IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 28, 2021, accepted June 20, 2021, date of publication June 24, 2021, date of current version July 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3092281

An Elephant in the Room: Using Sampling for Detecting Heavy-Hitters in Programmable Switches

PEDRO RODRIGUES TORRES, JR. [1], **ALBERTO GARCÍA-MARTÍNEZ** [2], **MARCELO BAGNULO** [2], **AND EDUARDO PARENTE RIBEIRO** [3]

[1]Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, Curitiba 81520-260, Brazil
[2]Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Madrid 28903, Spain
[3]Departamento de Engenharia Elétrica, Universidade Federal do Paraná, Curitiba 83255-976, Brazil

Corresponding author: Pedro Rodrigues Torres, Jr. (pedro.torres@ufpr.br)

The work of Alberto García-Martínez and Marcelo Bagnulo was supported by the TRUE5G Project ('Evolución hacia redes y servicios auto-gestionados para el 5G del futuro') by the Spanish National Research Agency under Grant PID2019-108713RB-C52/AEI/10.13039/501100011033.

ABSTRACT The ability to detect elephant flows in the forwarding device itself, i.e., a switch, facilitates the deployment of new advanced applications such as load-balancing, per-flow QoS management, etc. Sketches and Space Saving summarization techniques are used for elephant flow detection. However, their memory and computing requirements force the cooperation of an external controller device, due to the scarce resources of current programmable switches. To overcome this limitation, we adapt Sketch and Space Saving elephant flow detection techniques to operate with instant notification and sampled traffic. We evaluate the performance of the resulting techniques with three real traffic traces. The use of sampling allows the identification of a large share of the total traffic corresponding to the elephant flows with a low memory footprint and a reduction of the computing requirements in two orders of magnitude compared to unsampled versions. In turn, we observe a slight increase in the number of false positives and the number of flow notifications.

INDEX TERMS Sampling, elephant flows, sketches, space saving.

I. INTRODUCTION

Many recent network mechanisms leverage on the detection of large and long-lasting flows. Network performance can be improved by applying to these flows specific routing policies, queue management disciplines or by reshaping them. For example, a load-balancing application can be implemented as follows: The traffic traversing a switch is inspected in order to identify large flows; once a flow is detected, the rest of the packets belonging to this flow are forwarded to an alternative path already configured in the switch. This implies that the switch may automatically route packets belonging to these flows through a different path, even without any intervention of an external controller device. Such a mechanism requires both tracking per-flow information at wire speed and identifying the large flows that deserve the differentiated treatment. Programmable switches are a natural fit for this operation, as they enable flexible packet-to-flow processing without sacrificing performance [1], [2].

Elephant flows can be defined based on size and duration: an elephant flow is a flow with a minimum frequency or size that lasts for at least D seconds [3]. Elephant flow detection techniques adopted by the networking community were initially developed for *item summarising* in the context of *stream data processing*. They are mainly based on *sketches* or on *counters*. However, these techniques may not be suitable for both autonomous operation in the switch and the constrained memory and processing capacities available at these devices, that must usually be shared with other functions also key for network operation. In fact, most elephant-flow detection solutions require the cooperation of an external device, which receives information from the switch, and processes it to identify large and long-lasting flows. In this paper we propose and evaluate the use of sampling for implementing

The associate editor coordinating the review of this manuscript and approving it for publication was Hosam El-Ocla.

sketch and counter-based elephant flow detection exclusively in programmable switches, without the need of any external processing device.

The main families of programmable hardware switches are Network Processing Units (NPU) and pipeline architectures (Reconfigurable Match-Action Table, RMT). NPU switches provide a set of low-level instructions designed for packet processing. Alternatively, RMT switches implement a feature-rich pipeline with function blocks implemented in silicon in which different paths are activated according to matches with packet attributes. RMT switches provide higher performance than NPU ones, but are less flexible, as the operations to perform depend on the particular stages implemented in the hardware. Note that both NPU and RMT are restricted in the amount of operations that can be performed at wire speed, and by the small amount of fast memory they include. We can think of the computing and memory resources as a limited budget that has to be distributed among the different switch functions, so the lower the hardware resources assigned for flow identification, the better.

In this paper we adapt the sketch and counter-based techniques to work with traffic samples in order to reduce the hardware requirements of elephant flow detection performed (exclusively) in the data plane. We consider as a reference application the re-routing of elephant flows through an alternative path installed on the switches of an ISP [4], [5]. The requirement to operate as a fully data plane implementation, i.e., an implementation which does not need to send data to an external device such as a Software-Defined Networking (SDN) controller, results in a different trade-off with most current flow characterization proposals. In order to operate efficiently, we adapt the algorithms discussed to sampling.

To evaluate the performance of our solution, we use three sets of real traces captured at ISP egress links as inputs to emulate sketch and counter-based heavy-hitter detection. In this way, we are able to assess the application of these techniques to a system operating with Internet traffic, such as proposed in [4] and [5]. Although the use of sampling decreases the accuracy of the data stream frequency estimation, we are able to find a significant set of flows that contribute to most of the traffic.

The paper is structured as follows: We next review the state of the art related to elephant flow detection in programmable switches. In Section III we describe the modifications of the summarizing techniques required to operate with sampled traffic. Then we present the experiments performed for the different elephant flow identification techniques with real traces, varying memory and sampling rates, and discuss the results. We end with the conclusions and future work.

II. RELATED WORK: ELEPHANT FLOW DETECTION IN PROGRAMMABLE SWITCHES

Many elephant flow detection techniques derive from *item summarising* in the context of *stream data processing*. We next present the two main families of techniques, namely *sketch-based* and *counter-based*.

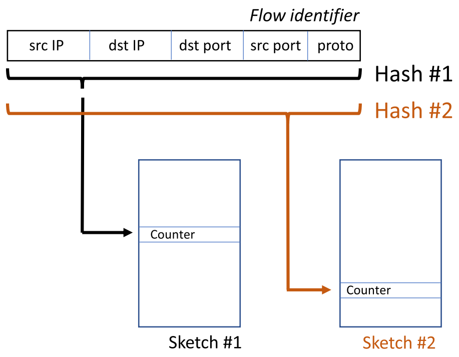

FIGURE 1. Example of a 5-tuple flow identifier mapping to two table sketches.

