Robust Distributed Monitoring of Traffic Flows

Vitalii Demianiuk, Sergey Gorinsky, *Member, IEEE/ACM,* Sergey Nikolenko, and Kirill Kogan

***Abstract*—Unrelenting traffic growth, device heterogeneity, and load unevenness create scalability challenges for traffic monitoring. In this paper, we propose Robust Distributed Computation (RoDiC), a new approach that addresses 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 away from the overloaded element might be infeasible because execution on multiple elements is inherent in the task or requires at least partial participation by the designated overloaded element. Furthermore, distributed execution of a stateful task has to be resilient to network noise in the form of packet reordering and loss. The RoDiC approach relies on two main principles of packet grouping and state overlap to support exact robust distributed monitoring of traffic flows under network noise. RoDiC uses an open-loop paradigm that does not add any control packets, communicates flow state in-band by appending few control bits to packets of monitored flows, and keeps measurement latency low. We apply RoDiC to the problem of flow-size computation and discuss how to instantiate our general technique for real-time packet-loss telemetry. The paper develops robust algorithms, proves their correctness and performance properties, and reports an evaluation driven by realistic traffic traces. The RoDiC algorithms successfully distribute the monitoring-task load while keeping the memory and computation overhead low.**

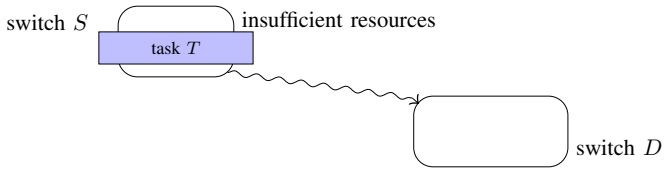
***Index Terms*—Traffic monitoring, distributed algorithm, stateful task, network noise, robust design, flow-size computation, real-time telemetry.**

I. Introduction

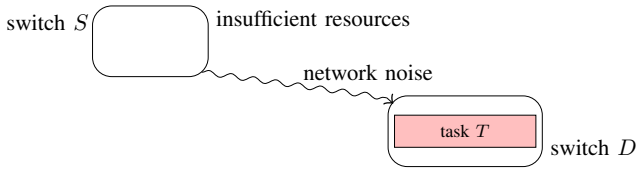
Network traffic monitoring underpins efficient, reliable, and secure operation of any network. Knowledge of traffic properties helps the network operator in planning capacity, providing quality of service, mitigating attacks, etc. Monitoring tasks include measurement of flow sizes [2]–[10] and real-time telemetry of network performance, e.g., loss experienced by flows between ingress and egress elements [11]–[14]. While such telemetry intrinsically requires distributed execution of the monitoring task in both ingress and egress elements of each flow, other tasks can be accomplished in a single element.

The work of Vitalii Demianiuk and Kirill Kogan was supported in part by the Israeli Innovation Authority under the Knowledge Transfer Commercialization Program (MAGNETON) file no. 71249, in part by the Ariel Cyber Innovation Center in cooperation with the Israel National Cyber Directorate in the Prime Minister's Office. The research by Sergey Gorinsky and Kirill Kogan was supported in part by the Regional Government of Madrid with grant P2018/TCS-4499, EdgeData-CM. The work of Sergey Nikolenko, in particular on theorems 1, 4, and 6 and section VI, was supported by the Russian Science Foundation grant no. 17-11-01276. A preliminary version of this article appeared in the proceedings of ICNP 2019 [1].

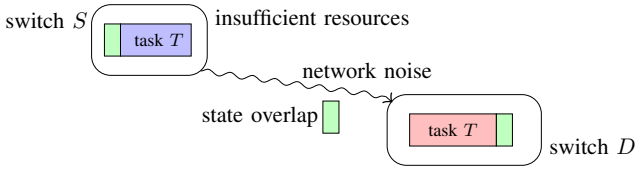
Vitalii Demianiuk is with Ariel University, 40700 Ariel, Israel (email: chavit92@gmail.com). Sergey Gorinsky is with IMDEA Networks Institute, 28918 Leganés, Spain (e-mail: sergey.gorinsky@imdea.org). Sergey Nikolenko is with the Steklov Institute of Mathematics, 191023 St. Petersburg, Russia, and National Research University Higher School of Economics, 194100 St. Petersburg, Russia (email: sergey@logic.pdmi.ras.ru). Kirill Kogan is with Ariel University, 40700 Ariel, Israel (e-mail: kirill.kogan@gmail.com).



(a) At designated switch S that has insufficient resources.



(b) Relocation of the task to switch D and network noise.



(c) Robust distributed execution by RoDiC at both S and D.

Fig. 1: Execution of a stateful monitoring task.

Challenges of scalable traffic monitoring include unrelenting traffic growth, device heterogeneity, and load unevenness. First, while traffic keeps growing in both volume and number of flows, the processing and memory 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 monitoring load on different network elements is uneven, and one element might get overloaded even when other elements have spare resources.

Even a relatively simple task, such as computing the sizes of all flows at a network element, puts the scarce data-plane resources under a strain when the number of flows becomes large. Figure 1a depicts an example where task T overwhelms the resources available at switch S. A prominent body of related work explores approximate solutions that give up accuracy of flow-size computation to reduce memory required in a single element. Such approximate solutions include estimators [2]–[5] and sketches: CM [6], CU [7], Pyramid Sketch [8], UnivMon [9], and Elastic Sketch [10]. A trace-driven evaluation of CEDAR [5], SAC [4], and DISCO [3] shows their average relative errors in excess of 12% for 8-bit per-flow estimators [5]. The average relative error of Elastic Sketch, one of the most advanced current proposals, can be equal to 4 even when using 0.2 MB to represent 110K flows, i.e., around 15 bits per flow, as figure 9a shows in [10]. Such accuracy might be insufficiently low.

An alternative technique for addressing the scalability challenge in a single network element is to monitor traffic flows by utilizing resources in multiple network elements as a shared pool. If a monitoring task overloads an element, then the task execution can be shifted from the overloaded element along the flow path to another element that has ample resources. Illustrating this approach for the above example, figure 1b shifts task T to switch D. The task relocation empowers the network to effectively leverage its global processing and memory resources and cope with local monitoring overloads.

Moving the entire execution of a monitoring task from a designated overloaded element to another element on the flow path can endanger the result accuracy due to *network noise* in the form of packet reordering and loss. As depicted in figure 1b, network noise might prevent switch D from executing task T as accurately as at switch S because the network might lose or reorder packets while delivering them from S to D. Such imprecision can arise regardless of whether the two network elements communicate over an unreliable or reliable transport protocol. Hence, the task execution should involve the originally designated element to at least some extent and be made resilient to network noise.

Our contributions. This paper proposes Robust Distributed Computation (RoDiC)[1], a general technique that executes a monitoring task in multiple network elements correctly despite network noise on the paths between the elements. For stateful tasks, the proposed technique not only maintains state in different elements but also resiliently communicates some state between these elements. RoDiC supports open-loop monitoring of flow metrics without introducing any control packets. Data packets of each monitored flow piggyback few (e.g., three) control bits to communicate flow state. This feature makes RoDiC applicable to unidirectional communications. In comparison to distribution of static policies [15]–[17], the proposed method incorporates additional mechanisms required to provide robustness. Instead of sacrificing accuracy as in scalable approximate solutions, the RoDiC technique enables scalable exact reconstruction of flow metrics.

While RoDiC is a general method that can be instantiated differently for different metrics, its two main design principles are as follows:

1) **Packet grouping:** RoDiC partitions each flow into groups of consecutive packets and distributes state so that a subsequent network element deals with the metric monitoring at the coarser granularity of packet groups.
2) **State overlap:** RoDiC maintains distributed state across multiple network elements in overlapping chunks. Sync bits in data packets communicate the chunk overlap to keep the distributed state consistent despite network loss.

Illustrating the RoDiC approach and its state-overlap principle, figure 1c shows robust distributed execution of task T at switches S and D. The state chunks at S and D overlap. S communicates the state overlap to D in-band.

We apply RoDiC to two problems in traffic monitoring. First, we focus on computation of flow sizes. Whereas the previous approximate solutions address lack of memory in a network element by sacrificing exactness of computation, RoDiC preserves the exactness by utilizing memory in multiple network elements. Second, we discuss how to instantiate RoDiC for real-time packet-loss telemetry, where distributed computation at both end elements of the path is inherent in the task. Our work combines theoretical analysis with experimentation. We analytically characterize correctness of the designed algorithms and evaluate them on realistic traffic traces. We show that our algorithms successfully distribute the monitoring-task load without imposing significant memory or computation overhead.

The paper has the following structure. Section II formulates the problem. Section III presents solutions to compute the flow sizes for specific kinds of network noise. Section IV examines robust distributed computation at two elements for a general model of network noise. Section V analyzes an alternative general model of network noise. Section VI considers robust distributed computation at three or more network elements. Section VII explains how to instantiate RoDiC for robust real-time telemetry of packet loss. Section VIII reports an evaluation study. Section IX discusses related work. Section X sums up the contributions of the paper.

