Robust Distributed Monitoring of Traffic Flows

Vitalii Demianiuk[1,2], Sergey Gorinsky[1], Sergey Nikolenko[2], and Kirill Kogan[1]
[1]IMDEA Networks Institute
[2]Steklov Institute of Mathematics at St. Petersburg
Email: {vitalii.demianiuk, sergey.gorinsky, kirill.kogan}@imdea.org, snikolenko@gmail.com

***Abstract*—Scalable monitoring of traffic flows faces challenges posed by unrelenting traffic growth, device heterogeneity, and load unevenness. We explore an approach that tackles these challenges by shifting a portion of the monitoring-task execution from an overloaded network element to another element that has spare resources. Moving the entire execution of the task to a lightly loaded element might be infeasible because execution on multiple elements is inherent in the task or requires at least partial participation by the particular overloaded element (e.g., flow-size computation at the ingress element for billing purposes). Distributed execution of a stateful traffic-monitoring task has to be robust against packet reordering or loss, i.e., network noise. This paper designs robust traffic monitoring where the goal is to determine a flow metric for each flow exactly in spite of network noise. We follow the open-loop paradigm that does not add any control packets, communicates flow state in-band by appending few (on the order of 2 or 4) control bits to packets of the monitored flows, and keeps latency low. We consider the task of flow-size computation, analytically derive conditions assuring correct operation of the designed algorithms, and evaluate the algorithms on realistic traffic traces. The algorithms successfully distribute the monitoring-task load without imposing significant computation or storage overhead.**

I. Introduction

Monitoring of network traffic is crucial for efficient, reliable, and secure operation of any network [1], [2]. Knowledge of traffic properties helps the network operator in capacity planning, QoS (Quality of Service) assurance, service differentiation, attack mitigation, etc. In some applications, the operator needs to know metrics of traffic flows exactly. For example, a billing application has to know exactly the size of a traffic flow at its ingress element to accurately bill the sender for delivering the traffic flow through the network.

Scalable monitoring of traffic flows is challenging due to unrelenting traffic growth, device heterogeneity, and load unevenness. First, while traffic keeps growing in both volume and number of flows, the processing and storage needed for traffic monitoring in network elements grow as well. Second, networks comprise elements of increasing heterogeneity ranging from basic IoT (Internet of Things) access devices with greatly limited capabilities to high-end core routers that forward millions of concurrent flows. Third, the traffic-monitoring load on different network elements is uneven, and one element might get overloaded even when other elements have spare resources.

978-1-7281-2700-2/19/$31.00 2019 © IEEE

A promising direction for meeting these challenges is to monitor traffic flows by utilizing resources in multiple network elements as a shared pool. If a traffic-monitoring task overloads an element, then – regardless of whether the task runs on this element alone or inherently executes on multiple elements – a part of the task execution can be shifted from the overloaded element to another element that has ample resources. This empowers the network to leverage its global processing and storage resources to effectively cope with local traffic-monitoring overloads.

Shift of a traffic-monitoring load to an element with abundant resources might require both distributed execution of the traffic-monitoring task and communication of flow state between the involved network elements. Moving the entire execution of a task from a particular overloaded element to another element might be infeasible even when the task does not inherently require multiple elements for its execution. For instance, let us revisit the billing application that monitors the sizes of the traffic flows entering the network: the specific ingress element of each flow has to participate in computing the flow size at least to some extent because otherwise packet losses inside the network would render the computation of the flow size inaccurate. Furthermore, when the execution of a stateful task is shifted from the overloaded element only partially, the distributed execution necessitates communication of flow state between the involved elements.

In practice, it is impossible to fully avoid packet reordering or loss, and distributed traffic monitoring has to be robust to such *network noise*. Unlike distribution of static policies [3]–[7], distributed execution of a stateful task needs additional means to acquire such robustness. Open-loop and closed-loop control constitute two general approaches to dealing with network noise. While a closed-loop design can adapt its operation to the current level of network noise, the feedback-driven robustness increases latency, which is undesirable for real-time monitoring. Also, asymmetric routing, restrictions on generation of new packets in a network element, and other factors might make it infeasible – or at least very difficult – to provide feedback to a previous element on the path of a unidirectional traffic flow [8]. Hence, the problem of robust distributed monitoring of traffic flows is more amenable to the open-loop approach that can communicate flow state in-band and keep latency low.

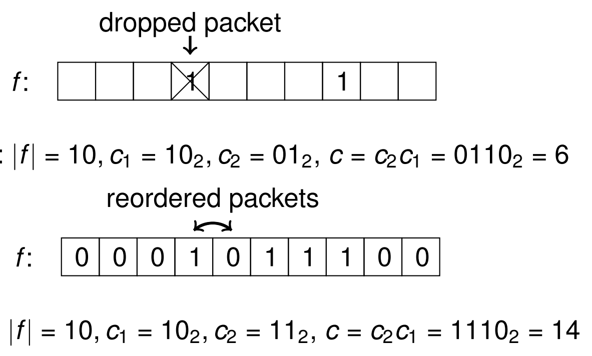
In this paper, we study robust distributed open-loop monitoring of traffic flows where the objective is to compute a flow metric for all flows exactly despite network noise. The explored approach does not introduce any control packets and communicates flow state by piggybacking few (on the order of 2 or 4) control bits on packets of the monitored flows. Our solution methodology relies on the following two design principles:

1) The elements involved in distributed monitoring manage flow state in chunks that overlap to keep the state representation consistent despite network noise.
2) We partition each flow into groups of consecutive packets so that performance and overhead of designed algorithms can be expressed with respect to the group size.

Our evaluation combines theoretical analysis with experimentation. For a given level of maximum network noise, we analytically characterize conditions under which the proposed algorithms are guaranteed to operate correctly in spite of network noise. We analytically derive conditions assuring correct operation of the designed algorithms and also evaluate the algorithms on realistic traffic traces. The algorithms successfully distribute the monitoring-task load without imposing significant computation or storage overhead.

While our paper focuses on exact reconstruction of flow metrics, a large body of related prominent work explore approximate solutions for scalable traffic monitoring. Even the relatively simple problem of computing the flow sizes for all flows turns greatly challenging when the number of flows becomes large. Although most flows are mice, i.e., have a small size, the lack of a priori knowledge about flow sizes forces a counting element to allocate space for all flows, and the element has insufficient memory on its data path to compute the flow sizes exactly for a large number of flows. Previous approximate solutions for the flow-size computation include estimators [1], [9]–[11] and sketches: CM [12], CU [13], Pyramid Sketch [14], UnivMon [15], and Elastic Sketch [2]. A trace-driven evaluation of CEDAR [1], SAC [11], and DISCO [10] shows their average relative errors in excess of 12% for 8-bit per-flow estimators [1]. Even Elastic Sketch, one of the most advanced current proposals, underestimates the sizes of mice flows by a factor of 4 when using 0.2 MB to represent 110K flows, i.e., around 15 bits per flow [2]. Such accuracy is insufficiently low for billing and other traffic-monitoring applications. For examples, if the network overcharges the sender of a flow by 10%, such inaccuracy is unacceptable in practice. Our paper pioneers an alternative distributed approach: instead of sacrificing the accuracy, we support scalable exact reconstruction of flow metrics by involving additional network elements that have spare resources.

The rest of the paper is organized as follows. Section II formulates the problem of robust traffic monitoring in the context of flow-size computation. Section III discusses implications of network noise. Section IV examines a split of distributed monitoring between two network elements. Section V considers alternative representations of network noise and assesses their design ramifications. Section VI extends the solution for a split of distributed monitoring between any number of network elements on the flow path as well as for multi-path flows. Section VII provides an evaluation study. Section VIII presents related work. Section IX concludes the paper by summing up its contributions.



(a) PR: $|f| = 10, c_1 = 10_2, c_2 = 01_2, c = c_2c_1 = 0110_2 = 6$

(b) PL: $|f| = 10, c_1 = 10_2, c_2 = 11_2, c = c_2c_1 = 1110_2 = 14$

Fig. 1: Vulnerability of the PR and PL approaches respectively to: (a) packet loss and (b) packet reordering.