Sketch-based techniques aim to keep track of the traffic transmitted by flows in a low memory footprint. Sketches track the packet frequency (or accumulated size) of a flow without storing its 5-tuple flow identifier, corresponding to the protocol and source/destination addresses and ports. An array $[(1, 1), \ldots (d, w)]$ of depth d (i.e., d tables or *sketches*) and width w is allocated to store the frequency count. Similar to *bloom-filters*, sketches use d different hash functions computed over the flow identifier tuple to point to the different memory locations in which flow frequency (size) is updated, as depicted in Figure 1. This occurs every time a new item arrives. For Count Min Sketch (CMS [6]), the final frequency value is the minimum of the values of the matching entries over the different positions associated to the same flow. Note that this minimum value is not affected even if the result of one of the d hashes is the same for two different flows. To allow concurrent access to memory, it is typical that each hash address points to an entry in a separate memory bank. Thus, the time required to update the flow counters is small and bounded. While the update process is quite efficient for sketch structures, the reverse mapping from entries to flow identifiers, required to get the flows complying with an elephant flow definition, is challenging and requires additional information. An example for this complexity is illustrated by FlowRadar [7], that includes an additional column in each sketch with a XOR of all the flow identifiers mapped in that entry over a period of time. The operation to get the flows and the frequency count corresponding to each flow is performed out of the switch, in the controller.

The interest in the use of sketches to perform flow summarization (including elephant flow detection) in programmable switches has resulted in a sheer number of recent proposals [2], [7]–[14]. The cited papers share the following architecture: the programmable switch is engineered to annotate in the sketches per-flow statistics for each received packet, at wire speed; periodically, the tables are conveyed to an external device, a controller, that extracts the relevant information, starting with the reverse mapping from entry to flow identifier. Any processing related to flows, such as detect attacks, compute traffic matrices, etc., is performed in the controller. If the switch has to be configured accordingly to the output of the sketch processing, e.g., to re-route or discard the packets of a flow, the controller is responsible to perform this configuration.

There are a number of differences between these proposals and ours. First, our focus is on elephant flow detection, formulated as a binary decision – a flow is or is not an elephant flow. The aforementioned sketch proposals are more flexible in the results they can provide. Notably, many are able to provide a precise characterization of the frequency of the flows involved, for which they can prove tight bounds. Finally, we highlight that our solution can operate in an standalone fashion, as the switch can detect and manage elephant flows autonomously, without the need to interact with any external device. Thus, it reduces the requirements for the capacity of managing devices, and the solution is not affected by high loads in the network through which control data is conveyed [10]. In turn, the requirement of standalone operation challenges the orchestration of the scarce resources at the switches, computing capacity and memory, promoting the use of aggressive approaches such as sampling.

IDEAFIX [15] is a sketch-based mechanism to detect elephant flows in the switch's data plane that is aligned to our problem formulation. To detect elephant flows, IDEAFIX proposes an *instant notification* strategy: every time a packet is processed, it checks if the corresponding (current) flow exceeds a minimum threshold on both the size and the duration, and in that case it generates a notification. IDEAFIX can be implemented in both NPU and RMT architectures, and it is a full data plane implementation of an elephant flow tracking and notification solution. In our proposal we adopt the instant notification approach, we adapt it to Space Saving elephant flow detection, and we combine it with sampling.

ConQuest [16] is a sketch-based architecture that performs wire-speed traffic summarization fully at the data plane. ConQuest aims to detect the flows contributing most to queue occupation in the switches. As a full data-plane implementation, the switch can autonomously take actions based on the queue measurement, for example, to set the ECN bit for specific flows. They operate at shorter timescales and track a much lower number of flows than the rest of the architectures described in the paragraphs above. Although they note that the approach could be used for heavy hitter detection, it is neither described how, nor its performance reported.

Counter-based techniques follow a different approach than sketches to track flow frequency. In this case, the 5-tuple flow identifier is stored and associated to its corresponding packet (or byte) counter. Then, the emphasis is set on the way the limited number of entries are managed. Among many realizations of the main idea, we focus on Space Saving [17] (SS) to illustrate the principles of counter-based techniques. SS allocates a table of k entries. When a packet arrives and its flow identifier was already present in the table, the corresponding counter is updated. If there is no match, the flow identifier is inserted in the place of the entry with the lowest counter value. However, the counter for the new flow is not zeroed, but it starts with the value of the counter of the previous flow. Then new flows can remain in the table if its instantaneous rate is high enough, and old elephant flows that stagnate – so that their instantaneous rate is low – are eventually evicted. One concern of SS performance is related to the process of performing the flow lookup in the table. To ensure bounded-time operation, CAM memory should be available. The other critical operation for performance is the selection of the entry with minimum value, in case the packet flow identifier was not in the table. Note that this last operation should be performed at the speed at which packets belonging to mice flows arrive, i.e., close to wire speed. While a linked list could be used for maintaining the order of the elements, SSH [18] is a Space Saving implementation in which the flows with minimum count (of packets or bytes) are tracked by means of a *heap structure*. The heap structure is a binary tree in which each child node has a higher (count) value than its parent. Tracking the minimum value entry in this data arrangement is more efficient than doing so in a linked list. We later build our Space Saving implementation using this heap-based data structure.

Regarding to the type of switches in which SS can be deployed, the main difficulty is to efficiently select the entry with minimum value. This operation is better suited for NPU switches than for RMT pipelined architectures, as in this latter case it may require more cycles than allowed per stage.

