i will refer, without any further ambiguity, to the switches belonging to the *top-k* set (also abbreviated in *top-k switches*).

C. Theoretical analysis

Theorem 1. *The traffic volume estimator $|\hat{S}_{tot}|$ is an asymptotically unbiased estimator of $|S_{tot}|$ as $m \to \infty$, $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$ and $n_{tot} \to \infty$.*

Proof. In the considered T_{int} where $|S_{tot}|$ has to be estimated, the network is characterized by a flow stream $F_{tot} = \{f_1, f_2, \cdots, f_{n_{tot}}\}$, where f_j indicates the flow packet count of flow j and $n_{tot} = |F_{tot}|$ is the overall number of distinct flows. It thus holds that $|S_{tot}| = \sum_{j=1}^{n_{tot}} f_j$. If we define f_j^i as the packet count of f_j as recorded in switch i, with n_i being the number of distinct flows seen in switch i, we can also write $|S_i| = \sum_{j=1}^{n_i} f_j^i$. With respect to the relative error of HLL, called ϵ_{HLL}, it holds that $\mathbb{P}(|\epsilon_{HLL}| \leq 3\frac{1.04}{\sqrt{m}}) \geq 0.997$ [12] and thus HLL accuracy depends on the register size m. So, as $m \to \infty$, the estimations $\hat{n}_i$ and $\hat{n}_{tot}$ obtained by querying the HLL registers (and their union) converge to the real values (i.e., $\frac{\hat{n}_i}{n_i} = 1$ $\forall i$ and $\frac{\hat{n}_{tot}}{n_{tot}} = 1$)[3] with arbitrary high probability.

We now assume that the $|F_{tot}|$ packet counts of distinct flows are independent and identically distributed (i.i.d.) random variables, meaning the number of packets generated in a flow does not give any information on the number of packets generated in another flow, and that the packet count random variables have all the same probability distribution. This makes sense in practice even if we know that there are many distinct types of flow, e.g. short-lived HTTP requests, VoIP calls, file transfers, etc. yielding very different packet numbers per unit of time. Even if the distribution of an f_j clearly depends on the type of flow j, the distribution of flow types during T_{int} can be viewed as fixed (though not easy to characterize), hence the f_j random variables can be seen as drawn from the combined distribution mapping a flow to its type (or even instance) and then the number of packets (over T_{int}) of that type.

We then pick k samples obtained by randomly sampling with replacement (i.e., with the possibility that the same flow f_j is included in multiple random samples), each associated to index i and characterized by $F_i = \{f_1^i, f_2^i, \cdots, f_{n_i}^i\}$. It is here assumed that $F_1 \cup F_2 \cup \cdots \cup F_k = F_{tot}$, meaning that each flow belongs to at least one of the k random samples. In the practical scenario considered in this paper, this means that any switch i among the selected *top-k* switches is randomly crossed by n_i flows and each flow j in F_{tot} is seen by at least one switch, which is ensured by design, based on how k is chosen in our strategy, when all q switches in the network are programmable switches and have installed the INVEST strategy. An additional requirement is that all packets are always routed on the same path or, alternatively, that a different flow key is specified for different routing paths [13] (e.g. in the case of multicast traffic); if this latter assumption does not hold, two different switches may count a different number of packets for the same flow j while, to apply INVEST, $f_j^i = f_j^z$ must always hold if random samples F_i and F_z of i and z both include f_j[4].

As already introduced, $R_i = \frac{|S_i|}{n_i} = \frac{\sum_{j=1}^{n_i} f_j^i}{n_i}$ is the average flow packet count in random sample i. According to the Central Limit Theorem, when $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$, $R_i = \frac{|S_i|}{n_i} \sim \mathcal{N}(\mu, \frac{\sigma^2}{n_i})$, where μ is the expected number of packets per flow, σ^2 the expected variance of the Gaussian distribution, and where R_i are independent random variables, since they refer to different random samples obtained by sampling with replacement.

As stated by the Strong Law of Large Numbers, as $n_{tot} \to \infty$, μ approaches the expected number of packets per flow in F_{tot}, that is $\mu = \frac{\sum_{j=1}^{n_{tot}} f_j}{n_{tot}} = \frac{|S_{tot}|}{n_{tot}} = R_{tot}$.

In INVEST, the estimation $|\hat{S}_{tot}|$ requires the estimation of R_{tot}, which is computed as $\hat{R}_{tot} = \frac{1}{k}\sum_{i=1}^{k} R_i$. $\hat{R}_{tot}$ is a Gaussian random variable, being a linear combination of Gaussian random variables (i.e., R_i). Its expectation $\mathbb{E}[\hat{R}_{tot}]$ can be expressed, for $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$ and $n_{tot} \to \infty$, in the following way:

$$\mathbb{E}[\hat{R}_{tot}] = \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} R_i\right] = \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}\left[\frac{|S_i|}{n_i}\right] = \frac{1}{k}\sum_{i=1}^{k}\mu = \frac{|S_{tot}|}{n_{tot}}$$

$\hat{R}_{tot}$ is thus an asymptotically unbiased estimator of R_{tot} as $n_{tot} \to \infty$ and $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$, since $\frac{\mathbb{E}[\hat{R}_{tot}]}{R_{tot}} = 1$.