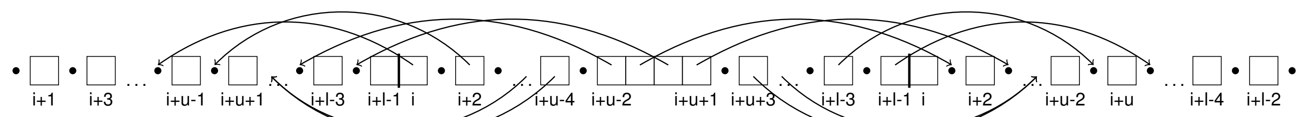
II. Model

We formulate the problem of robust traffic monitoring in the context of computing the flow sizes for all flows. For ease of exposition, this paper interchangeably refers to network elements as switches (even though the formulation is equally applicable to other types of network elements). Flow f enters the network in source switch S and exits in destination switch D. At switch S, the flow consists of $|f|$ packets p_0, p_1, …, $p_{|f|-2}$, $p_{|f|-1}$. While the objective is to compute flow size $|f|$ for all flows, switch S does not have enough resources to accomplish this task on its own and needs assistance from another switch. Switch i that participates in such distributed flow-size computation maintains counter chunk c_i consisting of n_i bits. Communication of flow state is in-band via packets of the monitored unidirectional flow. Switch i piggybacks at most t_i control bits on each packet. We refer to n_1 and t_1 as simply n and t respectively. Network noise might reorder or drop packets. When the flow terminates, the distributed algorithm should reconstruct $|f|$ exactly. While the flow is in progress, the estimates provided for the flow size by the algorithm should be non-decreasing and may not exceed the actual flow size.

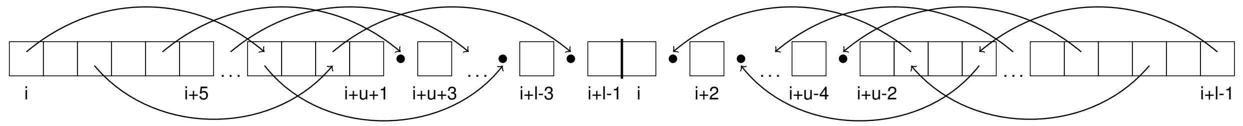
III. Implications of network noise

This section exposes ramifications of network noise for robust flow-size computation that is split between switches S and D. We do this for two extreme cases of network noise: (PR) Packet Reordering without loss and (PL) Packet Loss without reordering.

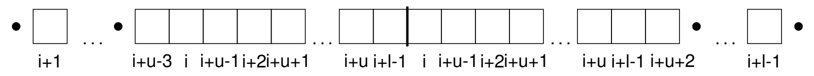
(PR) The following simple solution robustly computes the flow sizes under the PR kind of network noise: the source switch counts packets of the flow and, only when local counter chunk c_1 overflows, marks a control bit in the sent packet;



(a) Alteration of a 3-cycle state sequence by packet reordering and loss for even l and u



(b) Alteration of a 2-cycle state sequence by packet reordering and loss for even l and u



(c) The same packet sequence produced by the above sequence alterations

Fig. 2: State sequences in the proof of theorem 1.

the subsequent participating switch increments its counter chunk c_2 only upon receiving a packet with the marked control bit. This PR solution is fully resilient to packet reordering and greatly vulnerable to packet loss: it might massively underestimate the flow size upon losing a packet with the marked control bit. Figure 1a illustrates this vulnerability when a single packet loss causes the PR method to incorrectly compute the flow size as 6 instead of 10 packets.

(PL) For robust flow-size computation under the PL kind of network noise, the source switch can set a control bit in each sent packet to the most significant bit of local counter chunk c_1 and then increment c_1. The subsequent participating switch increments its counter chunk c_2 only upon receiving a packet where the control bit is set differently than in the previously received packet. This PL solution is resilient to loss of up to $2^{n-1} - 1$ consecutive packets, where n is the size of counter chunk c_1. However, the PL method is highly vulnerable to packet reordering, e.g., when packets $2^{n-1} - 1$ and 2^{n-1} of the flow arrive to the subsequent participating switch in reverse order, this switch incorrectly increments its counter chunk c_2 thrice instead of once: upon receiving packets 2^{n-1}, $2^{n-1}-1$, and $2^{n-1}+1$. For $n = 3$ bits, figure 1b illustrates the resulting overestimation: the PL method incorrectly computes the flow size as 14 instead of 10 packets.

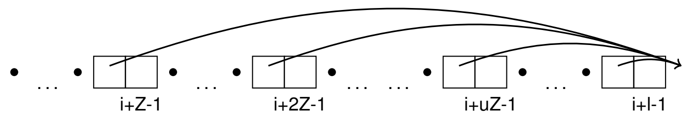
IV. Two-switch split

We now build on the above observations to robustly compute the flow size when network noise comprises both reordering and loss of packets. First, we consider a split of the distributed computation between two switches or, specifically, source and destination of the flow. Note that the flow traverses these two switches regardless of network routing. Resilience of any distributed solution is subject to fundamental feasibility limits, e.g., no such solution is able to handle loss of all packets. We employ two parameters to constrain packet reordering and packet loss: *reordering parameter R* captures the maximal distance of packet reordering, i.e., the destination switch can receive packet p_j before packet p_i only if $j \leq i + R$, and *loss parameter L* is the limit on *consecutive* packet losses, i.e., the destination switch receives at least one packet from any interval $p_i, \ldots, p_{i+L}$. For this $R\&L$ representation of maximum reordering and loss, we narrow down the feasibility limits for distributed flow-size computation as follows:

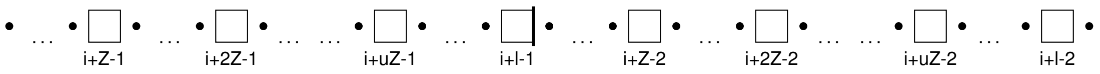
Theorem 1. *With a two-switch split where R and L bound packet reordering and loss respectively, no deterministic algorithm can guarantee correct distributed flow-size computation if $R > 2^{n-1}$ and $L > 0$.*

Proof. Let A be a distributed algorithm that computes the flow size on switches S and D. Algorithm A has n bits to store its counting state x_k for flow f on switch S, including the counter value and any auxiliary information. Upon receiving packet p, switch S updates its counting state to x_{k+1} and may modify p to communicate some state to switch D. We assume that A can record the entire state x_k into packet p. Consider a sequence of consecutive states $x_0, x_1, x_2, \ldots$ of algorithm A. Due to the n-bit representation, there are at most 2^n states. Hence, starting from some state, the counting follows a cycle of length $l \leq 2^n$, i.e., $x_{k+l} = x_k$ for any k.

First, assume that l is divisible by 4 and denote $u = \frac{l}{2}$. Figures 2a and 2b respectively show 3 and 2 cycles of length l in this sequence that are affected by packet reordering and loss. Switch D receives all other packets of the flow without any disruption. We use i to label the packets corresponding to state x_i, denote lost packets as black dots, and display packet reordering with arched arrows. In figure 2a, reordering shifts one half of the second-cycle packets to the first cycle and the other half to the third cycle. In figure 2b, reordering moves even-numbered packets from the first cycle forward by $u - 2$ positions and odd-numbered packets from the second cycle backward by $u - 2$ positions. These two alterations of two



(a) 1-cycle state sequence (reordering, loss)



(b) 2-cycle state sequence (loss)

Fig. 3: State sequences in the proof of theorem 3.