The first effort to accommodate SS into a RMT architecture is HashPipe [1]. In this proposal, tracked flow counters are stored into one of d different tables. Packets matching an entry in any of the tables are used to update the counters appropriately, as for regular SS. However, when a packet does not match an existing entry, the entry to be evicted is selected as follows: A candidate entry is randomly selected from the first table and the new flow is inserted there. For the flow previously present in the first table (lets call it F_1), a random entry in the second table is selected (namely F_2). The flow (either F_1 or F_2) with higher counter is assigned to the entry at the second table; the other is checked against the value of a random entry at table 3, repeating the process until a flow (with a counter value lower than the rest of the entries randomly selected) is evicted. The resulting process is bounded in time, but the accuracy in the identification of large flows, that depends on the proper selection of the flow with minimum value, is reduced.

Precision [19] is a variant of HashPipe that improves the accuracy in keeping track of large flows by recirculating packets to previous stages without incurring in memory access violations. Neither HashPipe or Precision consider the use of sampling to reduce the cost of managing the data structure used to track flows.

III. SAMPLING-ENABLED ELEPHANT FLOW DETECTION

As described in Section II, previous proposals to detect elephant flows that operate fully in the data plane, either based on sketches [15] or on counters [1], [19], do not use

```
Algorithm 1
    Parameter Description:
    $T_b$ = Tables $0..B-1$, each with positions $0..W-1$
    $T_b.Counter[i]$: Packet counter at table $b$ and position $i$
    $T_b.FirstTS[i]$: Timestamp for first update at table $b$ and position $i$
    $T_b.LastTS[i]$: Timestamp for last update at table $b$ and position $i$
    $T_b.Notified[i]$: Indicating if a flow at table $b$ and position $i$ has already being notified
    $s$: Minimum number of samples for elephant flow detection
    $D$: Minimum duration for elephant flow detection
    $r$: Elephant flow reset time

procedure INIT($T$)
    for each $b \in 0..B-1$ do
        for each $i \in 0..W-1$ do
            $T_b.FirstTS[i] \leftarrow T_b.LastTS[i] \leftarrow T_b.Counter[i] \leftarrow 0$

procedure ANNOTATEPACKET($T, FlowId, Now$)
    $minCounter \leftarrow \infty$
    $newestLastTS \leftarrow 0$
    $minNotified \leftarrow 1$
    for each $b \in 0..B-1$ do
        $i \leftarrow$ HASH($b, flowId$)
        if $(Now - T_b.LastTS[i] > r)$ or $(T_b.FirstTS[i] = 0)$ then          ▷ Start new flow
            $T_b.Counter[i] \leftarrow 1$
            $T_b.Notified[i] \leftarrow 0$
            $T_b.FirstTS[i] \leftarrow T_b.LastTS[i] \leftarrow Now$
        else                                                                    ▷ Update flow values
            $T_b.Counter[i] \leftarrow T_b.Counter[i] + 1$
            $T_b.LastTS[i] \leftarrow Now$
            $minCounter \leftarrow$ MIN($minCounter, T_b.Counter[i]$)
            $newestFirstTS \leftarrow$ MAX($newestFirstTS, T_b.FirstTS[i]$)
            $minNotified \leftarrow$ MIN($minNotified, T_b.Notified[i]$)
    if $(minNotified = 0)$ and $(minCounter \geq s)$ and $(Now - newestFirstTS) \geq D$ then    ▷ Decide to notify
        NOTIFYELEPHANTTOAPPLICATION($flowId$)
        for each $b \in 0..B-1$ do
            $i \leftarrow$ HASH($b, flow$)
            $T_b.Notified[i] \leftarrow 1$
```

FIGURE 2. Algorithm for CMSS update.

sampling.[1] By using sampling, we aim to reduce the amount of resources required to detect elephant flows. We adopt an elephant flow definition based on size and duration similar to Mori *et al.* [3], that also considers sampling: an elephant flow is a flow that is sampled at least s times using sampling rate S, and lasts for at least D seconds. This definition is now used to produce new sketch-based and counter-based mechanisms. We note that our objective is to find a significant set of long-lasting flows accounting for the largest possible share of traffic, and we are not interested in determining the exact size or duration for each of these flows (for this discussion, we refer to Mori *et al.* [3]).

In order to circumvent the need for off-line processing to identify elephant flows, the mechanisms we propose report these flows at the very moment the sampled packet both exceeds a threshold count and lasted for a minimum duration. Thus, they perform *instant notification* as IDEAFIX does.

[1]Proposals considering sampling [9], [13] require off-line processing to generate the resulting list of flows, so they are not suitable for full data plane operation.

A. CMS SAMPLED

We next describe in detail our sketch-based mechanism: When a packet is sampled, several hash functions are applied to the flow identifier to obtain the entry to update at each of the d sketches. Each entry stores a packet (or byte) `Counter`, two timestamps, one with the time of the first (`FirstTS`) and last (`LastTS`) updates for the entry, and a `Notified` bit. A notification occurs when the minimum of both the count of sampled packets and duration across all tables exceed its corresponding threshold and the notified bit is unset; then, the notified bit is set to prevent further notifications for the same flow. `LastTS` is used to clear the state for the entries of a flow when the time since the last received packet exceeds the flow reset time, to allow table entry reuse. We call this strategy *CMSS*, for *CMS-Sampled*. The algorithm is shown in Figure 2.

CMSS is able to generate a notification once the thresholds in duration and size are exceeded. Note that the same flow may be notified several times only if there are periods of inactivity in which the flow was evicted from the data structure and included back again.