$|\hat{S}_{tot}|$ is then estimated as $|\hat{S}_{tot}| = \hat{n}_{tot}\hat{R}_{tot}$. $|\hat{S}_{tot}|$ is a Gaussian variable, being so $\hat{R}_{tot}$. Since $\hat{R}_{tot}$ is an asymptotically unbiased estimator of R_{tot} as $n_{tot} \to \infty$ and $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$, $|\hat{S}_{tot}|$ is asymptotically unbiased as well if also $m \to \infty$ holds (i.e., $\frac{\hat{n}_{tot}}{n_{tot}} = 1$):

$$\frac{\mathbb{E}[|\hat{S}_{tot}|]}{|S_{tot}|} = \frac{\hat{n}_{tot}\mathbb{E}[\hat{R}_{tot}]}{|S_{tot}|} = \frac{\hat{n}_{tot}\frac{|S_{tot}|}{n_{tot}}}{|S_{tot}|} = 1 \qquad \blacksquare$$

Remark. *In a practical network scenario as the one considered in this paper, n_i is upper-bounded by $|S_i|$ (i.e., the number of distinct packets crossing the switch i): in this case, the Central Limit Theorem still holds if $n_i \to |S_i|$ $\forall i \in \{1, \dots, k\}$ and $|S_i|$ is big enough, a safe assumption for a large network such as that of an ISP.*

Theorem 2. *As $m \to \infty$, $n_i \to \infty$ $\forall i \in \{1, \dots, k\}$ and $n_{tot} \to \infty$, it holds that the relative error $\left|\frac{|\hat{S}_{tot}| - |S_{tot}|}{|S_{tot}|}\right| = 0$, i.e., the estimation $|\hat{S}_{tot}|$ equals $|S_{tot}|$.*

Proof. As already defined in Theorem 1, the relative error of HLL estimation is ϵ_{HLL}. Additionally, we define the relative error of the estimation $\hat{R}_{tot}$ as $\epsilon_{\hat{R}_{tot}}$. Being $\hat{n}_{tot}$ estimated by querying a union of HLL registers, it is then possible to write:

$$\begin{aligned}|\hat{S}_{tot}| = \hat{n}_{tot}\hat{R}_{tot} &= (1+\epsilon_{HLL})n_{tot}(1+\epsilon_{\hat{R}_{tot}})R_{tot} \\ &= (1+\epsilon_{HLL})(1+\epsilon_{\hat{R}_{tot}})|S_{tot}|\end{aligned}$$

We recall that $|\epsilon_{HLL}|$ decreases as m increases, where m is the HLL register size.

[3] In this demonstration, the notations n_i and $\hat{n}_i$ can be used interchangeably. The same holds for n_{tot} and $\hat{n}_{tot}$ and for $R_i = \frac{|S_i|}{n_i}$ and $\hat{R}_i = \frac{|S_i|}{\hat{n}_i}$.

[4] Note that, in our case, double counting a flow j does not generate any issue in flow cardinality estimation, since the union property of HLL ensures that a packet counted twice (e.g. by switches i and z) is considered only once when estimating the traffic volume.

The relative error of the estimation $\hat{R}_{tot}$ of R_{tot} can be instead obtained by looking at the probability distribution of $\hat{R}_{tot}$. By Theorem 1, as $n_{tot} \to \infty$ and $n_i \to \infty$ $\forall i \in \{1, \ldots, k\}$, $\hat{R}_{tot}$ is a Gaussian random variable with $\mathbb{E}[\hat{R}_{tot}] = \mu = \frac{|S_{tot}|}{n_{tot}}$. The absolute value of the relative error $|\epsilon_{\hat{R}_{tot}}|$ is the coefficient of variation of $\hat{R}_{tot}$:

$$|\epsilon_{\hat{R}_{tot}}| = \frac{\sqrt{Var[\hat{R}_{tot}]}}{\mathbb{E}[\hat{R}_{tot}]}$$

Being $\hat{R}_{tot}$ a linear combination of independent Gaussian random variables (i.e., R_i), $Var[\hat{R}_{tot}]$ is the following:

$$Var[\hat{R}_{tot}] = Var\left[\frac{1}{k}\sum_{i=1}^{k} R_i\right] = \frac{1}{k^2}\sum_{i=1}^{k} Var[R_i] = \frac{1}{k^2}\sum_{i=1}^{k}\frac{\sigma^2}{n_i}$$

As $n_i \to \infty$ $\forall i \in \{1, \ldots, k\}$, $Var[\hat{R}_{tot}] = 0$ and $|\epsilon_{\hat{R}_{tot}}| = 0$ as well. Thus, if this condition holds, the accuracy of the estimation $|\hat{S}_{tot}|$ is only affected by the HLL register size m and improves as m increases:

$$|\hat{S}_{tot}| = (1 + \epsilon_{HLL})|S_{tot}|$$

The above formula implies:

$$\left|\frac{|\hat{S}_{tot}| - |S_{tot}|}{|S_{tot}|}\right| = |\epsilon_{HLL}|$$

If also $m \to \infty$ holds, $|\epsilon_{HLL}| = 0$ with arbitrary high probability and we can finally write:

$$\left|\frac{|\hat{S}_{tot}| - |S_{tot}|}{|S_{tot}|}\right| = 0 \qquad \blacksquare$$

Remark. *Theorem 2 explains why INVEST considers only the top-k switches to estimate the traffic volume, and not all q switches. The reason is that, to ensure good performance in a practical scenario, n_i must be large enough for all the switches involved in the estimation of $\hat{R}_{tot}$, so that $Var[\hat{R}_{tot}] \to 0$. In the case of networks characterized by unbalanced traffic matrices, it is not unusual that n_i is relatively small for some of the switches i, and including such n_i in the computation of $\hat{R}_{tot}$ would jeopardize the estimation. By selecting the top-k switches (in terms of n_i) that cover all the flows in the network, such negative effect is instead strongly mitigated. This will also be experimentally shown in Section V.*