Algorithm 1 Two-switch computation of flow sizes

```
1:  function DIFFERENCE($a$,$b$,$t$)
2:      return $(a + 2^t - b \text{ mod } 2^t) \text{ mod } 2^t$
3:  procedure SOURCEUPDATE($p$)
4:      $h[p] \leftarrow \left\lfloor \frac{c_1}{2^{n-t}} \right\rfloor$
5:      $c_1 \leftarrow (c_1 + 1) \text{ mod } 2^n$
6:  procedure DESTINATIONUPDATE($p$)
7:      $\text{diff} \leftarrow \textbf{Difference}(h[p], c_2, t)$
8:      If $1 \le \text{diff} \le 2^{t-1}$ then $c_2 \leftarrow c_2 + \text{diff}$
9:  procedure TOTALCOUNT($c_1$, $c_2$)
10:     $c \leftarrow c_2 \cdot 2^{n-t}$
11:     $c \leftarrow c + \textbf{Difference}(c_1, c, n)$
```

different sequences produce the same packet sequence shown in figure 2c. Upon receiving this sequence, switch D is unable to distinguish whether it originated from the 3-cycle or 2-cycle state sequence in switch S. Thus, there is no guarantee that algorithm A counts the flow size correctly. Since both alterations of the sequence involve packet reordering with $R = u + 1 \le 2^{n-1} + 1$ and loss with $L = 1$, the above proof establishes the theorem for even u and l that is divisible by 4. For all other combinations of l and u, the proof remains almost the same. □

While theorem 1 establishes limits on robustness of distributed flow-size computation when R and L bound packet reordering and loss respectively, we now present algorithm 1 that achieves these limits. The algorithm employs *sync bits*, a common t-bit portion of the counter chunks in switches S and D, to synchronize the two counter chunks. The sync bits correspond to the t most significant bits in S's chunk c_1 and t least significant bits in D's chunk c_2, which means that they make up the middle bits in the resulting merged two-chunk counter c that counts up to 2^{n+n_2-t}. When switch S receives packet p, the switch writes the sync bits from c_1 into header $h[p]$ of packet p and increments its n-bit counter chunk c_1. When packet p arrives to switch D, the destination switch uses the DIFFERENCE procedure to compute, subject to modulo 2^t, the difference between $h[p]$ and the t sync bits in counter chunk c_2. If this difference is between 1 and 2^{t-1}, switch D adds it to c_2. To compute $|f|$, the TOTALCOUNT procedure of algorithm 1 sets the n most significant bits of counter c to c_2 and then adds to c the difference between c_1 and the n least significant bits of c.

Theorem 2. *Algorithm 1 correctly computes the flow size up to 2^{2n-t} packets under the following conditions:*

$$L + R < 2^{n-1} \quad \textit{and} \quad R \le 2^{n-1} - 2^{n-t}. \tag{1}$$

Correctness of theorem 2 directly follows from theorems 4 and 6 which we state and prove later in the paper. Parameter t controls a trade-off between the constraint on R and the size of counter chunk c_2 in destination switch D. Note that constraint $L+R < 2^n-1$ is independent of t. If $R = 0$, i.e., the network does not reorder packets, algorithm 1 counts the flow size correctly under loss of up to $L < 2^{n-1}$ consecutive packets. When $R > 2^{n-1}$, algorithm 1 offers no assurance of correct operation, which perfectly matches the infeasibility result of theorem 1.

In algorithm 1, counter chunk c_2 never decreases, and product $c_2 \cdot 2^{n-t}$ never exceeds the number of packets received by source switch S. Hence, $c_2 \cdot 2^{n-t}$ serves as a real-time lower bound on the number of packets that switch S has received so far.

Algorithm 1 is highly resilient to network noise in practical settings. For example, by allocating only $n = 8$ bits for the counter chunk in switch S and using $t = 2$ of them as sync bits, algorithm 1 guarantees its correct computation of the flow size under any reordering $R < 64$ packets and $L + R < 128$ packets, which constitute significant levels of network noise. Moreover, doubling the value of t from 2 to 4 bits raises the value of R to 112 packets.

V. IMPACT OF NOISE REPRESENTATION

Whereas section IV shows that robust distributed flow-size computation is feasible even under severe packet reordering and loss, we now study how the network-noise representation affects the feasibility. Specifically, we consider an alternative representation that bounds not only stretch but also frequency of network noise: we partition the flow at switch S into groups of 2^k consecutive packets and introduce $\text{span}(\gamma, k)$ as a packet sequence where each packet is followed by a packet from the same group or γ subsequent groups. Formally, $\text{span}(\gamma, k)$ is defined as a sequence of packets p_{z_i} from flow f with the following properties:

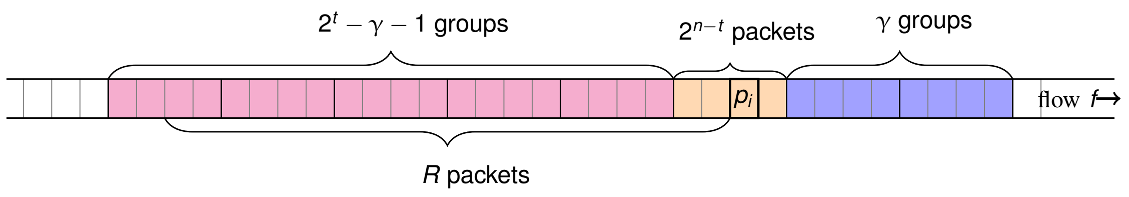


Fig. 4: Packet subsequences of the flow in its source switch S from the proof of correctness for algorithm 2.

(1) $k < n$, $1 \le \gamma < 2^{n-k}$, and $i = 0, \ldots, r-1$;
(2) indices z_i form an increasing sequence;
(3) switch D receives all packets p_{z_i};
(4) D receives p_{z_i} before p_{z_j} if $i < j$;
(5) $\left\lfloor \frac{z_{i+1}}{2^k} \right\rfloor - \left\lfloor \frac{z_i}{2^k} \right\rfloor \le \gamma$ for every i, i.e., $p_{z_{i+1}}$ is at most γ groups of 2^k packets away from p_{z_i} at switch S;
(6) $\left\lfloor \frac{z_0}{2^k} \right\rfloor \le \gamma$ and $\left\lfloor \frac{f}{2^k} \right\rfloor - \left\lfloor \frac{z_{r-1}}{2^k} \right\rfloor < 2^t$.

If $\mathrm{span}(\gamma, k)$ exists, condition 5 ensures that packet reordering and loss cannot affect all $\gamma \cdot 2^k$ packets from γ consecutive groups. With this representation of network noise, the maximum reordering extent is still R, and packet loss is arbitrary as long as switch D still receives $\mathrm{span}(\gamma,\ k)$. Hence, packet reordering and loss may exceed the network-noise limits examined in section IV. Also, parameter γ in this new network-noise representation controls a different trade-off between packet reordering and loss.

Theorem 3. *With a two-switch split where R bounds packet reordering, and packet loss preserves $\mathrm{span}(\gamma,\ n-t)$, no deterministic algorithm can guarantee correct distributed flow-size computation if $R > 2^n - \gamma \cdot 2^{n-t}$.*

Proof. We pursue a similar approach as in the proof of theorem 1 and construct two state-alteration sequences that produce the same packet sequence. For $Z = \gamma \cdot 2^{n-t}$ and $u = \left\lfloor \frac{l}{Z} \right\rfloor$ where $Z \le l$, figures 3a and 3b respectively show 1 and 2 cycles of length l that are affected by different kinds of network noise. Upon receiving this packet sequence, switch D cannot distinguish whether it originated from the 1-cycle or 2-cycle state sequence in switch S. Thus, no deterministic algorithm provides a guarantee of counting the flow size correctly. Since the two alterations have $R = l - Z + 1 \le 2^n - \gamma \cdot 2^{n-t} + 1$, keep a gap of at most $Z - 1$ between two consecutive packets in switch D, and thus conserve $\mathrm{span}(\gamma, n-t)$, we establish the theorem. For $l \mod Z = 1$, packets $i + uZ - 1$ and $i + l - 1$ are consecutive, and we have to relocate the latter packet to prove the theorem in this case. □

For $\gamma = 2^{t-1}$, the bound in theorem 3 is $R > 2^{n-1}$, i.e., the same as in theorem 1. Nevertheless, theorem 3 is not a generalization of theorem 1: the latter exacts more restrictive constraints. Algorithm 2 meets the limits stated in theorem 3. This algorithm sets k to $n - t$ and differs from algorithm 1 only in its DESTINATIONUPDATE procedure; the procedure now updates counter chunk c_2 only when switch D receives a packet from the next γ groups.