II. Problem formulation

Flow f traverses a network via a series of switches[2]. At source switch S, the flow consists of $|f|$ network-layer packets numbered consecutively from 0 to $|f|-1$. Because we consider each flow on the network layer, the original transmission and retransmission of a data segment by the transport or application layers appear in the flow as distinct packets. Some of the subsequent switches assist switch S in distributedly computing a metric of the flow. Whereas the network might reorder or lose packets, the objective is to design a distributed algorithm that correctly computes the metric for all monitored flows despite such network noise. Each participating switch s_m uses n_m bits to maintain its chunk c_m of the distributed state. Switch s_m piggybacks at most t_m *sync bits* on each data packet to communicate control information to the subsequent participating switch[3].

III. Either reordering or loss

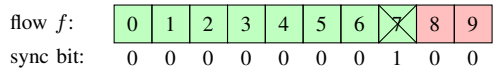
We start by examining the problem instance of distributedly computing the flow size, i.e., the flow metric is $|f|$, where source S and destination D are the only two switches participating in the computation. Note that each flow traverses its source and destination regardless of network routing. Initially, we consider two specific kinds of network noise: reordering only and loss only. For either kind, there is a simple robust algorithm that uses a single sync bit.

Reordering Only (RO): The RO algorithm follows only the principle of *packet grouping* and distributes the counting state between the two switches without overlap. The algorithm partitions flow f into groups of 2^n consecutive packets, with

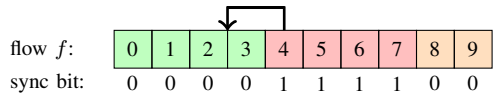
[1]Rodič means a parent in Czech and other Slavic languages.

[2]For ease of exposition, this paper interchangeably refers to network elements as switches. The formulation is equally applicable to other types of network elements.

[3]We refer to n_1 and t_1 as simply n and t respectively.



(a) RO: 010_2 at S, 000_2 at D, and $|f|$ as $000010_2 = 2$ packets.



(b) LO: $\mathbf{0}10_2$ at S, $10\mathbf{0}_2$ at D, and $|f|$ as $10\mathbf{0}10_2 = 18$ packets.

Fig. 2: Vulnerability of the RO and LO algorithms to respectively loss and reordering for $n = 3$ bits and $|f| = 10$ packets.

chunk c_1 at switch S counting packets within a group, and chunk c_2 at switch D counting the number of full groups. For each packet, S increments c_1 and, only when c_1 overflows, marks the sync bit in the packet. D increments c_2 only upon receiving a packet with the marked sync bit. When the flow terminates, the algorithm concatenates binary counters c_2 and c_1 to compute $|f|$. While immune to any packet reordering, the algorithm is vulnerable to losing a single packet. For $n = 3$ bits, figure 2a illustrates that when the network loses packet 7, the RO algorithm incorrectly computes $|f|$ as 2 instead of 10 packets.

Loss Only (LO): To combat packet losses, the LO algorithm complements packet grouping with the principle of *state overlap*. The most significant bit (MSB) of c_1 and least significant bit (LSB) of c_2 form a 1-bit overlap between the counting states at switches S and D. The 1-bit overlap represents the type of a packet group as even or odd. Each packet carries its group type in its sync bit. We refer to the other bits of each counting chunk as its *own bits*. S partitions flow f into groups of 2^{n-1} consecutive packets and uses the $n-1$ own bits of chunk c_1 to count packets within a group.

In the LO algorithm, S processes each packet by copying the group type from the MSB of c_1 into the sync bit of the packet and then incrementing c_1. Upon receiving a packet, D increments c_2 only if the group type in the LSB of c_2 differs from the sync bit in the packet, i.e., c_2 counts how many times the group type changes. Even without any network noise, the group type in the LSB of c_2 upon the flow termination can differ from the correct group type that c_1 captures in its MSB. For example, when the flow containing 2^{n-1} packets terminates, the group type in c_1 already became odd while the group type in c_2 remains even. If the group types in c_1 and c_2 differ when f terminates, the LO algorithm increments c_2. Then, the algorithm concatenates all bits of c_2 and $n-1$ own bits of c_1 to compute $|f|$.

Without packet reordering, the LO algorithm is assured to compute the flow size correctly if the network loses at most $2^{n-1}-1$ consecutive packets. However, reordering by only one packet can already lead to incorrect computation. For $n = 3$ bits, figure 2b shows that when the network reorders packets 3 and 4, switch D increments c_2 thrice (instead of once) upon receiving packets 4, 3, and 5, and erroneously computes $|f|$ as 18 instead of 10 packets.

IV. Computation on two switches

Our attention turns now to the general form of network noise that includes packet reordering and loss. We limit the solution space to deterministic algorithms, which always pass through the same sequence of states and produce the same output upon a particular input. Resilience of such algorithms is subject to fundamental feasibility limits, e.g., no solution is able to correctly handle loss of all packets.

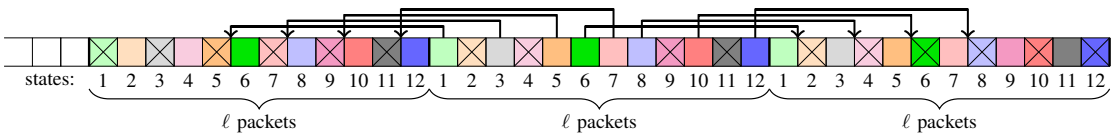
First, we consider a *consecutive-loss model* that constrains network noise via two parameters. *Loss parameter* L is the limit on *consecutive* packet losses, i.e., switch D receives at least one of packets i through $i+L$. *Reordering parameter* R captures the maximal distance of packet reordering, i.e., switch D can receive packet i before packet j only if $i \leq j+R$. We narrow down the feasibility limits on resilience in the consecutive-loss model as follows:

Theorem 1. *A deterministic RoDiC algorithm cannot guarantee correct distributed computation of the flow size on two switches if $L \geq 1$ packet and $R \geq 2^{n-1}+1$ packets.*

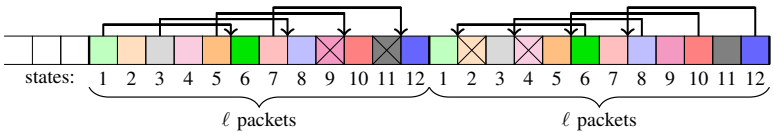
Proof. Given any deterministic RoDiC algorithm, we will construct two flows with different sizes at switch S and two flow-specific patterns of network noise that yield an identical sequence of counting states at switch D for both flows. Because the algorithm has n bits to store its counting state for a flow at switch S, there are at most 2^n such states, and – starting from some state – the counting follows a cycle of length l, which is at most 2^n states. Without loss of generality, we suppose that switch S records its entire current counting state into each packet of the flow.

First, when l is divisible by 4, consider flow A that contains $2^n + 3l$ packets. The last $3l$ packets of flow A carry the last three cycles of counting states for the flow at switch S. We label the packets/states in each cycle sequentially from 1 to l, denote $l/2$ as u, and construct the following network noise. The network loses all u odd-numbered packets in the first cycle and all u even-numbered packets in the third cycle. Each odd-numbered packet i among the first $\frac{l}{4}+1$ such packets in the second cycle arrives to switch D immediately before packet $i+u-1$ from the first cycle. Each even-numbered packet i among the last $\frac{l}{4}+1$ such packets in the second cycle arrives to switch D immediately after packet $i-u+1$ from the third cycle. The network loses the other $u-2$ packets from the second cycle. All the other packets of the flow arrive in order. For $l = 12$ states, figure 3a depicts the last $3l$ packets of flow A at switch S with the network noise superimposed on them.

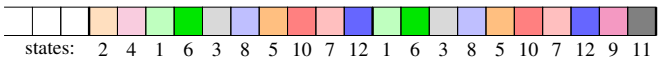
Now, compose flow B from the first $2^n + 2l$ packets of flow A, i.e., flow B is l packets shorter. The network noise impacts the last $2l$ packets of flow B and corresponding last two cycles of counting states as follows. The network loses the last $\frac{l}{4}-1$ odd-numbered packets in the first cycle and first $\frac{l}{4}-1$ even-numbered packets in the second cycle. In the first cycle, each odd-numbered packet i among the first $\frac{l}{4}+1$ such packets arrives to switch D right before packet $i+u-1$. In the second cycle, each even-numbered packet i among the last $\frac{l}{4}+1$ such packets arrives to switch D immediately after



(a) The last $3l$ packets of flow A at switch S with the superimposed network noise.



(b) The last $2l$ packets of flow B at switch S with the superimposed network noise.



(c) The identical packet sequence received by switch D from flows A and B.

Fig. 3: The impact of network noise on flows A and B in the proof of theorem 1 for $l = 12$ states.