Remark. *In general, Theorems 1 and 2 tell us that the bigger m, n_i and n_{tot} are, the better the estimation of $|S_{tot}|$ is. The value of m must be chosen big enough to ensure good estimations for n_i and n_{tot}, whose value instead depends on how a "flow" is defined. In practice, the trivial best possible estimation of $|S_{tot}|$ can be obtained when $n_i = |S_i|$ and $n_{tot} = |S_{tot}|$, i.e., when each packet is considered as an independent flow. However this solution, proposed in [14], requires a unique identifier/marker for each packet [15][16][17] (i.e., unique packet id), which does not scale well and does not permit to use HLL to support other flow-based monitoring tasks. Our Theorems show that considering sufficiently fine-grained flows (e.g., characterized by a $\{srcIP, dstIP\}$ pair as flow key rather than by simply $srcIP$ or $dstIP$) can effectively enhance the estimation of $|S_{tot}|$, since such a choice increases n_i and n_{tot}.*

Theorem 3. *If $F_1 \cup F_2 \cup \cdots \cup F_k = F_{tot}^k \subset F_{tot}$ and thus $n_{tot}^k < n_{tot}$, as $m \to \infty$, $n_i \to \infty$ $\forall i \in \{1, \ldots, k\}$ and $n_{tot} \to \infty$, it holds that $\left|\frac{|\hat{S}_{tot}^k| - |S_{tot}|}{|S_{tot}|}\right| = \left|\frac{n_{tot}^k}{n_{tot}} - 1\right|$, where $|\hat{S}_{tot}^k|$ is the estimation of the traffic volume over the k random samples.*

Proof. By definition, $\hat{R}_{tot}$ is the estimation of R_{tot} considering the k random samples, while by Theorem 2 it holds that, when $m \to \infty$, $n_i \to \infty$ $\forall i \in \{1, \ldots, k\}$ and $n_{tot} \to \infty$, $|\epsilon_{\hat{R}_{tot}}| = 0$ and $|\epsilon_{HLL}| = 0$ with arbitrary high probability. We can then write:

$$\begin{aligned}\left|\frac{|\hat{S}_{tot}^k| - |S_{tot}|}{|S_{tot}|}\right| &= \left|\frac{|\hat{S}_{tot}^k|}{|S_{tot}|} - 1\right| = \left|\frac{\hat{n}_{tot}^k \hat{R}_{tot}}{n_{tot} R_{tot}} - 1\right| \\ &= \left|\frac{(1+\epsilon_{HLL}) n_{tot}^k (1+\epsilon_{\hat{R}_{tot}}) R_{tot}}{n_{tot} R_{tot}} - 1\right| \\ &= \left|\frac{n_{tot}^k}{n_{tot}} - 1\right| \quad (\textit{As } |\epsilon_{\hat{R}_{tot}}| = |\epsilon_{HLL}| = 0) \qquad \blacksquare\end{aligned}$$

Remark. *Theorem 3 is relevant when it cannot be ensured that all flows are visible to INVEST, e.g. in a partial or incremental deployment scenario [3], where a number of programmable switches, q, coexists with legacy non-programmable devices. Theorem 3 shows that, in this case, it is better to first replace those non-programmable switches that are crossed by the largest number of distinct flows overall, so that n_{tot}^k is maximized and approaches n_{tot}. Such a strategy has been proposed in [3] and, as a consequence of Theorem 3, yields the best possible estimation of $|S_{tot}|$ in a hybrid scenario.*

IV. Implementation of INVEST in P4

We have successfully implemented our INVEST strategy, depicted in Fig. 1, in a small testbed including a P4 programmable commodity switch with a Tofino ASIC and a simple controller. An open-source version of the implemented P4_16 code and controller has been released in [18].

A. Tofino-based data plane architecture

This subsection briefly describes the architecture of a Tofino-based switch data plane [20][19], understanding which is needed to make sense of the implementation details. For additional information the reader is referred to [11].

Fig. 2 shows the Tofino architecture. The data plane includes different *pipes* (each associated to an *ingress* or *egress pipeline*); several ports are associated to a single pipe. P4 programs are executed in different pipes and use independent resources (computation/memory). When a packet enters the switch, it is processed by the ingress pipeline associated with its ingress port, then by the egress pipeline associated with its chosen egress port. Each pipeline includes a limited number of *stages*, each associated to limited computational resources that can be used to process each packet in sequence. Therefore, only a limited number of operations can be performed on each packet, to ensure line rate processing with bounded latency.

