Journal of Network and Systems Management
https://doi.org/10.1007/s10922-020-09556-7

BitMatrix: A Multipurpose Sketch for Monitoring of Multi-tenant Networks

Regis Francisco Teles Martins[1] · **Rodolfo da Silva Villaça**[2] · **Fábio Luciano Verdi**[1]

Received: 6 January 2020 / Revised: 21 May 2020 / Accepted: 11 July 2020

Abstract

Sketches are probabilistic data structures capable of summarizing and storing network data (packets, bytes, and flows), with a certain degree of accuracy, that have become widely popular for network measurement and monitoring. In this paper, we propose a new multi-purpose sketch, called BitMatrix, which is capable of working in multi-tenant networks. Owing to its multi-dimensional architecture, BitMatrix can differentiate between bit markings and byte/packet counting from different sources in a network. As a multi-purpose sketch, BitMatrix and its algorithms contribute to the literature by providing information regarding the paths traversed by each packet and are designed for use in multi-tenant networks. We also designed a statistical model to adjust the measurements owing to the probabilistic behavior of the sketches. Such a model is able to infer the standard error rate and approximate the BitMatrix counters to the real value. The adjusted BitMatrix measurement has a Mean Absolute Percentage Error of $\pm 6.14\%$. The BitMatrix sketch was implemented using P4 language and a simulator was also developed, that allowed its scaling using real traces from CAIDA in an NSF network topology.

Keywords Network monitoring · Sketches · Multi-tenant networks · Programmable networks

✉ Rodolfo da Silva Villaça
rodolfo.villaca@ufes.br

Regis Francisco Teles Martins
regisftm@gmail.com

Fábio Luciano Verdi
verdi@ufscar.br

[1] Department of Computing (DCOMP-So), Federal University of São Carlos (UFSCar), Sorocaba, SP, Brazil

[2] Industrial Technology Department (DTI), Federal University of Espírito Santo (UFES), Vitoria, ES, Brazil

1 Introduction

Accurate network measurements are of paramount importance for monitoring, management, capacity planning, and traffic engineering purposes. In addition, cloud computing has become very popular; thus, increasing the challenges in measuring a complex network infrastructure that supports multiple services and applications. Therefore, it is becoming increasingly urgent for an accurate and fine-grained measurement system to operate efficiently when considering its use in a complex infrastructure with multiple services and tenants[1].

Faced with the need for more effective ways to measure traffic that flows in a network link between network devices, understand its behavior to support traffic engineering decisions, a valuable contribution is made through the utilization of probabilistic data structures (sketches). An accurate measurement of traffic on links and between network devices can help network managers better distribute flows into the links as a load balancing action and to find bottlenecks in the network devices that may cause latency problems in the network. In addition, another interesting question that can be answered as a result of network monitoring is “given a packet and network topology (in the past), what was the path taken by this packet in the network?”

Sketches are probabilistic data structures used to store summarized information about network traffic. The two primary advantages of using sketches are (i) a lower memory usage in network devices (switch or router) when compared to traditional network monitoring tools, and (ii) configurable parameters (the size and occupancy of BitMatrix and the hashing function itself) that can be used to adjust the accuracy of the measurements[2–5].

Traditional traffic measurements, such as Netflow[6] and sFlow[7], have been used for reducing the amount of memory to store network packets for analysis. However, such solutions suffer with the overhead when copying these packets to the management interface, and it is occasionally also necessary to process such packets inside the network devices for summarization and forwarding to the management or control plane[8–10]. To avoid such an overhead, these tools usually apply sampling-based techniques, which typically have low accuracy and other weaknesses, such as: no support for data streaming algorithms to find the sources and destinations, or for the distributed version of the top-k problem, i.e., finding the globally *k*-most frequent flows, sources, or destinations in the network[11, 12].

There are also numerous measurement solutions for different applications in network monitoring, including heavy hitter detection[5, 13, 14], traffic change detection[15, 16], a flow size distribution estimation[17, 18], global iceberg detection[19], fine-grained delay measurements[20], an estimation of the flow-size distribution of traffic streams[18, 21], and an identification of anomalies in the network[15, 16, 22]. Differing from traditional measurement tools based on sampling, the use of sketches for traffic monitoring implies a trade-off between the amount of memory used to store data and required accuracy of the measurements.