Algorithm 1 Computation of the flow size on two switches in the consecutive-loss model

procedure INITIALIAZTION
  $c_1 \leftarrow 0$; $c_2 \leftarrow 0$
procedure SOURCE(j)
  $h[j] \leftarrow \lfloor \frac{c_1}{2^{n-t}} \rfloor$; $c_1 \leftarrow (c_1 + 1) \bmod 2^n$
procedure DESTINATION(j)
  diff $\leftarrow$ **Overshoot**($h[j]$);
  if $1 \leq \text{diff} \leq 2^{t-1}$ **then** $c_2 \leftarrow c_2 + \text{diff}$
procedure TERMINATION
  $c_2 \leftarrow c_2 + \textbf{Overshoot}(\lfloor \frac{c_1}{2^{n-t}} \rfloor)$;
  $|f| \leftarrow c_2 \cdot 2^{n-t} + c_1 \bmod 2^{n-t}$
function OVERSHOOT(g)
  return $(g - c_2 \bmod 2^t + 2^t) \bmod 2^t$

packet $i - u + 1$. All the other packets of the flow arrive in order. Figure 3b illustrates how the network noise affects the last $2l$ packets of flow B at switch S when l equals 12 states.

From either flow A or B, switch D receives an identical sequence of 2^n+3u+2 packets with the same counting states. Figure 3c depicts the last $3u + 2$ packets of this identical received sequence for $l = 12$ states. Because flows A and B have different sizes, D cannot guarantee correct distributed computation of the flow size.

The constructed network-noise patterns include consecutive loss of only one packet and reordering by at most $u + 1$ packets. Since $u = l/2$ is at most 2^{n-1} packets, correct flow-size computation cannot be assured if $L \geq 1$ packet and $R \geq 2^{n-1} + 1$ packets. When l is not divisible by 4, we use similar constructions to establish the theorem. □

Theorem 2. *A deterministic RoDiC algorithm cannot guarantee correct distributed computation of the flow size on two switches if $L \geq 1$ packet and $L + R \geq 2^n$ packets.*

Proof. Taking the same general approach as in the proof of theorem 1, we construct two network-noise patterns and apply them to two flows with different sizes so that switch D receives an identical packet sequence from either flow. Recall that starting from some state, the counting by a deterministic RoDiC algorithm follows a cycle of length l, which is at most 2^n states.

Flows A and B contain $2^n + 2l$ and $2^n + l$ packets respectively. When L is at least l, we achieve the construction goal by simply discarding the last l packets of flow A. When L is between 1 and $2^{n-1} - 1$ packets, the $L + R$ constraint implies $R \geq 2^{n-1} + 1$ packets, and theorem 1 establishes the result.

Finally, consider L between 2^{n-1} and $l - 1$ packets. For flow A, the network loses packet $2^n + L - 1$ and last l packets of the flow except packet $2^n + l + L - 1$. For flow B, the network reorders packet $2^n + L - 1$ to deliver it last. The network delivers all the other packets of the two flows in order. D receives an identical sequence of packets with the same counting states from either flow A or shorter flow B and cannot guarantee correct computation of the flow size. The constructed network-noise patterns include consecutive loss of at most $\textbf{max}\{L-1, l-L\} \leq L$ packets and reordering by at most $l - L$ packets. Hence, $L + R$ is at most l, i.e., 2^n packets, and this establishes the theorem. □

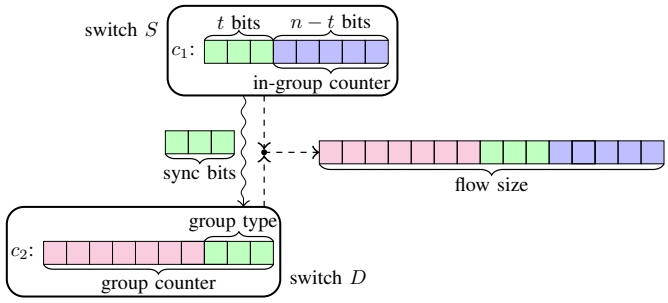


Fig. 4: Applying the RoDiC principles of packet grouping and state overlap to the problem of flow-size computation.

While theorems 1 and 2 detect feasibility limits for robust distributed flow-size computation, we now derive a RoDiC algorithm that approaches these limits. This algorithm 1 gen-

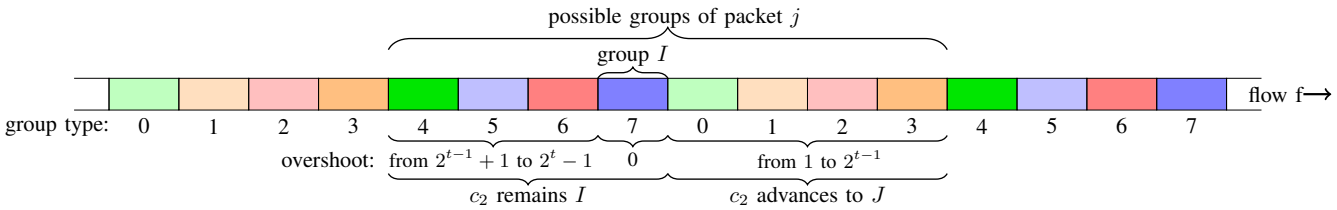


Fig. 5: Operation of the DESTINATION procedure in algorithm 1 when $t = 3$, $c_2 = I$, and packet j arrives to D from group J.

eralizes the LO algorithm from section III and follows both principles of packet grouping and state overlap as depicted in figure 4. The t MSBs of c_1 and t LSBs of c_2 form a t-bit overlap between the counting states at switches S and D. These t bits specify the type of a group and numbers the group types sequentially from 0 to $2^t - 1$. The group size becomes $n - t$ bits. Switch S uses its own $n - t$ LSBs of c_1 to count packets within a group. Counter c_2, including its t LSBs which capture the group type, tracks the largest group number among the groups that have delivered a packet to D.

In algorithm 1, S copies the group type from the t MSBs of c_1 into sync bits $h[j]$ of each packet j and then increments c_1 to count this packet. Upon receiving packet j, D uses arithmetic modulo 2^t to compute by how many groups the group type in $h[j]$ overshoots the group type in the t LSBs of c_2. For example, with $t = 3$ bits, there are eight group types numbered from 0 to 7, and type 4 overshoots type 7 by 5 groups. In the absence of network noise, the overshoot is either 0 or 1 group. With network noise, the overshoot can become as large as $2^t - 1$ groups. When the overshoot is 0 or between $2^{t-1} + 1$ and $2^t - 1$ groups, D ignores packet j because algorithm 1 assumes that c_2 has already accounted for the group of packet j. When the overshoot is between 1 and 2^{t-1} groups, the algorithm assumes that packet j arrives from a later, yet unaccounted group, and D advances c_2 by the amount of the overshoot. For instance, if t equals 3 bits, the group type in $h[j]$ is 1, and $c_2 = 7$, the overshoot is 2 groups, and D advances c_2 to 9. When the flow terminates, algorithm 1 calculates by how many groups the group type in c_1 overshoots the group type in c_2. Then, the algorithm advances c_2 by the amount of the overshoot and concatenates all bits of c_2 and $n - t$ own bits of c_1 to compute the flow size.

Theorem 3. *Algorithm 1 is guaranteed to compute the flow size on two switches correctly if*

$$R \leq 2^{n-1} - 2^{n-t} \text{ packets and } L+R \leq 2^{n-1} - 1 \text{ packets.} \quad (1)$$

Proof. Express $|f|$ as $K \cdot 2^{n-t} + m$ packets where K refers to the count of full groups, and m is between 0 and $2^{n-t} - 1$. Because network noise does not affect computations at switch S, c_1 always correctly captures m and the group type of K, in its $n - t$ LSBs and t MSBs respectively. Hence, we will prove the following two properties: (α) upon receiving a packet, switch D updates c_2 to correctly track the largest group number among the groups that have delivered a packet to D, and (β) when the flow terminates, algorithm 1 updates c_2 to capture group count K correctly.

First, we prove property α and illustrate this proof for $t = 3$ bits in figure 5. Suppose $c_2 = I$ where I is either initial value 0 or the largest group number among the groups that have delivered a packet to D. Upon receiving packet j from group J, switch D executes the DESTINATION procedure that computes by how many groups the group type in $h[j]$ overshoots group type $I \bmod 2^t$ in c_2. If $J \leq I - 1$, then packet j can precede group I at S by at most R packets, J is between $I - 2^{t-1} + 1$ and $I - 1$ due to the R constraint in conditions 1, the overshoot is between $2^{t-1} + 1$ and $2^t - 1$ groups, and D does not change c_2. If $J = I$, then the overshoot equals 0, and D does not change c_2 either. If $J \geq I + 1$, then packet j is the first packet that D receives from a group numbered larger than I. Hence, at most $L+R$ packets separate group I and packet j at switch S (in the worst case, the network loses the first L of these packets and delivers the next R packets to D after packet j), J is between $I+1$ and $I+2^{t-1}$ due to the $L + R$ constraint in conditions 1, the overshoot is between 1 and 2^{t-1} groups, and c_2 advances from I to J to correctly track the largest group number among the groups that have delivered a packet to D.