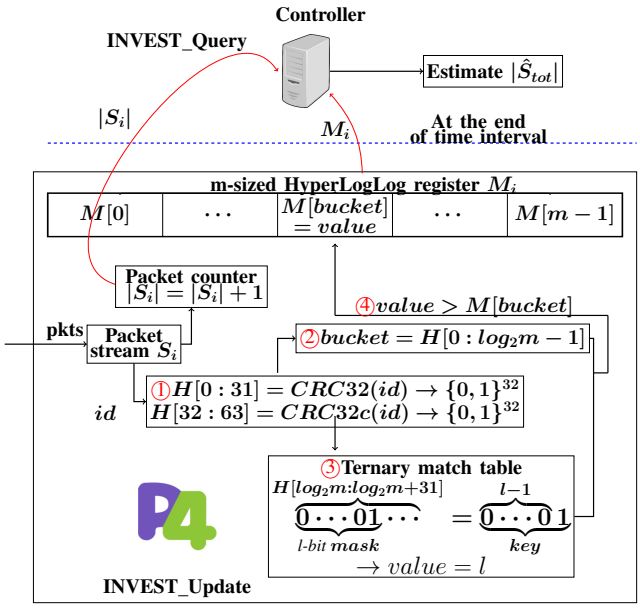


Fig. 1. Scheme of the proposed INVEST strategy

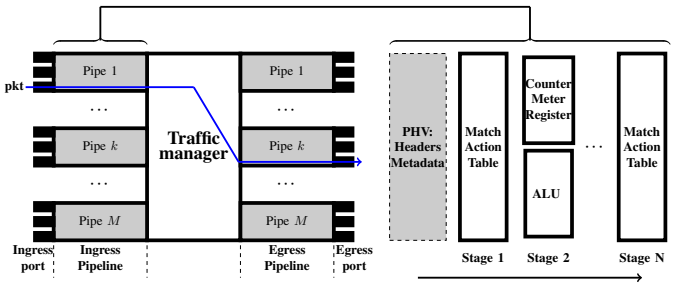


Fig. 2. Tofino-based switch data plane architecture [19]

Each stage executes operations such as applying match-action rules for customized packet processing, reading or writing counters/meters/registers, or calling Arithmetic Logic Units (ALUs) for local computations. As already mentioned, each stage has limited hardware resources, such as memory (both Static and Ternary Random Access Memories, i.e., SRAM and TCAM) and number of ALUs. Our prototype of INVEST_Update is implemented in the ingress pipeline.

B. INVEST_Update (P4-enabled Switch)

1) Counting the number of packets $|S_i|$: A register is used (a counter could also be used) to tally all incoming packets.

2) Updating the HLL Register: **Step 1** (see Fig. 1). An m-sized HLL register with $m = 2048$ is considered, where each cell is assigned $d = 5$ bits. A hash function with at least $\log_2 m + 2^d = 43$ bit of output size is thus needed by the HLL_Update operation (see Section II). Therefore, two 32-bit outputs are obtained by hashing the flow key id with two different supported hash functions and concatenated, thus generating a 64-bit hash H. Specifically, id is first hashed by the CRC32 function, setting bits $H[0:31]$; bits $H[32:63]$ are set by hashing id with the CRC32c function (see [21]).

TABLE I
TERNARY MATCH TABLE USED BY INVEST_UPDATE

Mask (32 bits)	Key (32 bits)	Action (l value)
$1000\cdots0$	$0000\cdots0$	No Action (Miss)
$1000\cdots0$	$1000\cdots0$	1
$1100\cdots0$	$0100\cdots0$	2
$\cdots$	$\cdots$	$\cdots$
$11\cdots110$	$00\cdots010$	31

Step 2 The *bucket* to be updated in the HLL register M is equal to the last $\log_2 m$ bits of H, which are easily obtained by truncating H.

Step 3 The *value*, which is the index of the rightmost 1 of the bits $H[\log_2 m : \log_2 m + 31]$ plus one (in brief l), is obtained via a ternary match table where some pre-computed values are stored, as shown in Table I. The table includes 31 entries: for each entry a *mask*, a *key* and an *action* (reporting the l value) are specified. The table is scrolled from top to bottom: the considered mask is used, by applying the bitwise AND operator with $H[\log_2 m : \log_2 m + 31]$, to retrieve the first l bits of $H[\log_2 m : \log_2 m + 31]$. Once the masked $H[\log_2 m : \log_2 m + 31]$ binary string is obtained, it is compared to the corresponding key in the table. If the masked binary string is equal to the key string, l (the *value* we are looking for) is retrieved from the *action* column and the search is stopped, otherwise the next row is considered.

Step 4 Finally, once $value$ is retrieved, if it is larger than the bucket-indexed value $M[bucket]$ in the HLL register, the new value replaces the old value.

C. INVEST_Query (Controller)

The controller is implemented on top of the Application Programming Interface (API) provided by the switch vendor. In a large-scale scenario, it could pull $|S_i|$ and M_i from each switch i in the network and estimate the traffic volume by executing the INVEST_Query operation described in Section III-B2. We implemented the INVEST_Query logic in Python.

V. PERFORMANCE EVALUATION

We implemented our entire INVEST strategy in Python to study its performance over simulated large networks. Additionally, we implemented INVEST_Update in P4_16 (see Section IV for details) in a commodity Edgecore Wedge-100BF-32X switch equipped with Intel Tofino 3.3 Tbps ASIC [11]. The switch supports up to 32 100 Gbps ports, but due to the cost of 100 Gbps interfaces, we connected it to two servers (Intel(R) Xeon(R), CPU E3-1220 V2 @ 3.10GHz, 16 GB RAM) using 10 Gbps Ethernet interfaces. The information collected by INVEST in the switch (i.e., packet counter $|S_i|$ and HLL register M_i) is queried using a Command Line Interface leveraging vendor-provided APIs.

We also implemented, in P4, a *simple forwarding* strategy for benchmarking purposes. It works as follows: if the destination IP of an incoming packet does not match any entry of an exact match-action table, the packet is forwarded to a specific egress port by applying Longest Prefix Match on the entries of another match-action table.

In the following, we will report the results obtained by *(i)* simulating INVEST in large networks and *(ii)* evaluating the performance of the INVEST P4 implementation in the Tofino-based switch.

A. Evaluation metrics and simulation settings

1) Testing flow trace and topology: In our simulations, we used the first 50 seconds of 2018-passive CAIDA flow trace [22] collected from a 10 Gbps backbone link. We divided the trace into different time intervals, considering two different widths: $T_{int} = 1$s and $T_{int} = 5$s. In the former case, each of the 50 resulting time intervals includes around 450 thousand packets, while, in the latter, each of the 10 resulting time intervals includes around 2.3 million packets.