Algorithm 2 Robust two-switch computation of the flow size under the span constraint

```
1: procedure DestinationUpdate($p$)
2:     diff ← Difference($h[p], c_2, t$)
3:     If $0 < \text{diff} \le \gamma$ then $c_2 \leftarrow c_2 + \text{diff}$
```

Theorem 4. *Algorithm 2 correctly computes $|f|$ under the following conditions:*

$$R \le 2^n - (\gamma + 1) \cdot 2^{n-t} \quad \textit{and} \quad \exists\ \mathrm{span}(\gamma, n-t). \qquad (2)$$

Proof. We assert that packet p_i received by switch D updates counter chunk c_2 correctly if DESTINATIONUPDATE modifies c_2 to become $\left\lfloor \frac{i}{2^{n-t}} \right\rfloor$. By induction, we show that each update of c_2 is correct. By definition of $\mathrm{span}(\gamma,\ n-t)$, there exists $z_0 \le \gamma \cdot 2^{n-t}$ such that packet p_{z_0} arrives to switch D. The index of any packet received by D before p_{z_0} does not exceed $z_0 + R \le \gamma \cdot 2^{n-t} + 2^n - (\gamma + 1) \cdot 2^{n-t} < 2^n$, meaning that all such packets are from the first 2^t groups of size 2^{n-t} packets. Then, either these packets update c_2 correctly, or DESTINATIONUPDATE ignores them. Hence, the first update is correct.

For the induction step, we consider the last packet p_i that updates c_2 correctly and show that the next update of c_2 is correct. Since there exists $\mathrm{span}(\gamma, n-t)$, at least one packet from the subsequent γ groups arrives to D after p_i (the blue region in figure 4); we denote by p_j the first such packet. Due to the reordering constraint, any packet p_t received by D before p_j and after p_i is either from the $2^t - (\gamma + 1)$ groups that precede the group containing p_i (the pink region in figure 4) or from the $2^t - (\gamma + 1)$ groups succeeding p_j's group. Also, p_t cannot be from the γ groups following p_i's group since p_j is the first such packet. Therefore, DESTINATIONUPDATE ignores all such packets p_t and updates c_2 correctly when switch D receives packet p_j.

To finish the proof, we show that the final update of c_2 is done by a packet from a group with a larger number than $\left\lfloor \frac{|f|}{2^{n-t}} \right\rfloor - 2^t$. We use induction to prove that the following situation does not happen: for some packet p_{z_j} from $\mathrm{span}(\gamma, n-t)$, packet p_i with $i < z_j$ arrives to D after p_{z_j} and updates c_2. Thus, when the last packet $p_{z_{r-1}}$ from this span arrives to D, either $c_2 \ge \left\lfloor \frac{z_{r-1}}{2^{n-t}} \right\rfloor$, or $p_{z_{r-1}}$ updates c_2. □

Theorem 4 exposes a variety of trade-offs. As with our first representation of network noise, a larger t value increases the size of c_2 and raises the maximal tolerated reordering R.

The following theorem demonstrates that under the same reordering constraint, algorithm 2 accommodates less restrictive traffic patterns for a larger value of t.

Theorem 5. *If a flow satisfies conditions 2 in theorem 4 for n, t, and γ, the flow also satisfies conditions 2 for n, $t+1$, and $2\gamma+1$.*

Proof. Because $R \leq 2^n - (\gamma+1)\cdot 2^{n-t} = 2^n - ((2\gamma+1)+1)\cdot 2^{n-t-1}$, an increase of t does not change the packet-reordering constraint. The largest possible difference between the indices of two consecutive packets from span(γ, $n-t$) is less than $(\gamma+1)\cdot 2^{n-t} = ((2\gamma+1)+1)\cdot 2^{n-t-1}$. This implies that these packets form span($2\gamma+1$, $n-t-1$). □

Furthermore, larger values of t provide us with a better leverage over the trade-off between the conditions of theorem 4. Setting $\gamma = 1$ leads to maximal reordering $R = 2^n - 2^{n-t+1}$. With $\gamma = 2^{t-1}$, algorithms 1 and 2 are identical, and the following theorem shows that our first representation of network noise is a special case of the second one.

Theorem 6. *When $\gamma = 2^{t-1}$, conditions 1 imply conditions 2.*

Proof. Since $L+R < 2^{n-1} = \gamma 2^{n-t}$, the sequence of packets that updates the counter chunk in switch D in algorithm 1 forms span(2^{t-1}, $n-t$). □

Settings with $\gamma \geq 2^{t-1}$ are useful in situations where loss is more frequent than reordering. Such settings imply $R < \gamma 2^{n-t}$, i.e., they shift the restricted zone for packet p_j towards later packets. This means that packet loss can be a bigger threat for the existence of span(γ, $n-t$) than packet reordering. When span(γ, $n-t$) does not exist, algorithm 2 underestimates $|f|$, and we derive a bound on this underestimation:

Theorem 7. *When there is no span(γ, $n-t$), and the total of X packets are lost, algorithm 2 computes $|f|$ with an underestimation of at most $\frac{2^t X}{2\gamma+1-2^t}$ packets under the following conditions:*

$$R \leq 2^n - (\gamma+1)\cdot 2^{n-t} \quad \textit{and} \quad \gamma \geq 2^{t-1}. \tag{3}$$

Proof. Let p_i and p_j be two packets that consecutively update c_2 upon their arrival to switch D. The condition on R enforces $j > i$, implying that algorithm 2 never overestimates $|f|$.

If p_j's group is $a2^t + b$ groups after p_i's group, i.e., $\lfloor \frac{j}{2^{n-t}} \rfloor - \lfloor \frac{i}{2^{n-t}} \rfloor = a2^t + b$, switch D adds b to its counter c_2. Then, algorithm 2 can miss up to $a2^n$ packets in total. Conditions 3 imply that at least $a\gamma 2^{n-t}$ out of these $a2^n$ packets are lost or arrive before p_i. At most $2^n - (\gamma+1)2^{n-t}$ packets with larger indices than i can arrive before p_i. Thus, at least $(a-1)\gamma 2^{n-t} + 2\gamma + 1 - 2^t$ packets are lost, and the missed-to-lost packet ratio is at most $\frac{a2^n}{(a-1)\gamma 2^{n-t}+2\gamma+1-2^t}$, which is at most $\frac{2^t}{2\gamma+1-2^t}$. This characterization holds for flow f's entire duration, including before the first update and after the last update of counter chunk c_2. Therefore, algorithm 2 underestimates $|f|$ by at most $\frac{2^t X}{2\gamma+1-2^t}$ packets. □

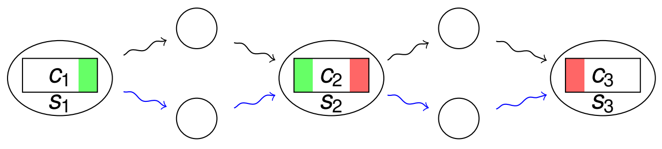


Fig. 5: Split of the size-counting state for flow f between three switches s_1, s_2, and s_3: counter chunks c_1 and c_2 overlap in their green portions; counter chunks c_2 and c_3 overlap in their red portions; packets can flow along either black or blue path segments.

When the constraint on R is satisfied, and there exists no span(γ, $n-t$), algorithm 2 with $t=2$ sync bits and $\gamma=2$ underestimates the flow size by at most $4X$ packets. With 3 sync bits and $\gamma = 6$, the underestimation is at most by $1.6X$ packets.

When $\gamma = 2^t - 1$, algorithm 2 does not tolerate any reordering and increases its loss resilience to $L < 2^n - 2^{n-t}$. When loss is unbounded, and there is no reordering, algorithm 2 with such values of γ underestimates the flow size by at most $\frac{X}{1-2^{-t}}$ packets.