Sketches offer a measurement technique that is suitable for application without sampling, processing every packet in the network, and achieving low memory usage and processing[23] with adjustable accuracy according to the sketch design[5, 16, 23–26]. Fast packet-processing technologies can be used to implement these sketches in the data plane, such as eBPF, XDP[27], and FPGA[28]. SmartNICs are network interface cards that are capable of having their functionality modified during runtime, thereby implementing different modes of operation. They can also be used to speed-up packet processing and enable the use of sketches.

Considering this scenario, many sketch-based solutions have begun to emerge on the market. In most cases, the sketches are used only for the marking of bits or bytes in a register, with all processing being accomplished by moving a few bytes to the management plane. In these cases, sketches have a lower overhead in the packet processing pipeline of the network devices and lower memory usage when compared to a traditional approach based on NetFlow and sFlow[29].

As an evolution of software defined networking (SDN), a Programming Protocol-Independent Packet Processor (P4) was presented as a high-level programming language for packet forwarding devices[30]. P4 allows the creation of several mechanisms for traffic measurement[31, 32], which include the implementation of sketches in programmable devices.

In this context, in this study, a new probabilistic structure called BitMatrix is designed and implemented based on the well-known bitmap and counter-array sketches, going further than the traditional monitoring process of counting packets and bytes. BitMatrix was created to support the multi-tenant monitoring capability, in addition to other features normally supported by this type of structure. To support multi-tenancy, the existent sketch solutions must have multiple instances installed on the network devices (at least one per tenant). These devices must also classify flows or packets and select the correct sketch to conduct the marking. This task must be achieved in each network device because multi-tenant scenarios are not intrinsic to such sketches. BitMatrix solves the problem with a single sketch, by generating the traffic classification according to the IP source (tenant) and separating such traffic by marking the correct position. BitMatrix uses an array of bitmaps combined with a storage method that enables the information to be segmented per tenant. In addition to the general probabilistic measurement, BitMatrix enables detailed analyses on a packet level in the network. In this study, we consider that each tenant has its own IP address sub-network, and can be identified by its address.

To achieve fine-grained measurements using sketches, we designed a machine learning (ML) model based on historical data to increase the accuracy of the measurements based on sketches. This model is then used in a polynomial regression algorithm to adjust the results and minimize the measurement errors caused by hash collisions. If a network device has infinite memory space and a perfect hashing function, this model is not necessary. However, because it is only a theoretical and non-practical situation, considering the same hashing function in all devices, the lower the memory space used for sketch instantiating, the greater the collision probability in the sketches and the lower the measurement accuracy. Under these real scenarios, this model is used to increase the measurement accuracy in a restricted memory network device.

BitMatrix was implemented in P4 language and deployed in a Mininet minimalist topology to be tested and evaluated. The results of this evaluation were described in a short paper[33], and the implementation is open source.[1] As the main difference between this study and the previous one[33], we designed and implemented a simulator to evaluate the BitMatrix with real traces from the Center for Applied Internet Data Analysis (CAIDA) and extensively analyzed the usage of linear and polynomial regression algorithms in this type of traffic. In summary, the contributions of this study are as follows:

- We designed a new probabilistic structure called BitMatrix that is capable of collecting the monitoring metrics in a multi-tenant infrastructure;
- We implemented BitMatrix using P4 language and made it available as open source for the community;
- We evaluated BitMatrix using real traffic traces from CAIDA in a NSFNet network;
- We designed a model to improve the measurements conducted using BitMatrix and minimize errors from hash collisions.

The remainder of this paper is organized as follows: In Sect. 2, some related studies are presented to highlight our contribution to the state-of-the-art in network monitoring using sketches. In Sect. 3, the design and implementation of the BitMatrix is detailed, which is a new sketch to be used for network monitoring. In Sect. 4, an evaluation of the BitMatrix is described, and an ML approach is presented for estimating the number of hash collisions and improving the measurement from BitMatrix. Finally, in Sect. 5, some concluding remarks and the future scope of studies in this area are provided.