We considered two different ISP network topologies: the 45-nodes GÉANT ISP backbone topology [23] and the 100-nodes DEFO synth100 topology [24]. The traffic matrix is generated by adopting a CRC32 hash function to randomly assign each packet to a source/destination node couple in the network. Unless otherwise specified, the output size of the CRC32 hash function is set to the number of nodes in the considered topology (45 for GÉANT and 100 for DEFO) and the source (destination) node is obtained by hashing the source (destination) IP of the packet; then, each packet is forwarded from source to destination on the shortest path (worst case for our approach, fewest traversed switches). We call this traffic matrix *balanced*, since any source/destination node couple has almost the same probability to be chosen for a source IP/destination IP couple.

2) Evaluated metric: We use *relative error*: $|S_{tot}|$ being the exact number of packets in a time interval and $|\hat{S}_{tot}|$ its estimated value, the relative error is defined as the average value of $\left|\frac{|\hat{S}_{tot}|-|S_{tot}|}{|S_{tot}|}\right| \cdot 100\%$ in all the consecutive time intervals (which are 50 for $T_{int} = 1$s and 10 for $T_{int} = 5$s).

3) Tuning parameters: Unless otherwise specified, HLL register M_i size is set to $m = 2^{11} = 2048$, with *bucket* size of 5 bits. The packet counters $|S_i|$ occupy 32 bit each. The considered flow key is $\{srcIP, dstIP\}$ pair.

B. Evaluation and comparison with existing strategies

We compare INVEST with three estimation methods:

1) *Sum*: the traffic volume is estimated by summing the packet counters $|S_i|$ as recorded by all the switches, thus neglecting the double counting problem. This trivial strategy is expected to lead to large overestimations.
2) *Sample*: the work [3] prevented double counting by using a sampling-based mechanism, where the packet count of the heaviest flows is kept in the switches in a *sample list* and delivered to the controller for the estimation of the traffic volume. However, a long tail of light flows or the existence of many heavy flows might lead to serious traffic volume underestimations.
3) *AROMA* [14]: employs a HLL-based strategy similarly to INVEST, but requires an unique packet identifier as key (i.e., it is not flow-based like INVEST). See Section VI-A for further details.

TABLE II
COMPARISON OF INVEST WITH EXISTING STRATEGIES

Estimation method	Relative error			
	GÉANT		DEFO	
	$T_{int} = 1$s	$T_{int} = 5$s	$T_{int} = 1$s	$T_{int} = 5$s
INVEST	2.33%	2.05%	2.44%	2.19%
Sum	333.01%	323.00%	412.79%	412.89%
Sample [3]	33.69%	38.78%	25.78%	30.71%
AROMA [14]	0.48%	3.10%	0.48%	3.10%

TABLE III
ESTIMATION ACCURACY OF INVEST PARAMETERS

Estimated parameter	Relative error			
	GÉANT		DEFO	
	$T_{int} = 1$s	$T_{int} = 5$s	$T_{int} = 1$s	$T_{int} = 5$s
$\|S_{tot}\|$	2.33%	2.05%	2.44%	2.19%
R_{tot}	1.92%	1.91%	2.39%	2.06%
n_{tot}	1.57%	1.91%	1.57%	1.91%

According to the parameters specified previously, the INVEST data structure occupies 10272 bits of memory in each switch (32 bits for $|S_i|$ and $2048 \cdot 5$ bits for M_i). AROMA uses the same data structure as INVEST, but adopts a different key. Instead, as specified in [3], each entry in a sample list of the Sample strategy requires 96 bits (64 bits to store the flow key, which is $\{srcIP, dstIP\}$ pair, and 32 bits to store the respective packet count). To make a fair comparison between INVEST and Sample, we thus consider a memory occupation for each sample list equal to the memory occupied by INVEST (the number of entries of the list is set to $\frac{10272 \text{ bits}}{96 \text{ bits}} = 107$).

Table II shows the comparison results. INVEST strongly outperforms both Sum and Sample in all cases, its relative error always <3%, which is considered the maximum error to ensure accuracy in practical monitoring applications [25]. As expected, Sum is an inadequate estimator (relative error >300%) as it does not handle the double counting problem. Sample also exhibits unappealing performance: a sampling list with 107 entries is not large enough to ensure good estimation (relative error >25%). INVEST shows instead similar performance to AROMA, with the already discussed great advantage of being a flow-based strategy.

Table III reports the estimation accuracy of the INVEST parameters, namely $|S_{tot}|$ (shown also in Table II), R_{tot} and n_{tot}. In most cases, the relative error in the estimation of R_{tot} and n_{tot} is below 2%, meaning that the adoption of Eq. 1 to estimate R_{tot} and the usage of HLL to estimate n_{tot} are both effective means to keep the relative error of $|S_{tot}|$ low.