To prove property β upon the flow termination, consider $c_2 = I$ where I either captures the largest group number among the groups that have delivered a packet to D or equals 0 if D has not received any packets from the flow. Then, $|f|$ is at least $I \cdot 2^{n-t}$ packets because the flow contains at least I full groups 0 through $I - 1$. On the other hand, $|f|$ is at most $(I + 1) \cdot 2^{n-t} + L + R$ packets, which occurs when the flow contains $I + 1$ full groups 0 through I followed by $L + R$ lost packets (i.e., when the maximum network noise includes loss only with $R = 0$ and $L = L + R$). Due to the $L + R$ constraint in conditions 1, $|f|$ is between $I \cdot 2^{n-t}$ and $(I + 2^{t-1}) \cdot 2^{n-t} + 2^{n-t} - 1$ packets. Hence, group count K is between I and $I + 2^{t-1}$, i.e., the overshoot is between 0 and 2^{t-1} groups. The TERMINATION procedure computes this overshoot and updates c_2 from I to K. □

Parameter t, which captures the number of sync bits, controls a trade-off between the communication overhead and assured reordering resilience of algorithm 1. With $t = 1$ bit, the algorithm is the same as the LO algorithm from section III and guaranteed to compute the flow size correctly when the network loses at most $2^{n-1} - 1$ consecutive packets and does not reorder any packets. When t increases from 1 to 2 bits, the assured reordering resilience jumps to 2^{n-2} packets. Subsequent increments of t exhibit a diminishing marginal utility: each such increment cuts by half the number of packets added to the assured reordering resilience. When t is n bits,

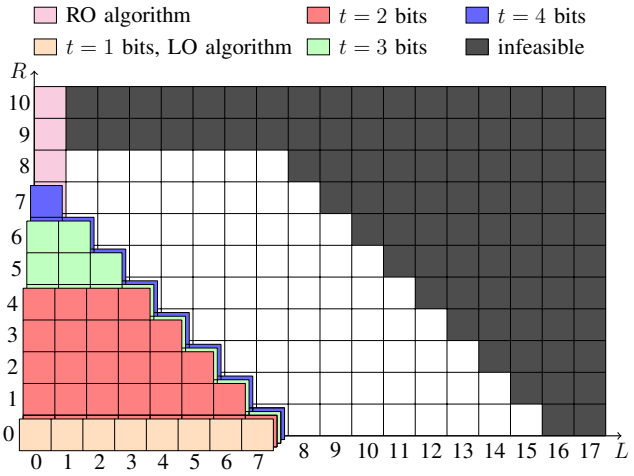


Fig. 6: Assured resilience of algorithms 1, LO, and RO to network noise in the consecutive-loss model and feasibility limits established by theorems 1 and 2 for $n = 4$ bits.

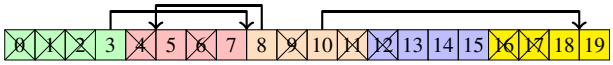
i.e., c_1 uses all its bits to track the group type only, the assured reordering resilience is $2^{n-1} - 1$ packets, i.e., not even twice larger than with $t = 2$ bits. Hence, we recommend setting t to 2, 3 or 4 bits in practice.

As t increases, the correct computation by algorithm 1 is assured for a strictly expanding set of (L, R) values. If the algorithm is guaranteed to compute the flow size correctly for specific values of L and R, then the guarantee also holds with any larger t. Figure 6 illustrates this property of algorithm 1 when n equals 4 bits. The figure also depicts the (L, R) settings where the RO and LO algorithms are guaranteed to operate correctly, as well as the (L, R) feasibility limits established by theorems 1 and 2.

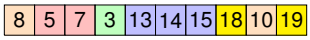
Even small numbers of sync bits support high levels of assured resilience to network noise. For example, $n = 8$ bits and $t = 2$ bits ensure correct computation with R by up to 64 packets and $L + R$ up to 127 packets. Doubling t to 4 bits increases the assured reordering resilience of algorithm 1 to 112 packets.

V. Impact of Noise Representation

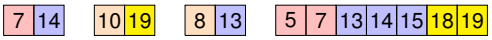
While the consecutive-loss model from section IV represents maximum network noise in terms of its local impact on a flow, this section considers an alternative representation linked to global properties of the packet sequence received by switch D from the flow. Inheriting the constraint of reordering by at most R packets, the new model does not impose an explicit limit on consecutive packet loss and instead characterizes the received packet sequence via a *span* notion. Defined in regard to the partition of flow f at switch S into groups of 2^k consecutive packets, a $\mathrm{span}(\gamma, k)$ refers to a subsequence of the packets at S that arrive to D in order, omit at most γ full groups at the beginning of the flow, and miss at most $\gamma - 1$ full consecutive groups after any packet in this subsequence. Figure 7 depicts four instances of a $\mathrm{span}(2, 2)$ for a flow that consists of 20 packets. The *span-based model* constrains network noise via reordering parameter R and span existence.



(a) The flow at switch S and network noise.



(b) The packet sequence received by switch D.



(c) Four instances of a $\mathrm{span}(2, 2)$.

Fig. 7: Span examples for a flow of 20 packets.

A $\mathrm{span}(\gamma, k)$ is also a $\mathrm{span}(\gamma+1, k)$. Hence, infeasibility of assuredly correct flow-size computation despite existence of a $\mathrm{span}(\gamma, k)$ implies such infeasibility with existing spans that have larger γ values. It also implies that enabling of assuredly correct computation necessitates existence of a span with a smaller γ value. Without packet reordering, the span-based model of maximum network noise comes with the following feasibility limits.

Lemma 1. *When k is at least 1 bit, existence of a $\mathrm{span}(2^{n-k}, k)$ is insufficient for a deterministic RoDiC algorithm to guarantee correct distributed computation of the flow size on two switches if $R \geq 0$.*

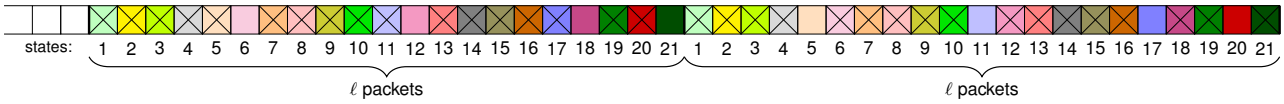
Proof. We follow the same general approach as in the proof of theorem 1. Starting from some state, the counting by a deterministic RoDiC algorithm follows a cycle of length l, which is at most 2^n states. Flows A and B contain $2^n + 1 + l$ and $2^n + 1$ packets respectively. The network loses the last l packets of flow A and delivers all other packets of both flows in order. D receives an identical sequence of $2^n + 1$ packets with the same counting states from either flow A or shorter flow B and cannot guarantee correct computation of the flow size. Because packet 2^n is the last packet received by D, and k is at least 1 bit, the group of packet 2^n in flow A also includes packet $2^n + 1$, which is lost. Since l is at most 2^n, the l lost packets at the end of flow A contain at most $2^{n-k} - 1$ full groups. Thus, the first $2^n + 1$ packets of flow A form a $\mathrm{span}(2^{n-k}, k)$ for both flows A and B, and there is no reordering. □

Due to the infeasibility result in lemma 1, we subsequently consider only spans where γ is at most $2^{n-k} - 1$ groups. With reordering, the infeasibility limits become as follows.

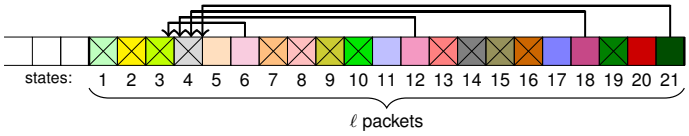
Theorem 4. *Existence of a $\mathrm{span}(\gamma, k)$ is insufficient for a deterministic RoDiC algorithm to guarantee correct distributed computation of the flow size if $R \geq 2^n - \gamma \cdot 2^k + 1$ packets.*

Proof. Using once again the technique from the proof of theorem 1, suppose that the counting cycle of a deterministic RoDiC algorithm has length l, which is at most 2^n states. Flows A and B contain $2^n + 2l$ and $2^n + l$ packets respectively.

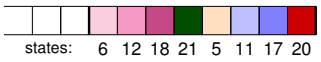
Let $Z = \gamma \cdot 2^k$ refer to the number of packets in γ full groups. When $l \bmod Z$ is at least 2 packets, consider the last $2l$ packets of flow A, which carry the last two cycles of counting states. We label the packets/states in each cycle sequentially from 1 to l and denote $\lfloor l/Z \rfloor$ as u. From the



(a) The last $2l$ packets of flow A at switch S with the superimposed network noise.

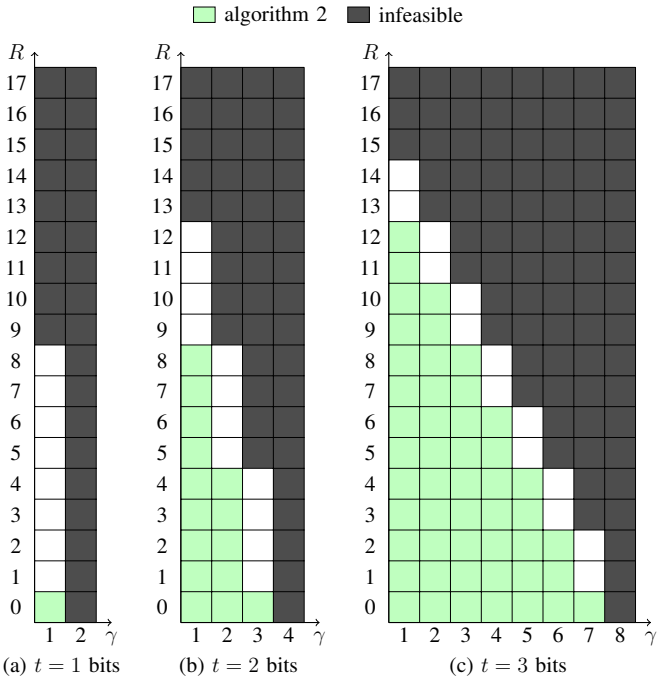


(b) The last l packets of flow B at switch S with the superimposed network noise.