2 Related Work

Network monitoring is used to understand issues or problems within a network environment. To understand, prevent, and resolve certain issues, there are numerous methods available for traffic monitoring. Studies on the monitoring of network traffic have evolved owing to an increase in traffic over the years. By monitoring, we can extract the relevant information for network management tasks such as attacks and anomaly detection[14, 34], forensic analysis[35], and traffic engineering[36–38]. Metrics at different levels are required for each management task, such as the flow size distribution[18], heavy hitters[36, 39], or detection of changes in traffic patterns[40, 16].

At a high level, there are two classes of techniques for obtaining performance metrics. The first is a traffic measurement using counters. The most widely used monitoring protocol for this purpose is the Simple Network Management Protocol, which collects real-time performance indicator values and with this available information maintains an in-memory database for queries; furthermore, it can send alarms when an indicator threshold is reached. The second class

[1] https://github.com/regisftm/bitmatrix.

of techniques involves two approaches to estimate the traffic metrics. The first approach is based on generic flow monitoring, typically using a protocol to collect traffic samples to estimate these metrics. The most popular protocols are NetFlow[6] and sFlow[41]. Although generic flow monitoring is generally sufficient to monitor traffic, previous studies have shown their low accuracy in generating more fine-grained monitoring metrics[17, 42, 43]. These limitations have led to an alternative sketching approach, where real-time algorithms and probabilistic data structures are designed to generate specific metrics[5, 15, 42–45].

Although studies on sketching and data streaming have made a significant contribution, in the long run, it remains impossible to sustain the continued creation of dedicated algorithms. The need to support new metrics demands the development of new algorithms, as well as the hardware and languages that support them. Tools such as OpenSketch[5] and SCREAM[3] provide libraries that reduce the implementation required and provide efficient resource allocation. In addition, because sketches are implemented on demand according to the set of metrics that require monitoring, blind spots are created for the metrics not monitored.

Most studies in this area have not addressed the fundamental problem of needing to create new sketches for each task. For example, UnivMon[46] was recently presented as a proposal for a framework that reconciles the generality and high fidelity for a broad spectrum of monitoring tasks, and FlexSketchMon[47] was introduced as a novel data plane architecture for collecting traffic flow statistics, providing a flow aggregation for monitoring applications. In this context, BitMatrix is a lightweight sketch that is able to answer some questions at the control plane. However, it was not designed for universal monitoring (as UnivMon) or flexible monitoring (as FlexSketchMon). The most valuable questions that can be answered with BitMatrix, which represent its main contributions to the current state-of-the-art approaches, are (i) given a packet (or a flow), what is the path traveled through the network during a specific time frame, and (ii) given only one sketch (BitMatrix), is it possible to obtain measurement statistics for different tenants in a multi-tenant network? We are not claiming that these questions cannot be answered using UnivMon/FlexSketchMon, but are simply convinced that it is not trivial (plug‘n’play) to do so without modifications to their implementations and streaming algorithms.

From the environment perspective, monitoring sketches can be implemented in SDNs[48, 49], as well as through the use of a packet-oriented language such as P4[30]. In addition to BitMatrix, there are some other implementations of network monitoring tools in P4, some using sketches[46, 48] and others not[31, 50, 51].

In Table 1, a summary of the related studies on network monitoring using sketches is shown. As shown in this table, there are few implementations of sketches in P4. None of the above studies are related to the capability of multi-tenant segmentation, and no correction mechanisms have been proposed to adjust the numbers collected by the sketches to the real number of packets processed in the data plane. This adjustment is of paramount importance for management tasks that require more accurate measurements.