C. Sensitivity analysis

1) Sensitivity to flow key types: Table IV shows how INVEST behaves when the HLL registers are updated considering different flow key types. As remarked in Section III-C, as a consequence of Theorems 1 and 2 when the flows are not sufficiently fine-grained (e.g. $srcIP$ or $dstIP$), high relative errors in the estimation occur (between 20% and 35%), as the number of distinct flows traversing each switch n_i is not large enough. Instead, when a flow is identified by the $\{srcIP, dstIP\}$ pair key, which implies finer-grained flows, the relative error significantly drops and is under 3%. We also considered as flow key $\{srcIP, dstIP, protocol\}$, where *protocol* (in brief *prot*) is, for instance, UDP, TCP, ICMP;

TABLE IV
ACCURACY OF INVEST WITH DIFFERENT FLOW KEY TYPES

Flow key	# Distinct flows n_{tot} in T_{int}		Variance σ^2 of F_{tot} in T_{int}		Relative error			
					GÉANT		DEFO	
	$T_{int}=1$s	$T_{int}=5$s	$T_{int}=1$s	$T_{int}=5$s	$T_{int}=1$s	$T_{int}=5$s	$T_{int}=1$s	$T_{int}=5$s
$srcIP$	$\sim$ 27K	$\sim$ 67K	$\sim$ 26K	$\sim$ 111.7K	20.15%	26.43%	24.80%	32.35%
$dstIP$	$\sim$ 22K	$\sim$ 58K	$\sim$ 46.5K	$\sim$ 288K	23.94%	28.64%	29.13%	34.80%
$\{srcIP, dstIP\}$	$\sim$ 47K	$\sim$ 147K	$\sim$ 13.8K	$\sim$ 40K	2.33%	2.05%	2.44%	2.19%
$\{srcIP, dstIP, prot\}$	$\sim$ 47.1K	$\sim$ 147.6K	$\sim$ 13.5K	$\sim$ 39.8K	2.36%	2.73%	2.56%	2.37%
Unique packet id (AROMA)	$\sim$ 450K	$\sim$ 2300K	0	0	0.48%	3.10%	0.48%	3.10%

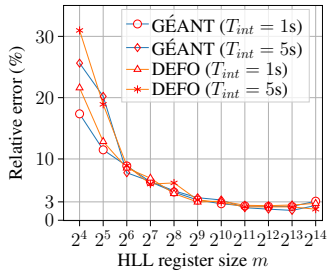


Fig. 3. Sensitivity to HLL register size m

in this case, the relative error is almost the same as for $\{srcIP, dstIP\}$, without any notable improvement. We thus decided to adopt the $\{srcIP, dstIP\}$ pair as default flow key for our evaluations, since it reduces the operations to be performed by INVEST in the data plane with respect to $\{srcIP, dstIP, prot\}$. For completeness, Table IV also reports the relative error if INVEST was updated considering *unique packet ids*, equivalently to AROMA as discussed above. In this case, the relative error caused by the estimation $\hat{R}_{tot}$ is zero. However, ensuring unique packet ids in real scenarios is challenging and this case was not implemented in hardware.

2) Sensitivity to time interval width: Table IV also shows the relative error of INVEST with different time interval widths. A larger width is naturally accompanied by a larger number of monitored flows n_{tot}, which should have a good effect on traffic volume estimation according to Theorems 1 and 2 of Section III-C; however, a higher variance σ^2 of flow packet counts in F_{tot} is also expected (see the table), and this negatively impacts on the estimation. Increasing the time interval width has thus a counteracting effect on traffic volume estimation but, unfortunately, the negative effect due to higher flow packet count variance tends to dominate. Adopting fine-grained flow keys can effectively mitigate such effect, since considering more flows in the network leads on average to less packets per flow and reduces the flow packet count variance. This is still an improvement on e.g. AROMA, because we do not need to use unique packet ids.

3) Sensitivity to network topology: If the same flow trace and flow key type are considered, obviously the variance of flow packet count in F_{tot} does not change when considering GÉANT and DEFO topologies. Since DEFO has a larger number of nodes, each switch i is assigned a smaller number of flows n_i. As we explained in Theorem 2 of Section III-C, smaller number of flows n_i leads to a higher variance on the estimation of R_{tot}: this is why the relative error of traffic volume estimation $|\hat{S}_{tot}|$ is generally slightly higher in DEFO.

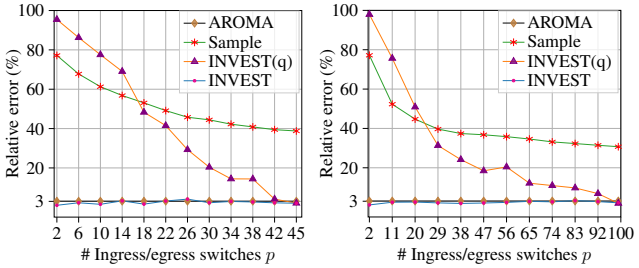
4) Sensitivity to HLL register size m: Fig. 3 shows how INVEST performs by varying the HLL register size m. Intuitively, the relative error decreases as m increases. When the HLL register size m is small (e.g., 2^4 and 2^5), the relative error is very significant. While increasing m, at a certain point (i.e., when $m = 2^{11}$) the curve flattens to a relative error that is always lower than 3%.

5) Sensitivity to flow distribution: As described in Section V-A, in previous evaluations we have considered a *balanced* traffic matrix, where any node in the network has the same probability to be picked as source (ingress) or destination (egress) of a traffic flow. Here, we want to see how INVEST performs in the case of *unbalanced* traffic matrices. To do so, instead of considering any node in the ISP topology as possible ingress or egress node for the flows, we select a subset of nodes with cardinality p as possible sources/destinations and we use the CRC32 hash function (with output size p) to assign the flows' source and destination IPs to the nodes in that subset. Once the flows' source and destination IPs have been assigned to the p switches, they are routed as before on the shortest path. Thus, with this procedure, it is possible to tune the skewness of the flow distribution: the smaller p is, the more skewed the distribution is, leading to large variances on flow packet counts in the different switches.