From a practical perspective, conditions 2 make algorithm 2 more robust to reordering and loss than under conditions 1. For example, when the source-switch chunk contains 8 bits, the settings with 2 and 3 sync bits and $\gamma = 1$ support $R \leq 128$ packets and $R \leq 224$ packets and leniently require D's packet sequence to form span(1, 64) and span(1, 32) respectively. For $n = 8$ bits, $t = 3$ bits, and $\gamma = 6$, this flow-accounting solution calculates $|f|$ correctly when there exists span(1, 192) and $R \leq 32$ packets.

VI. Flow accounting on 3+ switches

While sections IV and V split traffic monitoring between two switches only, we now relax this constraint and suppose that a flow utilizes multiple paths between its source and destination and that all these paths contain a fixed subsequence of switches $S = s_1, s_2, \ldots, s_{m-1}, s_m = D$ that maintains a distributed flow counter.

To characterize the maximum network noise affecting the packet sequence that arrives to switch s_i, we define parameters R_i and γ_i similarly to R and γ respectively. Packet p_j arrives to s_i before packet p_k only if $j \leq k + R_i$, and the packet sequence forms span($\gamma_i, N_{i-1} - t_{i-1}$) where $N_i = n_i + \sum_{j=1}^{i-1} n_j - t_j$.

We then develop algorithm 3 for distributed computation of the flow size in such settings. Switch s_i stores the n_i least significant bits of $\lfloor \frac{c}{2^{N_i - n_i}} \rfloor$ in its counter chunk c_i. The above-defined N_i is a partial joint counter maintained by the first i switches, and 2^{N_m} bounds the size of the total joint counter, i.e., maximal supported flow size.

Algorithm 3 reuses the SourceUpdate and DestinationUpdate procedures of algorithm 2 to update counter chunks c_1 and c_m respectively. The InternalUpdate procedure updates the t_{i-1} least significant bits of each internal counter chunk c_i using $h[p]$ according to algorithm 2. Then, InternalUpdate stores in packet header $h[p]$ the t_i most significant bits of c_i.

To compute $|f|$, the extended TotalCount procedure iteratively assembles all the c_i counters. After c_{i-1} is processed, c becomes $|f| \mod 2^{N_{i-1}}$. When the algorithm processes c_i, InternalUpdate sets c equal to $c_i \cdot 2^{N_{i-1}-t_{i-1}}$ and increases this counter to make its N_{i-1} least significant bits equal to c_{prev}.

There is an important difference between the SourceUpdate and InternalUpdate procedures: SourceUpdate first stores the bits in $h[p]$ and then updates counter chunk c_i, while InternalUpdate first updates counter chunk c_i and only then sets the header. This is because invariants for c_1 and another counter chunk c_i with $i > 1$ are different: c_1 stores the n least significant bits of the *next* packet index, while another c_i uses its bits for the index of the last packet received by source switch S.

Theorem 8. *Algorithm 3 correctly computes the flow size up to 2^{N_m} packets when the following conditions hold for every non-source switch s_i with $i > 1$:*

$$\begin{cases} R_i < 2^{N_{i-1}} - (\gamma_i + 1) \cdot 2^{N_{i-1}-t_{i-1}} \text{ and} \\ \exists \operatorname{span}(\gamma_i, N_{i-1} - t_{i-1}) \text{ in the i-th switch.} \end{cases} \tag{4}$$

Proof. To prove that counter updates in switch s_i are correct, we view s_i as the destination switch for a two-switch counter with $R = R_i$, $\gamma = \gamma_i$, $t = t_i$, and the N_{i-1} least significant bits of the two-switch counter are supported by an imaginary source switch that combines counter chunks $c_1, \ldots, c_{i-1}$. We use the proof of theorem 4 as the induction step. The settings differ from theorem 4 in the following detail: while packet header $h[p_j]$ in theorem 4 is constructed from the bits of p_j, header $h[p_j]$ in this proof can correspond to the bits of a packet with a larger index than j. In this case, c_{i-1} is not updated, $h[p]$ does not update c_i, and counting is not affected. After TotalCount processes c_i, counter c becomes $|f| \mod 2^{N_i}$. By iterating the above two-switch proof technique for each $i > 1$, we prove the theorem. □

Many parameters characterize a split of flow-size computation between m switches. For the n_i-bit counter chunks c_i, we need to specify $t_1, \ldots, t_{m-1}$ and $\gamma_2, \ldots, \gamma_m$. In practice, it makes sense to use $t_i = 2$ for $i > 1$ and $\gamma_i = 2$ for $i > 2$. The most important decision is the choice of t_1 and γ_2 because the most frequent state updates go from S to s_2, and the update frequency declines exponentially on each subsequent hop.

For a numerical illustration, consider a split between 3 switches, where the first two switches maintain 7-bit counter chunks. We use $t_1 = 3$ sync bits, which implies $R_2 + 16 \cdot \gamma_2 \leq 112$ packets. Increasing t_1 does not significantly improves the situation because theorem 3 rules out existence of a deterministic algorithm for $R_2 + 2^{n-t}\gamma_2 > 128$ packets. We set $\gamma_2 = 5$, which leads to $R_2 = 40$ packets and the constraint that at least one out of 5 consecutive sets of 16 packets is not completely changed by packet reordering and loss. This

Algorithm 3 Computation of the flow size on 3+ switches

```
1: procedure SourceUpdate($p$)
2:     $h[p] \leftarrow \left\lfloor \frac{c_1}{2^{(n_1 - t_1)}} \right\rfloor$
3:     $c_1 \leftarrow (c_1 + 1) \mathbf{~mod~} 2^n$
4: procedure InternalUpdate($p$)
5:     diff $\leftarrow$ Difference$(h[p], c_i, t_{i-1})$
6:     If $0 <$ diff $\leq \gamma_i$ then $c_i \leftarrow (c_i + \text{diff}) \mathbf{~mod~} 2^{n_i}$
7:     $h[p] \leftarrow \left\lfloor \frac{c_i}{2^{n_i - t_i}} \right\rfloor$
8: procedure DestinationUpdate($p$)
9:     diff $\leftarrow$ Difference$(h[p], c_m, t_{m-1})$
10:    If $0 <$ diff $\leq \gamma_m$ then $c_m \leftarrow c_m +$ diff
11: procedure TotalCount($c_1, c_2, \ldots, c_m$)
12:    For $i \leftarrow 2$ to $m$ with step $1$
13:        $c_{\text{prev}} \leftarrow c$
14:        $c \leftarrow c_i \cdot 2^{N_{i-1} - t_{i-1}}$
15:        $c \leftarrow (c +$ Difference$(c_{\text{prev}}, c, N_{i-1})) \mathbf{~mod~} 2^{N_i}$
```