Algorithm 2

```
Parameter Description:
$T$: Table with positions from $0..W-1$
$flowsInTable$: Number of flows in table
$FlowId[i]$: Flow id at position $i$
$Counter[i]$: Packet (byte) counter at position $i$
$FirstTS[i]$: Timestamp for first update at position $i$
$Notified[i]$: Indicating if a flow at position $i$ has already being notified to an application
$s$: Minimum number of samples for elephant flow detection
$D$: Minimum duration for elephant flow detection
$r$: Elephant flow reset time

procedure INIT($T$)
    $flowsInTable \leftarrow 0$
    for each $i \in 0..W-1$ do
        $T.FlowId[i] \leftarrow T.FirstTS[i] \leftarrow T.Counter[i] \leftarrow 0$

procedure FLOWIDMATCH($T, FlowId$)
    if $FlowId \in T$ then
        return $T.position$
    else
        return -1

procedure ANNOTATEPACKET($T, FlowId, Now$)
    $i \leftarrow$ FLOWIDMATCH($T, FlowId$)
    if $(i = -1)$ and $(flowsInTable = W)$ then        ▷ Evict element at position 0 (root of the structure)
        BALANCETREE($T$)                                ▷ Heap balance only needed to select a victim
        $T.FlowId[0] \leftarrow FlowId$
        $T.FirstTS[0] \leftarrow Now$
        $T.Notified[0] \leftarrow 0$
    else if $(i = -1)$ then                              ▷ Start new flow
        $flowsInTable \leftarrow flowsInTable + 1$
        $T.FlowId[flowsInTable] \leftarrow FlowId$
        $T.FirstTS[flowsInTable] \leftarrow Now$
        $T.Notified[flowsInTable] \leftarrow 0$
    else                                                 ▷ Update flow counter
        if $(Now - T.LastTS[i] > r)$ then                ▷ Reset expired flow
            $T.FirstTS[i] \leftarrow Now$
            $T.Notified[i] \leftarrow 0$
        else
            $T.Counter[i] \leftarrow T.Counter[i] + 1$
    if $(T.Notified[i] = 0)$ and $(T.Counter[i] \geq s)$ and $(Now - T.FirstTS[i]) \geq D$ then   ▷ Decide to notify
        NOTIFYELEPHANTTOAPPLICATION($flowId$)
        $T.Notified[i] \leftarrow 1$
```

FIGURE 3. Algorithm for SSHS update.

B. SSH SAMPLED

Our counter-based mechanism is a variation of SSH [18], a Space Saving implementation in which the flows with minimum count (of packets or bytes) are tracked by means of a *heap structure*. We refer to it as *SSHS*, for *SSH-Sampled*. The heap structure is a binary tree in which each child node has a higher value than its parent. The implementation maps this structure in an array, with a counter value at HEAP[k] (parent) lower than the value at HEAP[$2 \times k + 1$] and HEAP[$2 \times k + 2$] (children) for all k, $k \geq 0$. For the sake of comparison, non-existing elements are considered to be infinite. The most interesting property of a heap is that its smallest element is always at the root, HEAP[0]. As for CMSS, SSHS reports elephant flows instantaneously, once a sampled packet exceeds both a packet count and duration thresholds. Each entry also contains a notification bit and two timestamps for tracking flow duration and recent activity. The SSHS algorithm is shown in Figure 3.

The maximum packet's processing time in SSHS largely depends on the BALANCETREE(T) function. The maximum number of operations that can be required to rebalance the structure when a new element is inserted (i.e., a sampled packet for a new flow arrives) grows with $O(Wlog(W))$, with W the number of entries in the managed table. As the table size W grows, it is more unlikely that an RMT switch could accommodate this operation in its tight processing schedule. Thus, when large tables are used, the SSHS strategy could only be implemented in NPU switches.

As for CMSS, SSHS is also able to generate notifications when both the thresholds in duration and size are exceeded. However, for SSHS, it is also possible to generate a notification to indicate that a flow is no longer being tracked. Thus, SSHS allows bounding the maximum number of flows an application manages to the size of the table. Note that CMSS is not designed to provide this feature, as it would involve a table-to-flow mapping that is costly to implement in the data plane.

C. IMPLEMENTATION CONSIDERATIONS

The natural path for implementing SSHS and CMSS is to code them in P4 [20]. P4 is the most popular abstraction for programmable switches, defined by a target-independent language. It is suited for a wide range of hardware device types, allowing for example to program RMT devices by specifying match-action primitives in different pipeline stages. The primitives are implemented according to the hardware capabilities of the target architecture, by the mediation of a hardware-specific compiler. Sketch and variations of space saving have been implemented in P4, and tested in different real hardware devices [1], [15], [19]. We have coded an early prototype of CMSS, that is available at [21].

Another implementation consideration is the cost of the sampling operation for programmable switches, that should be low. In many cases, the hardware architectures can efficiently implement the random selection of a packet according to a given sampling rate, as reported for `sFlow` [22] devices. If this is not the case, the cost of taking a decision for every incoming packet can be replaced by deciding the number of packets to skip until the next sample, as described for Nitrosketch [13].

IV. PERFORMANCE EVALUATION

A. EXPERIMENT SETUP

We have developed a Python simulator implementing CMSS and SSHS, along with CMS and SSH.[2] For the strategies using sketches, we consider two variants with different number of tables: depth 2 (CMS-2 and CMSS-2), the minimum for sketch operation, and 5 (CMS-5 and CMSS-5), as selected by Nitrosketch [13]. The hash functions used, one for each depth level, must be mutually independent hash functions. Another requirement for the hash functions is that they must have a larger result domain than the number of table entries. Both conditions are fulfilled by the use of a *crc-32* hash function with different initial parameters.

All implementations, included unsampled ones, are engineered to operate in instantaneous notification mode.

The parameters that can be configured for each strategy (CMS, SSH, CMSS, SSHS) are:

- Elephant flow's minimum duration D.
- Elephant flow's minimum size measured in bytes, B, for unsampled strategies (CMS and SSH), and minimum number of packets sampled s and sampling rate S for sampled ones (CMSS and SSHS).
- Amount of memory allocated for the data structures.
- Elephant flow's reset time r. This parameter is used to clear flows as described in Section III.

[2]https://bitbucket.org/torresweb/suleiman/

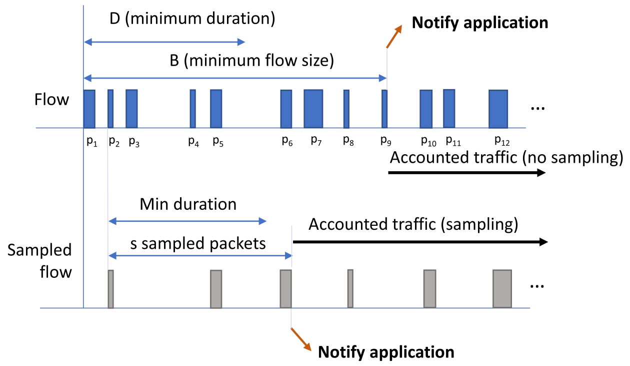

FIGURE 4. Traffic accounted for an elephant flow detection with and without sampling. For experiments involving sampling, we account the traffic after the flow is identified as an elephant, i.e., $p_7, p_8, \ldots p_{12}$, as these packets would be affected by the notified application (even though some of them are not sampled). For non-sampled elephant detection, traffic accounting starts after p_9.