As remarked in Section III-C, as a consequence of Theorem 2, selecting the *top-k* switches for traffic volume estimation is beneficial in the case of unbalanced traffic matrices, since it ensures to keep $Var[\hat{R}_{tot}]$ small. To evaluate the impact of the *top-k* selection on the overall INVEST strategy, we introduce *INVEST(q)*: INVEST(q), instead of only considering the *top-k* switches in the estimation of $\hat{R}_{tot}$, considers all of them, that is, Eq. 1 in Section III-B is replaced with $\hat{R}_{tot} = \frac{1}{q}\sum_{i=1}^{q}\hat{R}_i = \frac{1}{q}\sum_{i=1}^{q}\frac{|S_i|}{\hat{n}_i}$. In the case that $|S_i| = \hat{n}_i = 0$, we define $\hat{R}_i = 0$.

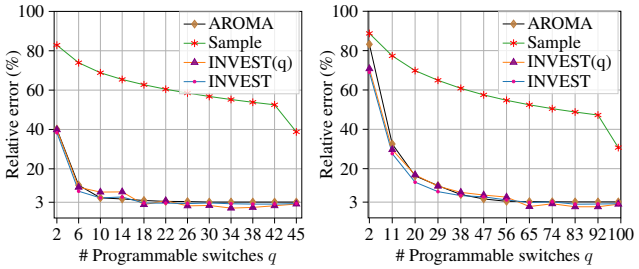
Fig. 4 shows the sensitivity of INVEST to different flow distributions when $T_{int} = 5$s. Similar results are obtained for $T_{int} = 1$s, but we do not report them for the sake of conciseness. For completeness, we include in the evaluation also the Sample strategy and AROMA [14], while we do not show results for Sum, which always leads to very large errors. The relative error of Sample decreases as p increases. However, when the traffic matrix is balanced, i.e., when p is equal to the overall number of nodes in the network, the relative error is still high (above 25%). INVEST(q) has even worse performance than Sample when p is small in both topologies, since the zero value of $\hat{R}_i$ related to the nodes that are not traversed by any flow compromises the estimated $\hat{R}_{tot}$. In contrast, INVEST always leads to good estimation performance as AROMA: no matter what network topology, time interval width or flow distribution is considered, the relative error of INVEST is always around 3%. Using the *top-k* switches for estimating the traffic volume is shown to be very effective, especially when the traffic matrix is unbalanced.

6) Sensitivity to number of programmable switches: Unlike previous evaluations, in this subsection we consider a hybrid network composed by both programmable switches and legacy devices (e.g., Openflow-based or SNMP-based switches) when



(a) GÉANT ($T_{int} = 5$s) (b) DEFO ($T_{int} = 5$s)

Fig. 4. Sensitivity to flow distribution



(a) GÉANT ($T_{int} = 5$s) (b) DEFO ($T_{int} = 5$s)

Fig. 5. Sensitivity to number of programmable switches q in a hybrid network

the traffic is *balanced*. Only some nodes are programmable switches able to implement the INVEST_Update strategy, while the remaining legacy devices are assumed not to provide any information to the SDN controller for the estimation of $|S_{tot}|$. The incremental deployment of programmable switches in the hybrid network is performed using the algorithm designed in [3], where the nodes leading to the largest union of distinct flows with previously-deployed programmable switches are iteratively replaced to ensure the highest possible flow visibility, until a number q of switches have been substituted with programmable equipment.

Fig. 5 reports the sensitivity of INVEST performance to a variation on the number of programmable switches q deployed in such a hybrid network. We include in the figure also Sample, AROMA and INVEST(q) strategies. Also in this case, we omit similar results obtained with $T_{int} = 1$s for the sake of conciseness. When more programmable switches q are deployed, the relative error of Sample decreases smoothly, since more sampled flows are reported to the controller. However, as before, the estimation is jeopardized by the missing statistics on the long tail of small flows. Conversely, INVEST, INVEST(q) and AROMA have good and comparable performance. Another observation is that, being programmable only 40% of the switches in a hybrid network and being them deployed following the strategy proposed in [3], INVEST is able to estimate $|S_{tot}|$ quite accurately (relative error <5%).

D. Evaluation of resource usage and processing time

Table V shows the switch's data plane resources required by INVEST_Update and simple forwarding implementations, as a percentage of the total resources available in the switch. To ensure a fair comparison, the INVEST_Update implementation also includes the simple forwarding logic for packet forwarding. INVEST plus simple forwarding requires around 25% stages (see Section IV-A) more than simple forwarding.

TABLE V
NORMALIZED SWITCH RESOURCE USAGE OF INVEST

Strategy	No. stages	SRAM	TCAM	No. ALUs	PHV size	Additional proc. time w.r.t. simple forwarding
Simple forwarding	16.67%	2.5%	8.33%	4.17%	7.30%	-
INVEST_Update + Simple forwarding	41.67%	3.23%	9.03%	8.33%	7.68%	45ns

Simple forwarding needs 2.5% of the total available SRAM and 8.33% of TCAM. Instead, INVEST_Update plus simple forwarding uses only 3.23% of total SRAM, which means that the occupied memory by both the packet counter $|S_i|$ and HLL register M_i is quite low. Additionally, INVEST needs 31 entries in a ternary match table (see Table I) to compute the values in the HLL register, occupying additional 0.7% of TCAM with respect to what is needed by simple forwarding. The number of required ALUs measures computational resource usage. Only 8.33% of total ALUs are used by INVEST plus simple forwarding to process the packets, double that simple forwarding alone. The packet header vector (PHV) size indicates the amount of packet header information that can be passed across the pipeline stages. INVEST plus simple forwarding uses only 7.68% of PHV and, compared to the 7.30% of simple forwarding, means that INVEST requires to pass across stages only few additional customized metadata. Moreover INVEST only requires 45 ns more than simple forwarding to process a single packet, which is an acceptable time overhead, and is always able to process packets at line rate (with a throughput or around 10 Gbps).