(c) The identical packet sequence received by switch D from flows A and B.

Fig. 8: The impact of network noise on flows A and B in the proof of theorem 4 for $l = 21$ states, $\gamma = 3$ groups, and $k = 1$ bit.



(a) $t = 1$ bits (b) $t = 2$ bits (c) $t = 3$ bits

Fig. 9: Assured resilience of algorithm 2 to network noise in the span-based model and feasibility limits established by lemma 1 and theorem 4 for $n = 4$ bits.

first cycle, D receives packet l and u packets i such that i is divisible by Z. From the second cycle, D receives packet $l-1$ and first u packets j such that $j+1$ is divisible by Z. The above packets as well as first 2^n packets of flow A arrive to D in order. The network loses all the other packets of the flow. For $l = 21$ states, $\gamma = 3$ groups, and $k = 1$ bit, i.e., $Z = 6$ packets, figure 8a depicts the last $2l$ packets of flow A at switch S with the packet losses superimposed on them.

Flow B consists of the first 2^n+l packets of flow A. Among the last l packets of flow B, packet $l-1$ and first u packets j such that $j+1$ is divisible by Z arrive to D in order, packet l and u packets i such that i is divisible by Z arrive immediately before packet $\mathbf{min}\{Z, l\} - 1$ in their original order, and the other $l-2u-2$ packets are lost. The network delivers the first 2^n packets of flow B in order. For the above example with $l = 21$ states, figure 8b shows the last l packets of flow B at switch S with the network noise superimposed on them.

From either flow A or B, switch D receives an identical sequence of 2^n+2u+2 packets with the same counting states. Figure 8c depicts the last $2u + 2$ packets of this identical received sequence for the above example. Because flows A and B differ in size, D cannot guarantee correct distributed computation of the flow size.

For flow A, the received packet sequence contains gaps of at most $Z-1$ consecutive packets, i.e., at most $\gamma-1$ full groups, and there is no reordering. Hence, the $2^n + 2u + 2$ received packets form a $\mathrm{span}(\gamma, k)$ for flow A. For flow B, the first 2^n packets, together with packet $l - 1$ and first u packets j such that $j + 1$ is divisible by Z among the last l packets, arrive in order, leave gaps of at most $Z - 1$ packets and thus form a $\mathrm{span}(\gamma, k)$, and reordering is at most by $l - Z + 1 \leq 2^n - \gamma \cdot 2^k + 1$ packets. Our constructions are similar and reach the same conclusion when l **mod** Z is at most 1 packet. □

Algorithm 2 Computation of the flow size on two switches in the span-based model

```
1: procedure DESTINATION(j)
2:     diff ← Overshoot(h[j]);
3:     if 1 ≤ diff ≤ γ then c_2 ← c_2 + diff
```

In theorem 4, parameter γ controls a trade-off between span existence and reordering constraint. When group size k equals $n - t$ bits, and t is fixed, figure 9 illustrates how an increase in γ facilitates existence of a $\mathrm{span}(\gamma, k)$ and tightens the R constraint. Increasing t relaxes the feasibility limits in both γ and R dimensions. With $\gamma = 2^{t-1}$ groups, the reordering constraint in theorem 4 for the span-based model is $R \geq 2^{n-1} + 1$ packets, i.e., the same as in theorem 1 for the consecutive-loss model. As we will see later, the span-based model not merely generalizes the consecutive-loss model via parameter γ but characterizes maximum network noise more effectively.

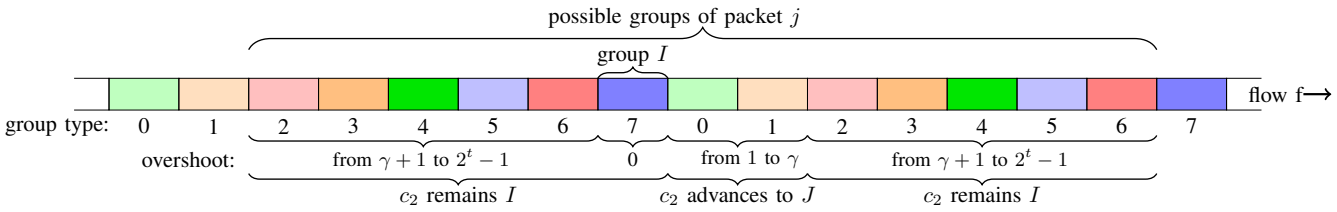


Fig. 10: Operation of algorithm 2 when $t = 3$ bits, $\gamma = 2$ groups, $c_2 = I$, and packet j arrives to D from group J.

We now present and analyze algorithm 2 which meets the feasibility limits established by theorem 4. Algorithm 2 uses $k = n - t$ and differs from algorithm 1 only in its updated DESTINATION procedure that advances c_2 if the overshoot is between 1 and γ groups.

Theorem 5. *When γ is at most $2^t - 1$ groups, algorithm 2 is assured to compute the flow size on two switches correctly if*

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

Proof. Similarly to the proof of theorem 3, we express $|f|$ as $K \cdot 2^{n-t} + m$ packets and prove two properties: (α) when D receives a packet from group J, c_2 advances from its current value I to J if and only if J is between $I+1$ and $I+\gamma$, and any packet received from a group numbered $I+1$ or larger before c_2 advanced to I does not belong to a $\mathrm{span}(\gamma, n-t)$, and ($\beta$) when the flow terminates, algorithm 1 updates c_2 to capture group count K correctly.

We start by proving property α and illustrate this proof for $t = 3$ bits and $\gamma = 2$ groups in figure 10. Suppose $c_2 = I$ where I is either initial value 0, or such that c_2 advanced to I from a value between $I-\gamma$ and $I-1$ upon receiving packet i from group I, and any packet received from a group numbered $I+1$ or larger before c_2 advanced to I does not belong to a $\mathrm{span}(\gamma, n-t)$. When D receives packet j from group J, the DESTINATION procedure calculates by how many groups the group type in $h[j]$ overshoots group type $I \bmod 2^t$ in c_2. If $J \leq I-1$, then packet j can precede group I at switch S by at most R packets, J is between $I - 2^t + \gamma + 1$ and $I-1$ due to the R constraint in conditions 2, the overshoot is between $\gamma+1$ and 2^t-1 groups, and switch D does not change c_2. If $J = I$, then the overshoot equals 0, and D does not change c_2 either.

If $J \geq I+1$, let Y refer to the packets of groups $I+1$ through $I+\gamma$. Because any packet received from a group numbered $I+1$ or larger before c_2 advanced to I does not belong to a $\mathrm{span}(\gamma, n-t)$, and since any $\mathrm{span}(\gamma, n-t)$ includes at least one of packets Y by the definition of a span, let y refer to the first such packet that arrives to D after c_2 advanced to I. Packet j is either y or another packet z received by D before y and belonging to a group numbered $I+\gamma+1$ or larger. If j is z, packet z can surpass y at S by at most R packets, J is between $I+\gamma+1$ and $I+2^t-1$ due to the R constraint in conditions 2, the overshoot is between $\gamma+1$ and 2^t-1 groups, and D does not change c_2. If j is y, i.e., J is between $I+1$ through $I+\gamma$, the overshoot is between 1 and γ groups, c_2 advances from I to J. Because all packets of a span by definition arrive to D in order, and since y is the first packet received from groups $I+1$ through J that can belong to a $\mathrm{span}(\gamma, n-t)$, any packet received from a group numbered $J+1$ or larger before c_2 advanced to J does not belong to a $\mathrm{span}(\gamma, n-t)$.

To prove property β, observe that property α in combination with the existence of a $\mathrm{span}(\gamma, n-t)$ implies that c_2 upon the flow completion is between $K-\gamma$ and K. The TERMINATION procedure computes the overshoot between 0 and γ groups and sets c_2 to K. □

In algorithm 2, as in algorithm 1 before, parameter t offers a trade-off between the communication overhead and assured resilience to network noise. The following theorem characterizes how the guaranteed resilience improves when the algorithm uses more sync bits. For t increasing from 1 to 2 and then 3 bits, figure 9 illustrates this expansion of the (γ, R) settings where algorithm 2 guarantees correct computation of the flow size.