To compare the strategies, we focus on the following results:

- Amount of traffic accounted after the flows are categorized as elephants. Figure 4 depicts an example of elephant flow detection in both unsampled and sampled strategies. The traffic that corresponds to the same elephant flows defined according to the unsampled definition of elephant flows is the *true positive* traffic. We can also extend these reasoning to flows, to identify *true positive* flows. Besides, the traffic spuriously identified as elephant, which does not comply with the unsampled definition, is the *false positive* traffic. We assume that the utility for the application (e.g., traffic engineering) is maximum when the identified traffic adheres to the size-duration elephant flow specification, i.e., the unsampled definition. Thus, we aim to capture as many *true positive* flows as possible. The number of *false positive* flows should be kept as low as possible, as its notification triggers application activity that is not justified by the size of the flow.
- Number of memory accesses. We count the number of memory accesses to flow entries, as the time to perform this operation dominates the whole algorithm execution. This number is used to estimate the computing requirements at the switch for the detection of elephant flows. The bigger the number, the lower the computing resources available for other tasks.
- Number of flow notifications. Each notification triggers an action in the application – either internal or external to the switch. Flow notifications are only useful when they correspond to elephant flows according to the size-duration specification, and should be avoided for other flows. Additionally, the same flow may be notified many times, if the flow is evicted from the flow

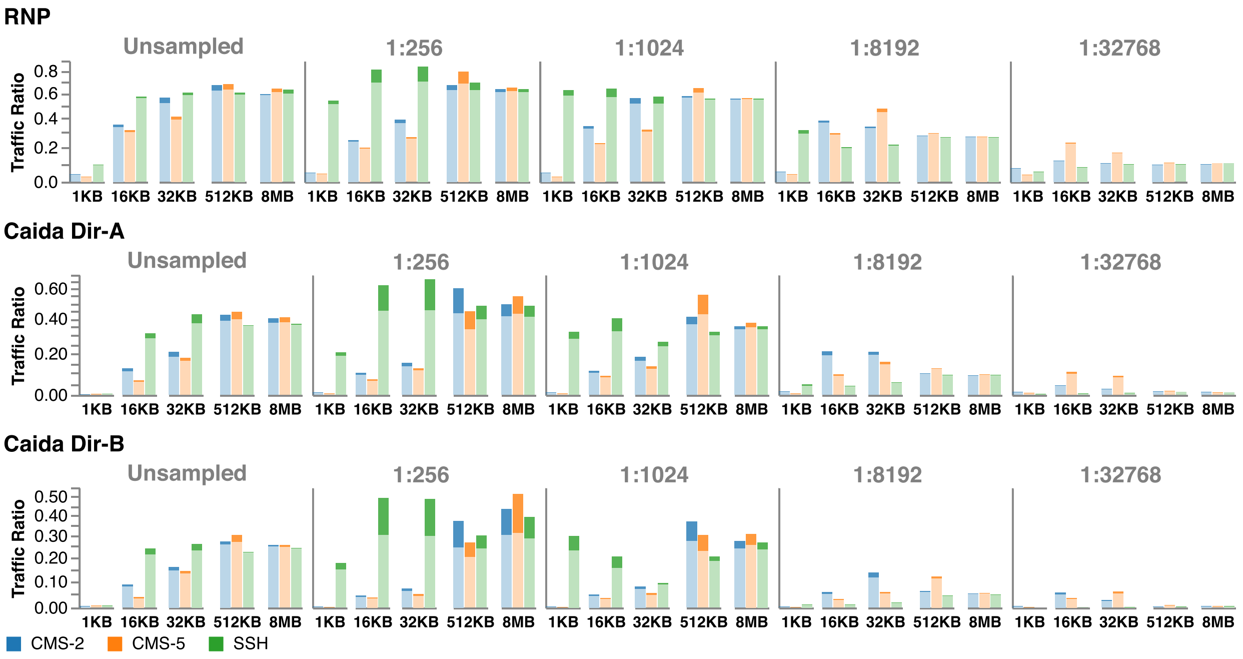

FIGURE 5. Ratio of managed traffic for RNP, CAIDA Dir-A and CAIDA Dir-B datasets. 1KB, 16KB, 32KB, 512KB and 8MB are the memory size for each configuration. Dim colors represent *true positives* and full colors *false positives*.

tracking table, and then returned. The number of flow notifications should be kept as low as possible, as they trigger additional computations.

We now describe the experiments. We first define a reference elephant flow pattern, that results from the processing of the input traces without sampling and with unlimited memory. For this case, we set D and B to the values indicated in IDEAFIX, i.e., $D = 10$s and $B = 10$MB. This reference pattern is used to identify *true positive* and *false positive* traffic and flow notifications. Flow reset time is $r = D$. The values of D and r are the same for CMS, SSH, CMSS and SSHS. We vary the memory available to represent different types of switches/configurations.

In our implementation, the memory allocated to represent the information corresponding to a flow in SSH (and SSHS) is the IP addresses and ports of the flows (12 bytes for IPv4 addresses), a 4-byte counter (except for one bit used to track notifications), and two 4-byte timestamps, indicating the arrival time of the first packet of the flow and the last time a packet was received, to prevent notifications for flows inactive for long time. Thus, 24 bytes are required per flow. For CMS and CMSS, the flow identifier is not required. Then, we only need to allocate a 4-byte counter (including a notification bit) and two 4-byte timestamps for each table entry. The first timestamp records the time for the first packet of the flow, and the other the time for the last packet of the flow. The second timestamp is used to determine whether the new packet corresponds to an exiting flow or to a new flow.