VI. RELATED WORK

A. Traffic volume estimation

Recently, Ben Basat *et al.* have proposed AROMA [14], a distributed traffic volume estimation strategy that, as ours, rely on HLL-like cardinality estimation, is explicitly designed to avoid the double counting problem and can be executed in programmable switches' data plane. AROMA requires that, to estimate the traffic volume exploiting the merge property of HLL, a unique identifier for each packet is needed. However, non straightforward mechanisms have to be adopted for assigning a unique key to each packet [16][17] and, for this reason, it may be infeasible to adopt such a solution on high-speed carrier-grade networks. Additionally, using a unique key for each packet to update HLL, hampers the usage of such data structure for the collection of other flow-based relevant metrics (e.g. number of distinct flows). Conversely our solution, which exploits the merge property of HLL as well, uses aggregated per-flow information for an accurate estimation of the traffic volume, being a more practical generalization of the strategy proposed in [14]. Ding *et al.* [3] proposed a strategy to estimate the traffic volume without double counting packets. It consists on locally storing the heaviest flows in a *sample list* maintained in the switches' data plane. The controller can prevent, when collecting the sample lists from multiple switches, from double counting the packets belonging to the same flow. The limitation of this solution is that the estimation accuracy significantly depends on the size of the sample lists. Considering available memory in current programmable

switches, the long tail of small flows is neglected in the traffic volume estimation. Contrariwise, INVEST can estimate the traffic volume of both large data streams and the small flows long tail, with good accuracy and low memory occupation.

B. Network flow cardinality estimation

Many sketch-based algorithms for estimating the cardinality of large data streams have been proposed in literature, including Linear Counting [26], LogLog [27] and HyperLogLog [10]; it has been proven that HyperLogLog is able to achieve the same accuracy of the other methods by requiring much less memory. With the advent of programmable data planes, estimating the cardinality of flows directly in the data plane pipeline has become an appealing solution to enhance network monitoring. Recently, AROMA [14] proposed and implemented a customized HLL algorithm in P4, where each HLL register cell requires 32 bits instead of the 5 bits required by standard HLL [10], breaking the best trade-off between estimation accuracy and memory occupation as evaluated in [10]. Instead, INVEST implements the standard Update operation of HLL in P4, without requiring additional memory.

VII. Conclusion

In this paper we presented INVEST, a novel traffic volume estimation method that exploits modern data-plane programmable switches to jointly estimate number of flows, average flow size and total packet count in the network, using a limited number of sampling locations and robust against packet double counting. We provided the theoretical justification of why the method is an unbiased estimator and can work with a relatively small number of measurement locations. In addition, we overcame the resource constraints of real P4-programmable hardware, to implement our logic in an Edgecore commodity switch equipped with Tofino ASIC. Our experimental evaluation showed that INVEST can estimate the traffic volume with high accuracy (relative error lower than 3%) and low memory occupation (few KB per switch). INVEST works well *(i)* in the case of strongly unbalanced traffic matrices and *(ii)* when only 40% of the network switches implement it. It also ensures line-rate packet processing, with only marginal time overhead with respect to a naïve forwarding strategy, and does not require any priori knowledge on the network topology, on routing and on flow distribution.

Acknowledgement

The research leading to these results has received funding from the GN4-3 project, within the European H2020 R&I program, Grant Agreement No. 856726. We also want to thank Intel for their valuable support.

References

[1] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, and D. Raz, "Network-wide routing-oblivious heavy hitters," in *IEEE/ACM Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2018.

[2] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *ACM Symposium on SDN Research (SOSR)*, 2018.

[3] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.

[4] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017.

[5] L. Tang, Q. Huang, and P. P. Lee, "MV-Sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[6] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic Sketch: Adaptive and fast network-wide measurements," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.

[7] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2016.

[8] L. Tang, Q. Huang, and P. P. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2020.

[9] P. Bosshart, D. Daly, G. Gibb *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, pp. 137–156, 2007.

[11] "Intel Tofino," 2020, https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/.

[12] P. Reviriego and D. Ting, "Security of HyperLogLog (HLL) Cardinality Estimation: Vulnerabilities and Protection," *IEEE Communications Letters*, vol. 24, no. 5, pp. 976–980, 2020.

[13] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against DDoS attacks," in *IEEE Symposium on Security and Privacy*, 2003, pp. 93–107.

[14] R. Ben-Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu, "Routing oblivious measurement analytics," *arXiv*, 2020.

[15] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Detecting heavy flows in the SDN match and action model," *Computer Networks*, vol. 136, pp. 1–12, 2018.

[16] Y. Zhu, N. Kang, J. Cao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[17] Y. Li, R. Miao, C. Kim, and M. Yu, "Lossradar: Fast detection of lost packets in data center networks," in *ACM Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2016.

[18] "P4 implementation of INVEST," 2020, https://github.com/DINGDAMU/INVEST.

[19] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[20] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[21] P. Koopman, "32-bit cyclic redundancy codes for internet applications," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2002.

[22] "CAIDA UCSD anonymized internet traces dataset - [passive-2018]," 2020, http://www.caida.org/data/passive/passive_dataset.xml.

[23] "GÉANT ISP topology," 2020, https://www.geant.org/Networks/Pan-European_network/Pages/GEANT_topology_map.aspx.

[24] "DEFO ISP topology," 2020, https://sites.uclouvain.be/defo.

[25] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 145–156, 2006.

[26] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, 1990.

[27] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *European Symposium on Algorithms*, 2003.