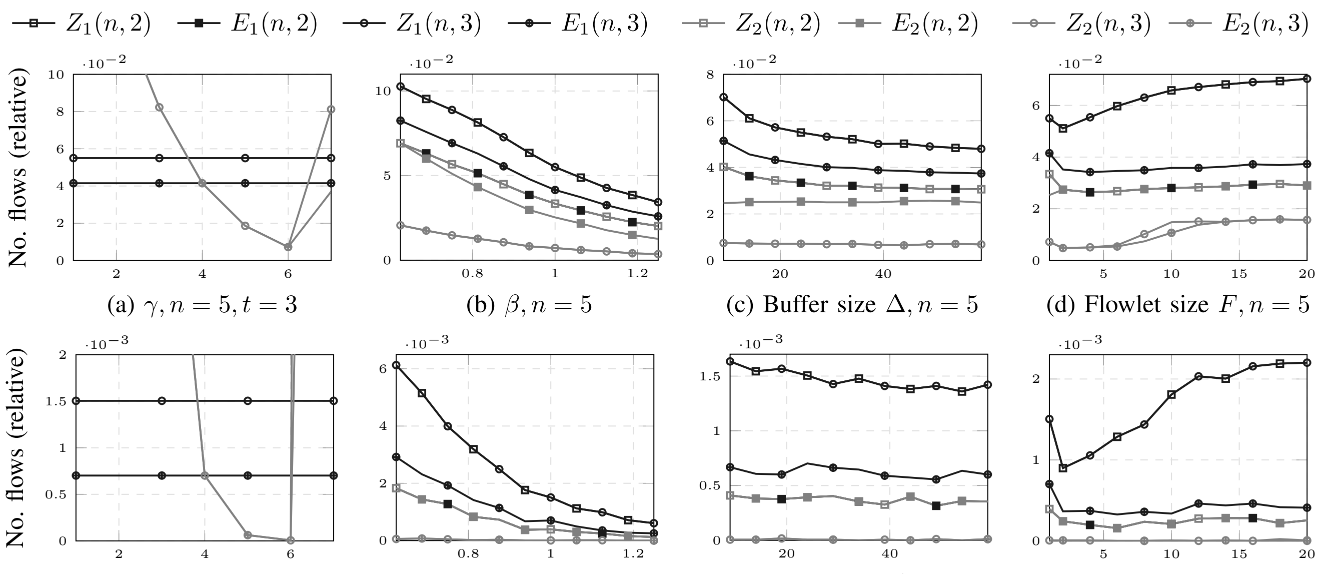
is reasonable in practical settings. For the first non-source switch s_2, even 2 sync bits and $\gamma_3 = 2$ produce $R_3 = 512$ packets and $2^{N_2 - t_2}\gamma_3 = 1024$ packets.

This example shows that communication quality on the first hop is the most important, whereas subsequent switches are amenable to more economical use of its resources because they typically do not face substantial network noise. This property holds in general, implying that the source switch should get the largest counter chunk, and the counter chunks on further switches can be smaller. For example, if we increase S' chunk to 9 bits and decrease s_2's chunk to 5 bits, the network-noise constraints for switch s_3 remain the same but the network-noise constraints on the first hop are significantly and usefully relaxed to $R_2 + \gamma_2 \cdot 64 \leq 448$ packets.

VII. Experimental Evaluation

Methodology. Our evaluation follows the same approach as those for VL2 [16], pFabric [17], and pHost [18]. Specifically, we perform simulations driven by realistic traffic traces generated from the data-mining distribution of flow sizes [16], where the number of flows is 10^6, and the maximum possible flow size of $\frac{2}{3} \cdot 10^6$ packets, which requires a 20-bit counter. We utilize the YAPS packet simulator in its unreliable transport configuration [19]. The network has a two-tier multi-rooted tree topology where four switches constitute the root. 90% of all flows traverse three internal switches on their way from the source to the destination. By default, YAPS sprays packets of each flow by probabilistically sending the packets to different internal switches in accordance with a chosen load-balancing strategy. The packet spraying causes packet reordering. The simulations examine the task of exact flow-size computation and have their source code publicly available [20].

We repeat the simulations for different congestion levels by varying parameter β that scales the expected time between the transmissions of two consecutive packets from the same source. Smaller values of β lead to larger congestion. We also evaluate different values of buffer size Δ in the switches



(a) $\gamma, n = 5, t = 3$ (b) $\beta, n = 5$ (c) Buffer size $\Delta, n = 5$ (d) Flowlet size $F, n = 5$

(e) $\gamma, n = 6, t = 3$ (f) $\beta, n = 6$ (g) Buffer size $\Delta, n = 6$ (h) Flowlet size $F, n = 6$

Fig. 6: Numbers Z_1 and Z_2 of flows violating conditions 1 and 2 respectively, and numbers E_1 and E_2 of flows with sizes calculated incorrectly by algorithms 1 and 2 respectively, as functions of γ, β, buffer size Δ, and flowlet size F; the source counter chunks are sized to: (a)-(d) $n = 5$ bits and (e)-(f) $n = 6$ bits.

along the flow paths. Flowlet size F parameterizes the packet-spraying strategy: for every group of F consecutive packets, the internal switch is chosen according to the round-robin strategy. Our *standard experiment* uses the following default parameter values: $\beta = 1$, $\Delta = 24$ packets, and $F = 1$ packet.

Number of bits in the source switch. To understand the size required for the counter chunks in source switches, we experiment with different values of β, Δ, and F. In these experiments, 7 bits for the source-counter chunk are sufficient, reducing the source counter-chunk size by 65% and 78% for 20-bit and 32-bit counters respectively. The experiments track the following metrics:

1) $Z_1(n, t)$ is the number of flows that violate conditions 1 with n-bit source counter chunks and t sync bits;
2) $E_1(n, t)$ refers to the number of flows for which algorithm 1 computes the flow size incorrectly;
3) $Z_2(n, t)$ denotes the number of flows that violate conditions 2;
4) $E_2(n, t)$ is the number of flows for which algorithm 2 miscalculates the flow size.

These metrics are normalized to the number of flows sized to at least 2^n packets. For other flows, the counter fits in the source switch entirely. Since 7-bit counter chunks on source switches are already sufficient, we compute the metrics for n of 5 and 6 bits. The number of sync bits t is 2 or 3. In the experiments where γ is not specified explicitly, we choose such a γ value that leads to the minimal value of the corresponding metric.

Dependency on γ. Figures 6a and 6e show how Z_2 and E_2 depend on γ for $t = 3$ in the standard experiment. The effect of packet loss is more pronounced than packet reordering, and Z_2 and E_2 decrease when γ increases up to 6. For $\gamma = 2^{t-1} = 7$, Z_2 and E_2 increase drastically because any packet reordering violates conditions 2. For $\gamma = 7$, we have $Z_2 > E_2$ since conditions 2 are usually violated by a packet reordering, which does not necessarily lead to an invalid counter value. For $\gamma \leq 6$, we have $Z_2 = E_2$ because conditions 2 are violated only by absence of $\mathrm{span}(\gamma, n - t)$, which always results in incorrect counting by algorithm 2.

For $n = 5$ and optimal γ, we have $Z_2(5, 3) = E_2(5, 3) = 7 \cdot 10^{-3}$, meaning that it is sufficient to maintain a 5-bit source-counter chunk for 99.3% of flows with at least 32 packets. For $n = 6$ and optimal γ, the outcome improves further as only one flow does not satisfy conditions 2.

In figures 6a and 6e, Z_1 and E_1 appear as horizontal lines because they do not depend on γ. For $n = 5$, we have $Z_1(5, 3) = 5.5 \cdot 10^{-2}$ and $E_1(5, 3) = 4.2 \cdot 10^{-2}$, i.e., 94.5% of the flows with at least 32 packets satisfy conditions 1, and algorithm 1 computes the flow sizes correctly for 95.8% of the flows. Hence, the number of flows for which algorithm 2 computes sizes incorrectly is 6+ times lower than for algorithm 1. For $n = 6$, this gap widens further as algorithm 1 errs for few hundreds of flows while algorithm 2 fails for only one flow. The difference observed for $Z_1(5, 3)$ vs. $Z_2(5, 3)$ with $\gamma = 4$ corroborates theorem 6 by showing that our second representation of network noise is more expressive than the first one: only 76% of the flows violating conditions 1 also violate conditions 2.

Dependency on β**.** Figures 6b and 6f exhibit how the examined metrics depend on β for buffer size $\Delta = 24$ packets and flowlet size $F = 1$ packet. As β increases, network congestion and all the metrics decrease. Specifically, as β grows to 1.25, $Z_2(5, 2)$ decreases by 40%, $Z_2(5, 3)$ plummets by a factor of two, $E_2(5, 2)$ drops by 50%, and $Z_2(6, 2)$ and

$E_2(6, 2)$ lower by 65%. For each evaluated setting of β, the number of flows that violate conditions 2 for $n = 6$ and $t = 3$ does not exceed 13. With $\beta \geq 1.1875$, the number of such flows is strictly zero, implying that 6-bit source-counter chunks are sufficient for all flows.