Theorem 6. *Correct computation of the flow size on two switches by algorithm 2 remains assured if the values of t and γ change to $t+1$ and $2\gamma+1$ respectively.*

Proof. If the values of t and γ change to $t+1$ and $2\gamma+1$ respectively, the R constraint in conditions 2 remains the same because $2^n - (\gamma+1)\cdot 2^{n-t}$ equals $2^n - ((2\gamma+1)+1)\cdot 2^{n-(t+1)}$. Also, due to the reduction of k from $n-t$ to $n-(t+1)$, groups become twice shorter, γ full groups at the beginning of a flow become at most $2\gamma+1$ full reduced groups, and $\gamma-1$ full groups after any packet in the flow become at most $2\gamma = (2\gamma+1)-1$ full reduced groups. Hence, a $\mathrm{span}(\gamma, n-t)$ is also a $\mathrm{span}(2\gamma+1,\ n-(t+1))$. Conditions 2 remain valid and guarantee the correct computation by algorithm 2. □

Algorithm 2 generalizes algorithm 1 via parameter γ. When γ equals 2^{t-1} groups, algorithms 1 and 2 are the same. This property enables the following theorem to compare the span-based and consecutive-loss models in regard to the conditions that assure correct flow-size computation by the algorithm in these models.

Theorem 7. *For assuring the correct flow-size computation, the span-based model characterizes maximum network noise more effectively than the consecutive-loss model.*

Proof. With $\gamma = 2^{t-1}$ groups, algorithms 1 and 2 are identical, and conditions 1 and 2 impose the same R constraint. Consider the packet sequence received by D from a flow when conditions 1 hold. From this sequence, extract all packets such that algorithm 1 advances c_2 upon the arrival of the packet to D. By construction in theorem 3, these packets form a

span(γ, $n-t$) because they arrive to D in order, and at most $\gamma-1$ full groups are missing at the beginning of the flow and after any packet in this constructed subsequence. Hence, conditions 2 also hold.

While conditions 1 imply conditions 2, the reverse is not true. If D receives only the last packet from a flow that consists of $(\gamma+1)\cdot 2^{n-t}$ packets, then there is no reordering, the received packet forms a span(γ, $n-t$), and the $L+R$ constraint in conditions 1 does not hold. For $n=4$ bits, $t=3$ bits, and $\gamma=4$ groups, figures 6 and 9c illustrate this scenario: D receives only the last packet from a flow that consists of 10 packets, $L=9$ packets, $R=0$, and the correct computation of the flow size by the algorithm is assured by theorem 5 but not by theorem 3. Thus, for assuring the correct flow-size computation, the span-based model characterizes maximum network noise more effectively than the consecutive-loss model. □

Parameter γ affects loss tolerance of algorithm 2. When γ is only 1 group, the algorithm is guaranteed to be robust against loss of all packets in group 0 and all but one packet in each of the subsequent groups, the assured reordering resilience is 2^n-2^{n-t+1} packets, and D advances c_2 upon receiving a packet from the next group only. Figure 10 shows that as γ increases, D starts advancing c_2 upon receiving packets from groups farther away, the assured reordering resilience decreases, and the same amount of packet loss poses a smaller danger for existence of a span(γ, $n-t$). When γ becomes 2^t-1 groups, algorithm 2 increases its guaranteed loss resilience to $2^n-2^{n-t}-1$ consecutive packets without assuring any resilience to reordering.

If conditions 2 do not hold, algorithm 2 still might be able to compute the flow size correctly. This can happen because conditions 2 are unnecessarily tight. For example, without affecting the correctness guarantees, we can relax conditions 2 so that the definition of a span allows omitting at most 2^t-2, rather than $\gamma-1$, full groups after the last packet in the span.

When network noise violates conditions 2, and algorithm 2 computes the flow size incorrectly, the algorithm underestimates $|f|$. The following theorem establishes an upper bound on such underestimation.

Theorem 8. *If γ is at least 2^{t-1} groups, the network loses X packets of the flow, reordering R is at most $2^n-(\gamma+1)\cdot 2^{n-t}$ packets, and a* span(γ, $n-t$) *does not exist, then algorithm 2 underestimates the flow size by at most $\frac{2^t}{2\gamma+1-2^t}\cdot X$ packets.*

Proof. Consider packet i from group I and packet j from group J that consecutively advance c_2. The R constraint ensures $j>i$ and thus implies that algorithm 2 never overestimates $|f|$. Express J as $I+a\cdot 2^t+b$ groups where a is a nonnegative integer, and b is between 0 and 2^t-1. Upon receiving packet j, switch D advances c_2 by b groups and fails to account for $a\cdot 2^t$ groups. Since i and j advance c_2 consecutively, at least $a\cdot\gamma$ of these unaccounted $a\cdot 2^t$ groups are lost or arrive to D before packet i. At most $2^t-\gamma-1$ groups that succeed i at S can arrive to D before i. Thus, at least $a\cdot\gamma-(2^t-\gamma-1)$ of the unaccounted $a\cdot 2^t$ groups are lost. The ratio of the unaccounted packets to the lost packets is at most $\frac{a\cdot 2^t}{a\cdot\gamma+\gamma+1-2^t}$, which is bounded from above by $\frac{2^t}{2\gamma+1-2^t}$ when a equals 1. This upper bound holds for the entire flow, including before the first advancement, between two consecutive advancements, and after the last advancement of c_2. Hence, algorithm 2 underestimates the flow size by at most $\frac{2^t}{2\gamma+1-2^t}\cdot X$ packets. □

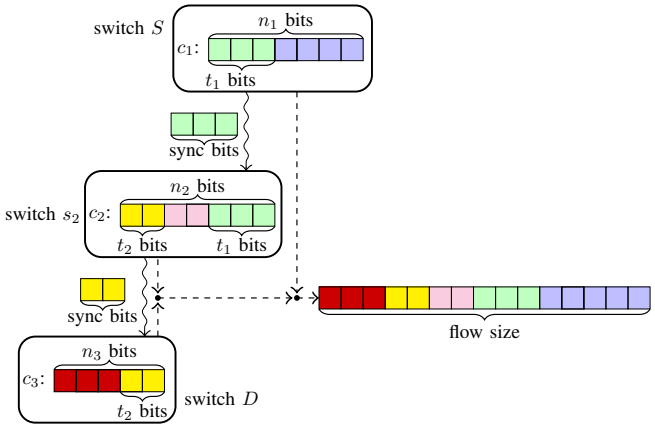


Fig. 11: Computation of the flow size on $p=3$ switches.

For $\gamma=2^t-1$ groups, the upper bound on the flow-size underestimation in theorem 8 is $\frac{1}{1-\frac{1}{2^t}}\cdot X$ packets.

We conclude the section by quantitatively illustrating the assured resilience of algorithm 2 to network noise. When $n=8$ bits, $t=2$ bits, and $\gamma=1$ group, algorithm 2 is guaranteed to compute the flow size correctly if a span(1, 6) exists with reordering by at most 128 packets. With t increased to 3 bits, the computation correctness is assured if a span(1, 5) exists, and reordering is at most by 192 packets. When n and t remain at 8 and 3 bits respectively, and γ increases to 6 groups, the assuredly correct computation comes with existence of a span(6, 5) and reordering by at most 32 packets. To illustrate theorem 8 when conditions 2 do not hold, the choice of $t=2$ bits and $\gamma=2$ groups guarantees that algorithm 2 underestimates the flow size by at most $4\cdot X$ packets. With $t=3$ bits and $\gamma=6$ groups, the flow-size underestimation is at most by $1.6\cdot X$ packets.

VI. Computation on three or more switches

This section generalizes our earlier results to support robust distributed computation of the flow size on more than two switches. With s_1 and s_p referring to switches S and D respectively, and m varying from 1 to p, the flow traverses a sequence of p switches s_m and potentially utilizes multiple paths between each pair of consecutive switches. For $p=3$ switches, figure 11 illustrates how the p switches maintain a distributed counter for the flow.

We employ the span-based model to represent maximum network noise for each of the $p-1$ stretches from S to s_m, where m varies from 2 to p. Instead of γ, we use γ_{m-1} for characterizing how network noise affects the flow at S to produce the packet sequence received by s_m. Whereas R_{m-1} refers to the reordering parameter for the stretch that

Algorithm 3 Computation of the flow size on three or more switches in the span-based model

```
1: procedure INITIALIAZTION
2:     for m = 1, ..., p do c_m ← 0
3: procedure INTERNAL(j, m)
4:     diff ← Overshoot(h[j], m);
5:     if 1 ≤ diff ≤ γ_{m−1} then c_m ← (c_m + diff) mod 2^{n_m};
6:     h[j] ← ⌊ c_m / 2^{n_m − t_m} ⌋
7: procedure DESTINATION(j)
8:     diff ← Overshoot(h[j], p);
9:     if 1 ≤ diff ≤ γ_{p−1} then c_p ← c_p + diff
10: procedure TERMINATION
11:    for m = 2, ..., p do
12:        c_m ← c_m + Overshoot(⌊ c_{m−1} / 2^{n_{m−1} − t_{m−1}} ⌋, m);
13:    |f| ← c_p;
14:    for m = p − 1, ..., 1 do
15:        |f| ← |f| · 2^{n_m − t_m} + c_m mod 2^{n_m − t_m}
16: function OVERSHOOT(g, m)
17:    return (g − c_m mod 2^{t_{m−1}} + 2^{t_{m−1}}) mod 2^{t_{m−1}}
```

reaches s_m, packet i may arrive to s_m before packet j only if $i \leq j + R_{m-1}$.