For a given memory size and summarization strategy, we vary the number of entries according to its corresponding row size. Memory size varies from very small (1KB), medium (16KB, 32KB), large (512KB) and very large (8 MB).

TABLE 1. Summary for RNP and CAIDA datasets.

Source	Date	Capture time	GB	Flows	Eleph. GB	Eleph. Flows
RNP	2018-05-04	21:50+5m UTC	110.58	732,409	63.06 (57%)	1464
CAIDA Dir-A	2019-01-17	13:00+5m UTC	150.15	11,406,158	54.17 (36%)	1979
CAIDA Dir-B	2019-01-17	13:00+5m UTC	150.77	25,659,863	36.73 (24%)	1497

We consider 1:256, 1:1024, 1:8192 and 1:32768 sampling rates. Finally, the minimum number of samples to detect an elephant flow, s, is set to 3 (our experiments show that $s = 2$ results in too few flows and there is no significant difference for s values ranging from 3 to 8).

We feed the simulator with three real traces, see Table 1, two obtained from CAIDA [23] and one from RNP.[3] The CAIDA datasets were captured in the Equinix datacenter in New York, with CAIDA Dir-A corresponding to traffic from Sao Paulo to New York and CAIDA Dir-B to the opposite direction. The RNP dataset is the outgoing traffic captured at the Curitiba Point of Presence.

Results are obtained from different executions over a 5-min period selected from each of the available traces, for a total number of 10 executions[4] for every set of parameter values.

B. RESULTS AND ANALYSIS

1) MANAGED TRAFFIC

In Figure 5 we show the traffic share associated to the elephants flows over the total amount of traffic for each detection method.

Many different configurations with sampling rates 1:256 and 1:1024 provide a large share of the maximum achievable *true positive* traffic for each trace. With sampling

[3]RNP is the National Research and Educational network of Brazil

[4]We found that the 95% confidence interval is in all cases below 6% for the parameter with higher variation, the number of *false positive* notifications. The interval is lower for the rest of the reported results.

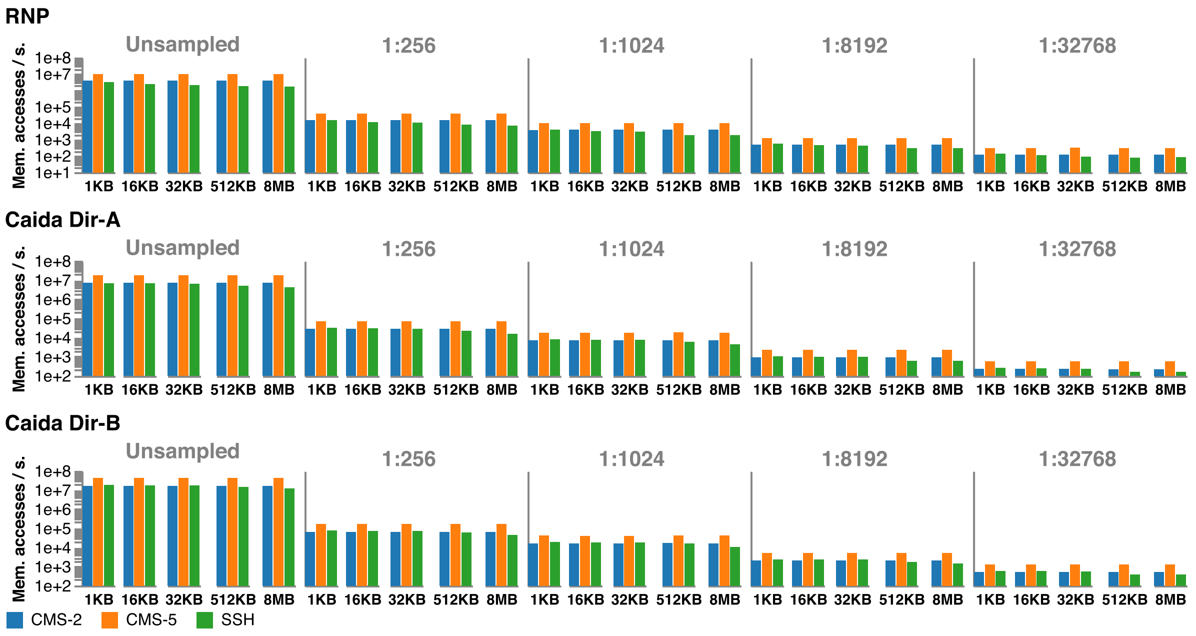

FIGURE 6. **Logarithm of memory accesses per second for RNP, CAIDA Dir-A and CAIDA Dir-B datasets. 1KB, 16KB, 32KB, 512KB and 8MB are the memory sizes for each configuration.**