For any β value, equalities $Z_2(6, 3) = E_2(6, 3)$, $Z_2(6, 2) = E_2(6, 2)$, and $Z_2(5, 3) = E_2(5, 3)$ hold, and $\gamma = 2^t - 2$ is optimal for these metrics. This is because traffic patterns that violate existence of $\operatorname{span}(\gamma, n-t)$ in conditions 2 always lead to incorrect flow-size computation by algorithm 2. $E_1(5, 2) = Z_2(5, 2)$ and $E_1(6, 2) = Z_2(6, 2)$ arise because algorithms 1 and 2 are identical for $\gamma = 2^{t-1}$, which is the optimal γ value for $Z_2(5, 2)$ and $Z_2(6, 2)$.

On the other hand, $E_2(5, 2) < Z_2(5, 2)$ since the optimal γ for $E_2(5, 2)$ is 3 rather than 2. Under any packet reordering with this γ value, all flows violate the packet-reordering constraint in conditions 2, which does not necessarily cause incorrect computation of flow sizes by algorithm 2. Also, as t increases, the number of flows that violate conditions 1 does not change since the packet-reordering constraint holds for all flows even for $t = 2$, and the constraint on $L + R$ does not depend on t.

By comparing $E_1(5, 2)$ and $E_1(5, 3)$, we can see advantages provided by our second representation of network noise. Although conditions 1 loosen as t grows, the number of flows for which algorithm 1 computes incorrect sizes increases as t steps up from 2 to 3, i.e., $E_1(5, 3) > E_1(5, 2)$. This happens because flows that satisfy conditions 2 for $\gamma = 2^{t-1}$ and violate conditions 1 may violate conditions 2 for $t + 1$ sync bits and $\gamma = 2^t$. According to theorem 5, all such flows satisfy conditions 2 for $t + 1$ sync bits when γ is $2^t + 1$.

Dependency on buffer size Δ. Figures 6c and 6g plot the dependencies of the examined metrics on buffer size Δ for $\beta = 1$ and $F = 1$ packet. As the buffer size decreases, all the metrics rise but at lower rates than in response to changes in β. For $\Delta = 9$, $Z_1(5, 2)$ increases by 27%, $Z_2(5, 3)$ and $E_2(5, 3)$ grow by less than 1.5%, and $Z_1(6, 2)$ rises by 15%. For each evaluated β value, the number of flows that violate conditions 2 with $n = 6$ and $t = 3$ is smaller than 4.

In general, the plots in figure 6c are much smoother than in figure 6g because the absolute metric values are much higher for $n = 5$ bits, and the randomness in the number of flows that violate the corresponding conditions affects the curves less. A similar observation holds for the plots in figures 6b and 6d vs. figures 6f and 6h.

Dependence on flowlet size F. Figures 6d and 6h reveal effects of flowlet size F on the assessed metrics when β and Δ are 1 and 24 respectively. As F grows from 2 to 20 packets, the metrics increase, reflecting the increased network noise. When F reaches 20 packets, $Z_2(5, 3)$ and $E_2(5, 3)$ rise by more than thrice, $Z_1(6, 2)$ increases by about 2.5 times, and $Z_2(5, 2)$ and $E_2(5, 2)$ grow by only 5%. However, as F steps down from 2 packets to 1, the metrics surge abruptly, and $Z_2(5, 2)$ and $E_2(5, 2)$ reach their maximum values with $F = 1$ packet. This effect can be due to the increased probability of traffic loss when packets of the same flow follow a single path rather than multiple.

Also, as F increases, the impact of packet reordering becomes more perceptible. Unlike what we have seen before, $\gamma = 2$ with $F \geq 2$ is also optimal for $Z_2(5, 2)$ and $E_2(5, 2)$ because packet reordering becomes so common that algorithm 2 with parameters $n = 5$, $t = 2$, and $\gamma = 3$ starts to compute incorrect sizes for many flows. For the same reason, $Z_2(5, 3)$ and $E_2(5, 3)$ begin to grow quickly with $F = 8$ packets and have the optimal γ value of 5 (rather than 6) with $F = 14$ packets.

Accuracy of algorithm 2. When algorithm 2 computes the size of flow f incorrectly because $\operatorname{span}(\gamma, n-t)$ does not exist, theorem 7 states that the computed flow size with $\gamma \geq 2^{t-1}$ lies in interval $[|f| - \frac{2^t X}{2\gamma+1-2^t},\ |f|]$, where X is the number of lost packets. In the standard experiment with n of 5 or 6, t of 2 or 3, and $2^{t-1} \leq \gamma < 2^t - 1$, algorithm 2 can err only due to lack of $\operatorname{span}(\gamma, n - t)$. In such cases, the computed flow size for any flow belongs to interval $[|f| - X, |f|]$ with $n = 6$ and falls outside interval $[|f| - X, |f|]$ for at most 9 flows with $n = 5$. For $\gamma = 2^{t-1}$, algorithm 2 fails mostly due to excessive packet reordering and overestimates the flow size. With n of 5 or 6, t of 2 or 3, and $\gamma = 2^{t-1}$, the overestimation for 98% of all such flows is less than 2%.

The above evaluation suggests that the proposed distributed execution of traffic-monitoring tasks can significantly help in effective utilization of resources available in a network. In practice, a 7-bit (or even a 6-bit) per-flow source-counter chunk is sufficient, compared to the 32-bit counters used for exact computation of flow sizes in a single element. Our assessment under various settings of n, γ and t also shows that even in the case of 5-bit source-counter chunks, the distributed method correctly computes flow sizes for 99.3% of the flows that contain at least 2^5 packets. Besides, when algorithm 2 computes the size of flow f imprecisely, the computed flow size almost always lies in interval $[|f| - X, 1.02 \cdot |f|]$, where X is a number of lost packets. Finally, the empirical evaluation confirms the analytical advantages of our second representation of network noise.

VIII. Related work

Flow-size computation in a single switch. A long line of research deals with efficient state representation for flow-size computation in a single network element. To utilize SRAM memory effectively, [21]–[23] propose hybrid SRAM/DRAM counting architectures. These methods allocate in SRAM a small counter only for frequent updates and maintain the entire counter in slower but significantly bigger DRAM memory. In contrast, our approach does not require a big pool of additional cheaper memory and distributes the computation to leverage resources available elsewhere in the network.

SAC [11], DISCO [10], and CEDAR [1] calculate an approximate number of per-flow packets by probabilistically incrementing a counter, which allows reducing the number of counter bits for long flows. In SAC, the counter is split between the exponent and estimation parts. To increment a counter, SAC probabilistically increments the estimation part, and the increment probability depends on the exponent part. When the estimation part overflows, SAC increments the exponent. In DISCO and CEDAR, the entire counter corresponds to a counter value. For n-bit counters, CEDAR constructs variables as an array mapping values of a counter variable to real counter values, the increment probability is inversely proportional to the difference between two corresponding consecutive values in this array. Among all possible arrays, CEDAR finds one that minimizes the relative error.

Methods such as CounterBraids [24] and CounterTree [25] store counters for all flows in hash-based data structures. The idea of CounterBraids is based on sparse random graph codes. The scheme maintains all variables in a tree-like architecture where leaves correspond to less significant bits of the counter variable, and internal nodes correspond to most significant bits.

Flow-size computation is frequently tackled by sketch-based solutions. One of the first such solution is CounterMin (CM) [12]. A CM sketch is a table with r rows and w columns, where each row has a corresponding hash function mapping a flow to a cell that stores a corresponding variable. For an arriving packet, CM increments values of the corresponding variables in all rows. To estimate a counter, CM takes the minimum of the corresponding variables among all rows. Pyramid Sketch [14] combines the ideas of the CounterTree and CM sketches, reducing the number of bits in each cell of the sketch table. UnivMon [15] exploits a sketch hierarchy for different measurement tasks, such as heavy-hitter detection or moment estimation. Elastic Sketch [2] separates mice and elephant flows: mice flows are stored in a CM sketch, and elephant flows are stored in a hash table. Elastic Sketch uses the Ostracism principle to move counter variables between the CM sketch and hash table.