Algorithm 3 is our RoDiC solution for the generalized problem. The t_{m-1} MSBs of chunk c_{m-1} and t_{m-1} LSBs of chunk c_m form an overlap between the counting states at s_{m-1} and s_m. The t_{m-1} sync bits in each packet sent by s_{m-1} carry the group type of the packet on the hop to s_m. Algorithm 3 inherits the SOURCE procedure from algorithms 1 and 2. When packet j arrives to s_m where m is between 2 and $p-1$, the INTERNAL procedure computes the overshoot of the group type in $h[j]$. If the overshoot is between 1 and γ_{m-1} groups, s_m advances c_m modulo 2^{n_m} packets. After resetting the sync bits in packet j to the t_m MSBs of c_m, the INTERNAL procedure forwards the packet into the next hop. Thus, unlike in the SOURCE procedure, the INTERNAL procedure advances c_m before setting the sync bits in the sent packet to track the last packet that advanced c_m. The DESTINATIONUPDATE procedure remains essentially the same as in algorithm 2, with γ and c_2 being renamed to γ_{p-1} and c_p respectively. When the flow terminates, the TERMINATION procedure iteratively assembles all the c_m counters to compute the flow size.

In algorithm 3, each stretch of switches from S to s_m maintains a distributed subcounter that consists of $N_m = n_m + \sum_{k=1}^{m-1} (n_k - t_k)$ bits. Hence, the overall distributed counter can count flow sizes up to 2^{N_p} packets.

Theorem 9. *Algorithm 3 computes the flow size on p switches correctly if, for each m between 2 and p, there exists a* $\mathrm{span}(\gamma_{m-1}, N_{m-1} - t_{m-1})$ *at s_m where the span packets are the packets that advance c_{m-1}, and $R_{m-1} < 2^{N_{m-1}} - (\gamma_{m-1} + 1) \cdot 2^{N_{m-1} - t_{m-1}}$ packets.*

Proof. To prove that switch s_m advances its chunk c_m correctly, we interpret the stretch of switches from s_1 to s_m as a two-switch counter where s_m acts as the destination switch, the stretch of switches from s_1 to s_{m-1} represents the source switch, and γ_{m-1} groups, R_{m-1} packets, t_{m-1} sync bits determine the resilience settings. By combining chunks c_1 through c_{m-1}, the integrated source switch maintains a counter with N_{m-1} bits. We iteratively apply theorem 5 for m from 2 to p and establish the result for the spans where the packets are the packets that advance c_{m-1}. □

Flow-size computation on p switches involves a number of parameters. Each configuration of algorithm 3 needs to specify size n_m for chunks $c_1, \ldots, c_p$, numbers $t_1, \ldots, t_{p-1}$ of the sync bits, and values $\gamma_1, \ldots, \gamma_{p-1}$. In practice, the most important decision is to select the t_1 and γ_1 values because the most frequent state updates occur on the hop from S to s_2, with the update frequency declining exponentially on each subsequent hop. For example, consider computation on three switches where the first two maintain seven-bit counter chunks. Setting t_1 to 3 sync bits implies $R_1 + 16 \cdot \gamma_1 \leq 112$ packets. Increasing t_1 does not significantly improve the situation because theorem 4 rules out a deterministic RoDiC algorithm that guarantees correct computation for $R_1 + 2^{n_1 - t_1} \cdot \gamma_1 \geq 129$ packets. Setting γ_1 to 5 groups leads to $R_1 = 40$ packets and the constraint that network noise does not completely alter at least 1 out of 5 consecutive groups of 16 packets. This is reasonable in practice. For internal switch s_2, even the setting with $t_2 = 2$ sync bits and $\gamma_2 = 2$ groups produces $R_2 = 512$ packets and $2^{N_2 - t_2} \cdot \gamma_2 = 1024$ packets.

The above example unveils that communication quality on the first hop is the most important, whereas subsequent switches are amenable to more economical use of their resources because network noise creates weaker impacts on later hops. This property holds in general, implying that the source switch should get the largest counter chunk, and the chunks at later switches can be made smaller. For example, if we increase the chunk at S to 9 bits and decrease the chunk at s_2 to 5 bits, the network-noise constraints at s_3 remain the same but the network-noise constraints on the first hop are significantly and usefully relaxed to $R_1 + 64 \cdot \gamma_1 \leq 448$ packets.

VII. REAL-TIME TELEMETRY OF PACKET LOSS

While our paper mostly focuses on computation of the flow size, this section briefly discusses application of the RoDiC approach to real-time telemetry of packet loss, a different kind of monitoring tasks. Because computation of the packet loss over a path has to involve both ends of the path, packet-loss telemetry is an intrinsically distributed task. Specifically, we seek to measure the packet loss of flow f on its path from source S to destination D. Furthermore, telemetry in real time necessitates measuring the packet loss at a finer granularity than the entire flow duration [11]–[14].

In instantiating the RoDiC principles of packet grouping and state overlap for real-time packet-loss telemetry, we inherit and complement the elements of our flow-size computation solution. While we similarly partition f at S into groups of consecutive packets, our real-time telemetry design measures packet loss at the granularity of groups. The group size is fixed in advance to be the same for all groups. The group

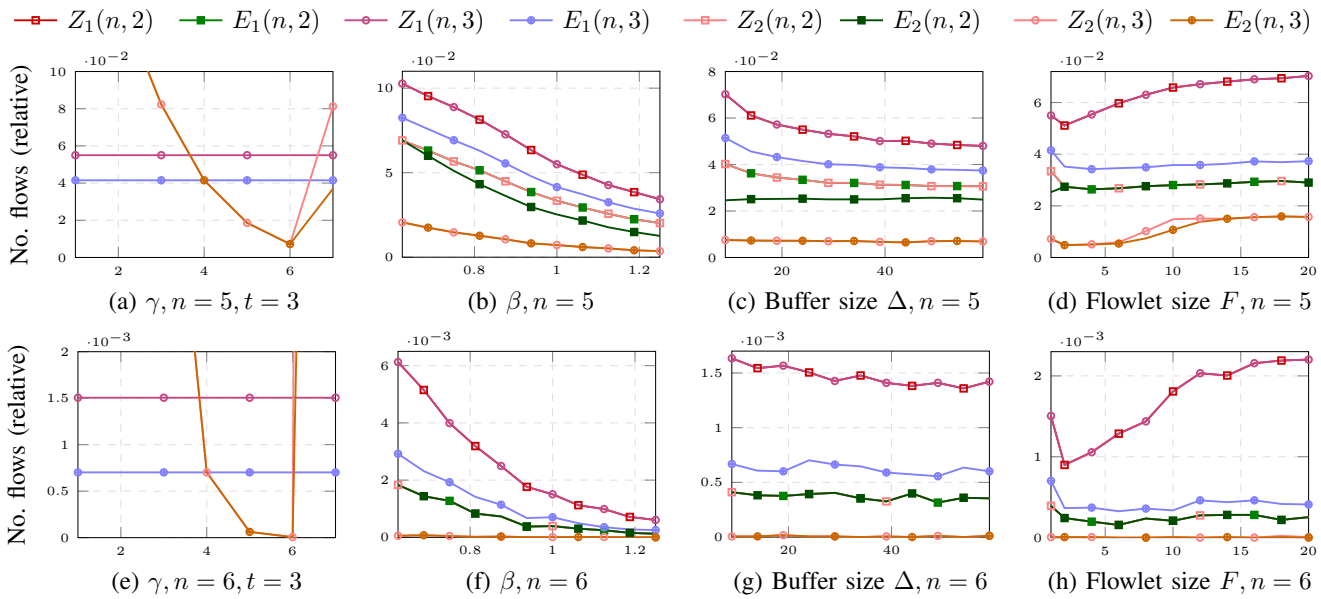


Fig. 12: Numbers Z_1 and Z_2 of flows violating conditions 1 and 2 respectively, and numbers E_1 and E_2 of flows with sizes calculated incorrectly by algorithms 1 and 2 respectively, as functions of γ, β, buffer size Δ, and flowlet size F; the source counter chunks are sized to: (a)-(d) $n = 5$ bits and (e)-(f) $n = 6$ bits.

type serves again as the overlap between the counting states at S and D. Each packet carries the group type in the sync bits to ensure that D correctly tracks G, the largest group number among the groups that have delivered a packet to D. Compared to the flow-size application, the packet-loss telemetry design maintains additional state at D for a certain number of groups. These groups include and immediately precede group G and are such that D does not yet assuredly know how many packets it ends up receiving from the group. For each of the earlier groups, our solution computes the packet loss for the group by subtracting from the group size the final number of packets received by D. We will elaborate and evaluate the above design for real-time packet-loss telemetry in our future work.

VIII. Experimental Evaluation

Methodology. Our evaluation follows the same approach as those for VL2 [18], pFabric [19], and pHost [20]. Specifically, we perform simulations driven by realistic traffic traces generated from the data-mining distribution of flow sizes [18], 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 [21]. 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. Our simulation code is publicly available [22].