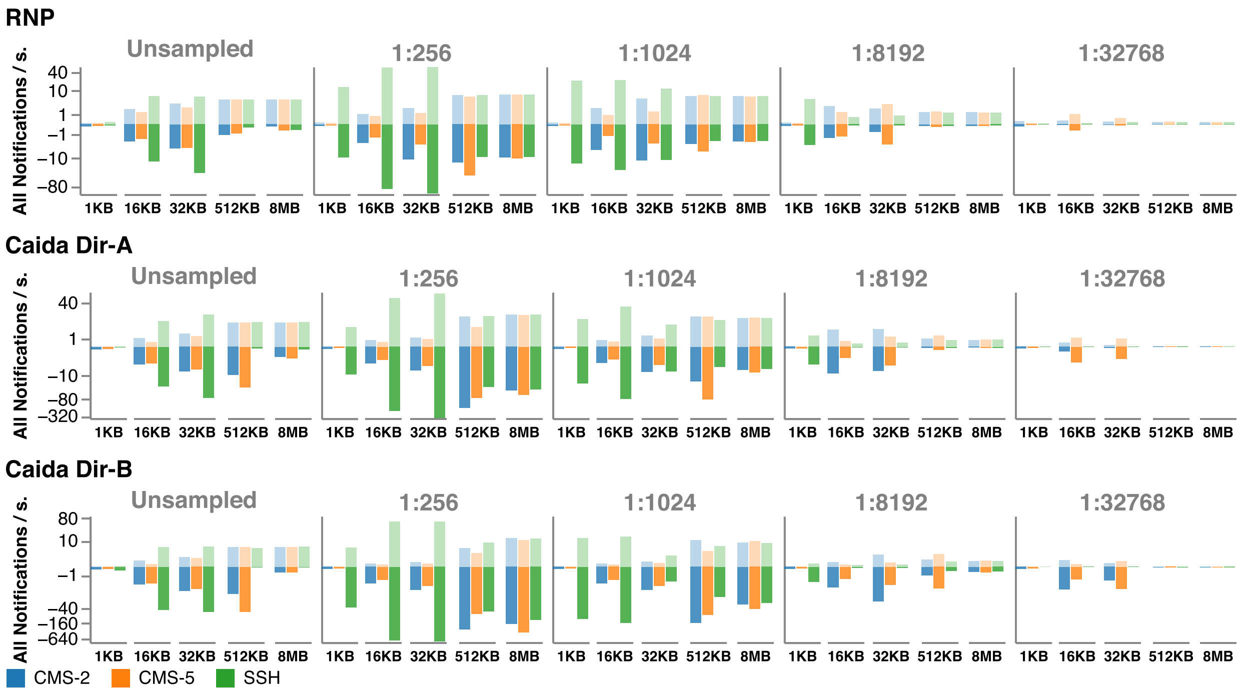

FIGURE 7. **Total number of notifications per second for RNP, CAIDA Dir-A and CAIDA Dir-B datasets. 1KB, 16KB, 32KB, 512KB and 8MB are the memory sizes for each configuration. Dim colors represent *true positives* and full colors *false positives*. In addition, negative values are used to represent *false positive* notifications.**

rates of 1:8192 or larger, the amount of managed traffic drops, specially for CAIDA traces. In most cases, being equal the rest of parameters, SSH(S) performs better in terms of amount of traffic managed than any CMS(S) option. This observation is consistent with previous studies [1]. When this is not the case, the difference is relatively small. This holds for all traces, and specially for small memory footprints. Note that when memory is scarce (16KB, 32KB) the use of small sampling rates (1:256 or 1:1024) results in more traffic than for the corresponding unsampled case. The low-pass filter function performed by sampling compensates for the difficulties of CMS(S)/SSH(S) to operate in low memory scenarios.

When the two flavors of CMS(S) are compared, in general CMS(S)-2 provides better results than CMS(S)-5.

The maximum amount of managed traffic is not necessarily achieved with the configuration with largest memory. This is because elephant flows are notified before the time specified by the minimum number of bytes transmitted and duration. First, the sampling procedure to detect flows can result in flows being notified before they would be in a non-sample scenario (see Figure 4). In addition, more notifications can also result from inaccuracies of the sketch and space saving schemes such as entry collisions and the inheritance of the counter values from dismissed flows. All these factors contribute to the number of false positives.

2) COMPUTATION COST

Computation cost is estimated in terms of the memory access rate. We use a logarithmic y-axis in Figure 6 to show the results. 1:256 sampling consistently shows a reduction of two orders of magnitude in the computing resources. Then, as the sampling rate increases, the gain decreases. The amount of allocated memory does not have a relevant impact in this cost.

While the total number of memory accesses is similar for CMS(S) and SSH(S), it is worth to note that the latter experiences a higher variability in the number of accesses required to process an incoming packet, that accounts for the identification of the minimum counter in the table.

3) NOTIFICATION RATE

The reference for the notification rate is the value required to track the real elephants: 4.9 notifications/s for RNP, 6.6 for CAIDA Dir-A and 5.0 for CAIDA Dir-B. The additional overhead is due to repeated notifications for the same flow identifier or notifications of non-elephant flows (*false positive* notifications).

The number of notifications per second is depicted in Figure 7. Note that the y-axis is logarithmic. Notifications are always well below a thousand per second.

False positive notifications are inherent to the use of flow summarizing strategies and appear when no sampling is applied. However, the use of sampling can increase its prevalence. In particular, in some cases, false positives exceed in an order of magnitude the number of *true positive* elephant flows (e.g., for CAIDA-B, sampling rate of 1:256). As sampling rate grows, false positives are reduced.

V. CONCLUSION AND FUTURE WORK

We have evaluated the performance of two new families of mechanisms that are suited for elephant flow detection with the scarce resources available in programmable switches, taking advantage on the use of sampled traffic. The resulting mechanisms allow the detection of elephant flows fully in the data plane. Sampling-enabled versions of sketch and counter based algorithms provide a whole set of new trade-offs between the share of managed traffic, the amount of memory required, the number of operations devoted to per-packet processing and the number of notifications to an external application.

We show by using three real traces with very different traffic patterns that most of the traffic associated to elephant flows can be detected and notified to interested applications in real-time, exclusively with data plane primitives, with restrained memory and computing requirements. In particular, if memory is at least 512KB, sampling at 1:256 or 1:1024 provides a large traffic share with low per-packet processing costs and reasonable increase in the number of notifications. At this operating point, space saving and sketches perform similarly. With less memory, e.g., 16KB, SSHS achieves a similar fraction of managed traffic at the cost of an increase in the number of notifications. Sampling combined with a very small table (1KB memory) can manage half of the traffic for the three traces considered (while the traffic drops almost to 0 for the equivalent unsampled case). Increasing the sampling rate to 1:8192 or beyond reduces too much the amount of traffic managed for all traces, so it is not recommended.

While the results provided in the paper inform about the different trade-offs involved in fully data-plane elephant detection, future work should involve deploying these mechanisms in hardware and evaluate the real performance that can be achieved. Note that each target hardware imposes its specific requirements in terms of available memory type (SRAM, CAM) and size, number of operations that can be executed per stage (for an RMT switch), or impact in the overall performance/managed traffic if there are not strict duration limits for data packet processing. This analysis will indicate which data structure sizes are feasible with current technology and its real performance. While we expect CMS(S) to allow implementations with large tables in both RMT and NPU switches, we are interested in evaluate the limits imposed in RMT architectures for SSH(S) table sizes, lower than for CMS(S).