Network-wide flow-size computation. Focusing on the objective of minimizing the communication complexity, [26] detects network-wide heavy hitters in a model where switches report their local counters to a coordinator. FlowRadar [27] maintains a small efficient hash-based data structure in each switch to support storage of encoded flow information, including counters, and a controller can leverage its network-wide view on these data structures to decode the flow information precisely. The given paper extends our preliminary ideas reported in [28].

Distributed flow state that changes routing. DIFANE [29] and vCRIB [30] exploit switches in the network to enforce endpoint ties. They both route traffic through intermediate switches, deviating from the routing policy given by the users.

Distributed flow state that obeys routing. Distribution of static policy state over the flow path is already considered in [3]–[5]. These schemes are static and hence immune to network noise.

Telemetry. Telemetry of various characteristics such as real-time packet loss inherently involves distributed flow state based on the number of packets in the flow source and destination. Prior telemetry solutions [31], [32] require an additional out-of-band communication channel, depend on the quality of time synchronization, and do not provide any analytical robustness guarantees. One can address these drawbacks by extending and applying our approach to real-time telemetry.

Relation to CRDT. Conflict-free replicated data type (CRDT) supports replicas in multiple network nodes without coordination and concurrency on updates [33], resolving inconsistencies by its mathematical properties. Since state-based CRDT (CvRDT) functions that merge states from the replicas must be commutative, associative, and idempotent, CvRDT is stable to reorderings in the sequence of update operations. A flow counter in S and D can be interpreted as a special case of a vector grow-only counter CvRDT [33], but our framework requires only a partial counter at S and D; unlike CRDT, network constraints now have to be incorporated into feasibility analysis.

IX. Conclusion

This paper pioneered a distributed approach to improving scalability of per-flow traffic monitoring. Instead of sacrificing the accuracy of traffic monitoring, the approach enabled scalable exact reconstruction of flow metrics by involving network elements that had spare resources. To make distributed execution of a stateful traffic-monitoring task robust against network noise, we adhered to the open-loop paradigm that introduced no extra packets, communicated flow state in-band by piggybacking few control bits on packets of the monitored flows, and kept latency low. Our methodology relied on two main design principles: use of state overlap in different elements to consistently represent distributed state despite network noise, and partitioning of each flow into groups of consecutive packets to express performance and overhead of designed algorithms in regard to the group size. We applied these design principles for robust exact computation of flow sizes for all flows. We analytically established conditions guaranteeing correct operation of the designed algorithms and complemented the analysis with simulations driven by realistic traffic traces. Our evaluation suggested that the distributed execution of traffic-monitoring tasks could successfully balance the monitoring load on the network without imposing significant storage or computation overhead. In the future work we plan to study applicability of the proposed design principles for telemetry tasks.

Acknowledgments

This work was made possible by grants from various organizations. The Cisco University Research Program Fund, an advised fund of Silicon Valley Community Foundation, supported Vitalii Demianiuk and Kirill Kogan. The Regional Government of Madrid enabled the research by Sergey Gorinsky and Kirill Kogan with grant P2018/TCS-4499, EdgeData-CM. The Russian Science Foundation provided grant no. 17-11-01276 to support the contributions by Sergey Nikolenko and, in particular, Theorems 1, 3, 5 and Section VI. We also thank the anonymous reviewers for their detailed comments and insightful suggestions.

References

[1] E. Tsidon, I. Hanniel, and I. Keslassy, "Estimators Also Need Shared Values to Grow Together." in *INFOCOM*, 2012, pp. 1889–1897.

[2] 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 *SIGCOMM*, 2018, pp. 561–575.

[3] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "One Big Switch" Abstraction in Software-Defined Networks," in *CoNEXT*, 2013, pp. 13–24.

[4] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing Tables in Software-Defined Networks," in *INFOCOM*, 2013, pp. 545–549.

[5] P. Chuprikov, K. Kogan, and S. Nikolenko, "How to Implement Complex Policies on Existing Network Infrastructure," in *SOSR*, 2018, pp. 9:1–9:7.

[6] A. Kesselman, K. Kogan, S. Nemzer, and M. Segal, "Space and Speed Tradeoffs in TCAM Hierarchical Packet Classification," *J. Comput. Syst. Sci.*, vol. 79, no. 1, pp. 111–121, 2013.

[7] K. Kogan, S. I. Nikolenko, P. T. Eugster, and E. Ruan, "Strategies for Mitigating TCAM Space Bottlenecks," in *HOTI*, 2014, pp. 25–32.

[8] S. Gorinsky, S. Jain, H. M. Vin, and Y. Zhang, "Design of Multicast Protocols Robust Against Inflated Subscription," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 249–262, 2006.

[9] A. Cvetkovski, "An Algorithm for Approximate Counting Using Limited Memory Resources," in *SIGMETRICS*, 2007, pp. 181–190.

[10] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, Y. Cheng, and H. Wu, "Discount Counting for Fast Flow Statistics on Flow Size and Flow Volume," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 970–981, Jun. 2014.

[11] R. Stanojevic, "Small Active Counters," in *INFOCOM*, 2007, pp. 2153–2161.

[12] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[13] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, Aug. 2003.

[14] Y. Tong, Z. Yang, J. Hao, C. Shigang, and L. Xiaoming, "Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1442–1453, Aug. 2017.

[15] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," in *SIGCOMM*, 2016, pp. 101–114.

[16] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *SIGCOMM*, 2009, pp. 51–62.

[17] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-Optimal Datacenter Transport," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

[18] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed Near-Optimal Datacenter Transport over Commodity Network Fabric," in *CoNEXT*, 2015, pp. 1:1–1:12.

[19] "YAPS: Yet Another Packet Simulator," http://wiki.github.com/NetSys/simulator/.

[20] "Robust Distributed Monitoring of Traffic Flows." https://github.com/RobustCounter/RoubstMonitoring.

[21] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a Statistics Counter Architecture," in *HOTI*, 2001, pp. 107–111.

[22] ——, "Maintaining Statistics Counters in Router Line Cards," *IEEE Micro*, vol. 22, no. 1, pp. 76–81, 2002.

[23] Q. Zhao, J. J. Xu, and Z. Liu, "Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency," in *SIGMETRICS/Performance*, 2006, pp. 323–334.

[24] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," in *SIGMETRICS*, 2008, pp. 121–132.

[25] M. Chen, S. Chen, and Z. Cai, "Counter Tree: A Scalable Counter Architecture for Per-Flow Traffic Measurement," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1249–1262, April 2017.

[26] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-Wide Heavy Hitter Detection with Commodity Switches," in *SOSR*, 2018, pp. 8:1–8:7.

[27] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers," in *NSDI*, 2016, pp. 311–324.

[28] V. Demianiuk, S. Gorinsky, S. I. Nikolenko, and K. Kogan, "Distributed counting along lossy paths without feedback," in *SIROCCO*, 2018, pp. 30–33.

[29] M. Yu, J. Rexford, M. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," in *SIGCOMM*, 2010, pp. 351–362.

[30] M. Moshref, M. Yu, A. B. Sharma, and R. Govindan, "vCRIB: Virtualized Rule Management in the Cloud," in *HotCloud*, 2012.

[31] G. Fioccola, A. Capello, M. Cociglio, L. Castaldelli, M. Chen, L. Zheng, G. Mirsky, and T. Mizrahi, "Alternate-Marking Method for Passive and Hybrid Performance Monitoring," RFC 8321, January 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8321

[32] T. Mizrahi, C. Arad, G. Fioccola, M. Cociglio, M. Chen, L. Zheng, and G. Mirsky, "Compact Alternate Marking Methods for Passive and Hybrid Performance Monitoring," IETF, October 2018. [Online]. Available: https://tools.ietf.org/html/draft-mizrahi-ippm-compact-alternate-marking-03

[33] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of Convergent and Commutative Replicated Data Types," Inria – Centre Paris-Rocquencourt ; INRIA, Research Report RR-7506, Jan. 2011. [Online]. Available: https://hal.inria.fr/inria-00555588