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 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. 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] number $Z_1(n,t)$ of flows that violate conditions 1 with n-bit source counter chunks and t sync bits, [2] number $E_1(n,t)$ of flows for which algorithm 1 computes the flow size incorrectly, [3] number $Z_2(n,t)$ of flows that violate conditions 2, and [4] number $E_2(n,t)$ 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. t is equal to 2 or 3 sync bits. 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 12a and 12e 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 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 12a and 12e, 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 7 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 12b and 12f 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 $\mathrm{span}(\gamma, n-t)$ 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 optimal $\gamma = 2^{t-1}$.

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 6, all such flows satisfy conditions 2 for $t + 1$ sync bits when γ is $2^t + 1$.

Dependency on buffer size Δ. Figures 12c and 12g 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 12c are much smoother than in figure 12g 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 figures 12b and 12d vs. figures 12f and 12h.

Dependence on flowlet size F. Figures 12d and 12h 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 loss probability when packets of the same flow follow one path rather than many.

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 $\mathrm{span}(\gamma, n-t)$ does not exist, theorem 8 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 $\mathrm{span}(\gamma, n-t)$. 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 distributed execution of 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 model of network noise.

IX. 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, [23]–[25] 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 [4], DISCO [3], and CEDAR [5] 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. CEDAR constructs variables as an array that maps values of a counter variable to real counter values, with the increment probability being 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 [26] and CounterTree [27] 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) [6]. 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 [8] combines the ideas of the CounterTree and CM sketches, reducing the number of bits in each cell of the sketch table. UnivMon [9] exploits a sketch hierarchy for different measurement tasks, such as heavy-hitter detection or moment estimation. Elastic Sketch [10] 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. Geared toward minimizing communication complexity, [28], [29] detect network-wide heavy hitters in a model where switches report their local counters to a coordinator. Their works can be further improved by applying the q-MAX algorithm [30]. FlowRadar [31] 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.

Distributed flow state that changes routing. DIFANE [32] and vCRIB [33] 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 [15]–[17]. These schemes are static and hence immune to network noise.

Telemetry. Telemetry of packet loss is inherently distributed. While [11]–[14] measure packet loss per time interval, we pursue a different approach of measuring loss per packet group. The per-group granularity of our approach has salient advantages. For example, because the number of packets per group is fixed a priori, there is no need to communicate to the destination the number of packets that arrived to the source during a fixed time interval as in the time-based approach. Also, one can emulate the time-based approach by periodically collecting the group-based measurements of our algorithm at the destination.

Relation to CRDT. Conflict-free replicated data type (CRDT) supports replicas in multiple network nodes without coordination and concurrency on updates [34], 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 reordering of update operations. While the flow-size counter in switches S and D can be viewed as an CvRDT vector grow-only counter [34], our approach involves only partial counters, and – unlike with CRDT – feasibility analysis needs to incorporate network constraints.

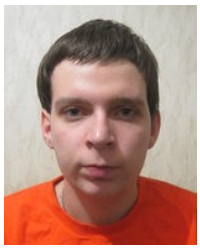
X. Conclusion

This paper proposed RoDiC, a new distributed approach for per-flow traffic monitoring under network noise. Instead of making the monitoring less accurate to deal with resource insufficiency at the designated network element, our approach computes flow metrics exactly by involving extra network elements that have spare resources. In providing robustness against network noise, RoDiC employs an open-loop paradigm that introduces no extra packets, communicates flow state in-band by piggybacking few control bits on packets of the monitored flows, and keeps latency low. The proposed technique is general and relies on two main principles of packet grouping and state overlap. First, we applied the RoDiC approach to problem of computing the sizes of all flows. Then, we briefly discussed how to instantiate RoDiC for real-time telemetry of packet loss. We analytically established conditions guaranteeing correct operation of the designed RoDiC algorithms and complemented the analysis with simulations driven by realistic traffic traces. Our evaluation substantiated the potential of RoDiC to effectively balance the monitoring load on the

network while keeping the computation and storage overhead low.

REFERENCES

[1] V. Demianiuk, S. Gorinsky, S. I. Nikolenko, and K. Kogan, "Robust Distributed Monitoring of Traffic Flows," in *ICNP*, 2019, pp. 1–11.
[2] A. Cvetkovski, "An Algorithm for Approximate Counting Using Limited Memory Resources," in *SIGMETRICS*, 2007, pp. 181–190.
[3] 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.
[4] R. Stanojevic, "Small Active Counters," in *INFOCOM*, 2007, pp. 2153–2161.
[5] E. Tsidon, I. Hanniel, and I. Keslassy, "Estimators Also Need Shared Values to Grow Together." in *INFOCOM*, 2012, pp. 1889–1897.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] 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.
[13] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. G. Chen, and G. Mirsky, "AM-PM: Efficient Network Telemetry Using Alternate Marking," *IEEE Network*, vol. 33, no. 4, pp. 155–161, 2019.
[14] A. Riesenberg, Y. Kirzon, M. Bunin, E. Galili, G. Navon, and T. Mizrahi, "Time-Multiplexed Parsing in Marking-Based Network Telemetry," in *ACM SYSTOR*, 2019, pp. 80–85.
[15] 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.
[16] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing Tables in Software-Defined Networks," in *INFOCOM*, 2013, pp. 545–549.
[17] P. Chuprikov, K. Kogan, and S. Nikolenko, "How to Implement Complex Policies on Existing Network Infrastructure," in *SOSR*, 2018, pp. 9:1–9:7.
[18] 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.
[19] 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.
[20] 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.
[21] "YAPS: Yet Another Packet Simulator," http://wiki.github.com/NetSys/simulator/.
[22] "Robust Distributed Monitoring of Traffic Flows." https://github.com/chavit/RobustMonitoring.
[23] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a Statistics Counter Architecture," in *HOTI*, 2001, pp. 107–111.
[24] ——, "Maintaining Statistics Counters in Router Line Cards," *IEEE Micro*, vol. 22, no. 1, pp. 76–81, 2002.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] R. B. Basat, G. Einziger, S. L. Feibish, J. Moraney, and D. Raz, "Network-Wide Routing-Oblivious Heavy Hitters," in *ANCS*, 2018, pp. 66–73.
[30] R. B. Basat, G. Einziger, J. Gong, J. Moraney, and D. Raz, "q-MAX: A Unified Scheme for Improving Network Measurement Throughput," in *IMC*, 2019, pp. 322–336.
[31] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers," in *NSDI*, 2016, pp. 311–324.
[32] M. Yu, J. Rexford, M. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," in *SIGCOMM*, 2010, pp. 351–362.
[33] M. Moshref, M. Yu, A. B. Sharma, and R. Govindan, "vCRIB: Virtualized Rule Management in the Cloud," in *HotCloud*, 2012.
[34] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A Comprehensive Study of Convergent and Commutative Replicated Data Types," INRIA, Research Report RR-7506, 2011.



Vitalii Demianiuk is a postdoctoral fellow at Ariel University (Israel). While pursuing his Ph.D. studies at the Steklov Institute of Mathematics at St. Petersburg (Russia), he worked as a Research Assistant in IMDEA Networks Institute (Spain) in 2017-2019. He obtained his M.Sc. degree from ITMO University in St. Petersburg (Russia) in 2016. His research interests include packet classification, software defined networks, network function virtualization, and combinatorial optimization.

Sergey Gorinsky (Member, IEEE/ACM) is a tenured Research Associate Professor at IMDEA Networks Institute (Spain), where he leads the NetEcon research group. Dr. Gorinsky received his Ph.D. and M.S. degrees from the University of Texas at Austin (USA) in 2003 and 1999 respectively and Engineer degree from Moscow Institute of Electronic Technology (Russia) in 1994. From 2003 to 2009, he served on the tenure-track faculty at Washington University in St. Louis (USA). The areas of his primary research interests are computer networking, distributed systems, and network economics. He served as a TPC chair of ICNP 2017 and other conferences, as well as a TPC member for a much broader conference population including SIGCOMM, CoNEXT, and INFOCOM. Sergey Gorinsky contributed to conference organization in many roles, such as a general chair of SIGCOMM 2018.

Sergey Nikolenko is with the Steklov Institute of Mathematics at St. Petersburg and National Research University Higher School of Economics (St. Petersburg, Russia). He performs research in machine learning (deep learning, Bayesian methods, natural language processing, etc.), algorithm analysis (networking algorithms, competitive analysis, theoretical computer science), and mathematics. He obtained his Ph.D. degree from the Steklov Institute of Mathematics at St. Petersburg (Russia) in 2009 and his M.Sc. degree from St. Petersburg State University (Russia) in 2005.

Kirill Kogan is a Senior Lecturer at Ariel University (Israel). He received his Ph.D. degree from Ben-Gurion University (Israel) and worked as a Technical Leader at Cisco Systems (Israel) during 2000-2012. His current research interests are in design, analysis, and implementation of networked systems broadly defined and, in particular, network processors, switching fabrics, packet classification, network management, service architecture, and cloud computing.