Continuing with future work, we are interested in evaluating the feasibility of applying CMSS and SSHS to datacenter traffic traces. Kandula *et al.* [24] report that most of the traffic is associated to a small fraction of flows that last more than 10s. However, the possible high elephant flow count may exceed current switch capacities, so that additional conditions may be required to select the heaviest flows.

To improve accuracy in the detection of elephant flows, samples belonging to TCP flows can be inspected to obtain TCP sequence numbers. This information can be used to infer more accurately the number of transferred bytes. Such approach has been proposed for Planck [25] and Opensample [26].

REFERENCES

[1] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res.*, 2017, pp. 164–176.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX NSDI*, 2013, pp. 29–42.

[3] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 115–120.

[4] L. A. D. Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, "SDEFIX—Identifying elephant flows in SDN-based IXP networks," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2016, pp. 19–26.

[5] P. R. Torres-Jr, A. García-Martínez, M. Bagnulo, and E. Parente Ribeiro, "Bartolomeu: An SDN rebalancing system across multiple interdomain paths," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107117.

[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] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *Proc. 13th USENIX NSDI*, 2016, pp. 311–324.

[8] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 101–114.

[9] R. Ben Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 127–140.

[10] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 113–126.

[11] 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 *Proc. 2018 Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 561–575.

[12] Q. Huang, P. P. C. Lee, and Y. Bao, "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 576–590.

[13] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 334–350.

[14] F. Tang, H. Zhang, L. T. Yang, and L. Chen, "Elephant flow detection and differentiated scheduling with efficient sampling and classification," *IEEE Trans. Cloud Comput.*, early access, Feb. 26, 2019, doi: 10.1109/TCC.2019.2901669.

[15] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "IDEAFIX: Identifying elephant flows in P4-based IXP networks," in *Proc. IEEE GLOBECOM*, Oct. 2018, pp. 1–6.

[16] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. and Wang, "Fine-grained queue measurement in the data plane," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.* New York, NY, USA: Association for Computing Machinery, 2019, pp. 15–29, doi: 10.1145/3359989.3365408.

[17] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-*k* elements in data streams," in *Proc. Int. Conf. Database Theory*. Berlin, Germany: Springer, 2005, pp. 398–412.

[18] G. Cormode and M. Hadjieleftheriou, "Methods for finding frequent items in data streams," *VLDB J.*, vol. 19, no. 1, pp. 3–20, Feb. 2010.

[19] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1172–1185, Jun. 2020.

[20] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[21] P. R. Torres-Jr. (2020). *Python RMT and NPU Simulator—CMS, CMSS, SSH and SSHS*. [Online]. Available: https://bitbucket.org/torresweb/suleiman/

[22] P. Phaal and M. Lavine. (2004). *Flow Specification Version 5*.[Online]. Available: https://sflow.org/sflow_version_5.txt

[23] U. CAIDA. (2019). *The CAIDA UCSD Anonymized Internet Traces Dataset—Equinix-NYC 2019-01-17*. [Online]. Available: https://www.caida.org/data/monitors/passive-equinix-nyc.xml

[24] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2009, pp. 202–208.

[25] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. and Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 407–418, 2014.

[26] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, May 2014, pp. 228–237.

PEDRO RODRIGUES TORRES, JR. received the B.S. degree in computer science, the M.S. degree in informatics, and the Ph.D. degree in electrical and telecommunications engineering from the Universidade Federal do Parana (UFPR), Curitiba, Brazil, in 2003, 2007, and 2020, respectively.

From 2002 to 2017, he was a Network Manager with the Brazilian Academic Network in the State of Parana (RNP/PoP-PR), performing activities in the area of interdomain routing, optical networks, and data center infrastructure. Since 2009, he has been an Assistant Professor with the Professional and Technological Education Department, UFPR. From 2018 to 2020, he was an Invited Researcher with the Universidad Carlos III de Madrid (UC3M), Spain. He collaborated with the creation and operation of the Internet Exchange Points in the IX.Br Project. He is currently the coauthor of a book on linux and network operation. His research interests include Internet architecture and protocols, interdomain routing, programmable networks, load balancing, and network anomaly identification.

ALBERTO GARCÍA-MARTÍNEZ received the degree in telecommunication engineering, in 1995, and the Ph.D. degree in telecommunications, in 1999.

In 1998, he joined the Universidad Carlos III de Madrid (UC3M), where he has been an Associate Professor, since 2001. He has published more than 50 articles in technical journals, magazines, and conferences. He has coauthored three RFCs. His main interests include interdomain routing, transport protocols, network security, and blockchain technologies.

MARCELO BAGNULO received the Electrical Engineering degree from the University of Uruguay and the Ph.D. degree in telecommunications from the Universidad Carlos III de Madrid (UC3M), Spain. Since 2008, he has been a tenured Associate Professor at UC3M. He has published more than 80 articles in the field of advanced communications in journals and congresses, including IEEE INFOCOM, ACM SIGCOMM, ACM Mobicom, ACM IMC, and IEEE/ACM TRANSACTIONS ON NETWORKING. He is the author of 19 RFCs in the Internet Engineering Task Force (IETF), including the Shim6 protocol for IPv6 multihoming and the NAT64/DNS64 tools suite for IPv6 transition. He has 26 H-index and 4258 total citations. His research interests include Internet architecture and protocols, interdomain routing, and security. From 2009 to 2011, he was a member of the Internet Architecture Board.

EDUARDO PARENTE RIBEIRO received the bachelor's degree in electrical engineering and the M.Sc. and Ph.D. degrees from the Pontifical Catholic University of Rio de Janeiro, in 1990, 1992, and 1996, respectively. In 1995, he was an Invited Researcher at Vanderbilt University, USA. He has been a Professor with the Electrical Engineering Department, Universidade Federal do Parana, Brazil, since 1997. In 2006, he was a Visiting Professor with The University of British Columbia, Canada. His research interests include data communication, signal processing, and instrumentation.