Table 1 A summary of related studies on network monitoring using sketches

Work	Goals	Implementation and evaluation
Lossy data structure[18]	Traffic engineering	Trace analyses
Reversible sketches[16]	Traffic pattern detection	Trace analyses
FlexSample[43]	Anomaly detection, traffic monitoring	Trace analyses
Sketch-based change detection[15]	Anomaly detection	Trace analyses
OpenSketch[5]	Anomaly detection, traffic monitoring	NetFPGA, trace analyses
Scream[3]	Traffic engineering	Trace analyses
UnivMon[46]	Flow monitoring	P4, trace analyses
SketchVisor[48]	Top-k algorithm	Open vSwitch (OvS)
FlexSketchMon[47]	Flow monitoring, traffic monitoring	NetFPGA
BitMatrix	Flow monitoring, traffic monitoring, path detection, designed for multi-tenant monitoring	P4, trace analyses

3 Design and Implementation of BitMatrix

As mentioned earlier, BitMatrix is composed of two common sketches: bitmaps and counter arrays. A bitmap is a sequence of n bits with a fixed length, as shown in Definition 1. Thus, each bit position will create an index for the bitmap, making it possible to refer to a specific position by using this index. The bitmap is used to mark a fingerprint of each packet processed in each network device. To determine which index position a packet should occupy, we hash some fields from the header of each packet.

$$bitmap = \{bit\_1, bit\_2, bit\_3, \dots, bit\_n\} \tag{1}$$

The packet fields, used as input for the hashing algorithm, must be able to represent the packet uniquely and may not change across the hops during the forwarding process. The hash will be calculated using the invariant portion of the packet and the first 8 bytes of the payload, namely, TCP, UDP, or another subsequent layer, if present. IP fields that can be modified during the packet forwarding across the network are not used as an input because this would result in a different index for the same packet along its path in each network device. For the same hashing function, the lower the size of the bitmap in bits (n), the lower the measurements accuracy owing to the increase in the probability of hashing collisions. Because bitmaps are probabilistic data structures that can only be used for distinct counting, there are no false positives. A false positive is a relevant metric for sketches used in a set membership problem, such as a Bloom Filter[28], and none of our intended uses for BitMatrix requires such an answer.

According to Snoeren[52], the first 28 invariant bytes of a packet are sufficient to differentiate almost all non-identical packets in a network. With each packet represented by an index created by hashing these fields, the bitmap was used to store fingerprints of the packets by setting to 1 (one) the bit corresponding to its index. Initially, the bitmap has all positions set to zero, and for each packet processed, one position, defined by the hashing of each packet, is set to 1. In this way, every time a packet is processed, it will generate a hash number and will change the value of the bit in the position corresponding to its index to 1. Because bitmap has fewer positions than the hashing values, a modulo function needs to be used to adjust the corresponding index. Modern technologies for fast packet-processing can be used to implement such sketches directly in the data plane, including eBPF and XDP[27], FPGA[28], and Netronome SmartNICs[53]. It is important to reinforce that only hashing and marking operations are implemented in the data plane, and all processing operations are made in the control plane.

Algorithm 1 demonstrates the procedure used to process a packet, generate a hash, and determine the corresponding position for that packet in the bitmap vector.

Algorithm 1 Marking the position for every packet in the bitmap vector

```
1:  procedure BITMAP VECTOR POPULATION
2:      Input: First 28 invariant bytes of the packet (i),
3:      Input: Bitmap length (n),
4:      Output: Corresponding bitmap vector position set to 1,
5:      $hash\_value = hash\_function(i)$
6:      if $hash\_value > n$ then
7:          $bitmap\_position = \mod(hash\_value, n)$
8:      else
9:          $bitmap\_position = hash\_value$
10:     end if
11:     $bitmap[bitmap\_position] = 1$
12: end procedure
```

Furthermore, a counter array sketch is used to store the total length of each packet processed by the network device. A counter array is a sequence of n counters with a fixed length, as shown in Definition 2. It will use the same hash value calculated for the bitmap as the index for the counter array. In this way, it is possible to recover the corresponding size (total length) of each packet by using the index from the bitmap sketch.

$$counter - array = \{counter\_1, counter\_2, \dots, counter\_n\} \tag{2}$$

3.1 Multi-tenant Segmentation

The use of a bitmap is limited to indistinctly marking the positions for the packets (one bit), however, and because a bitmap is composed of a single vector of bits, a segmentation is not allowed. In a multi-tenant environment, it may be desirable to use one bitmap to count the packet traces for each tenant. Instead of using several vectors individually, we propose the use of a bitmap matrix, which we call BitMatrix. BitMatrix is a set of m bitmaps of size n, where m is the number of tenants and n is the size of the bitmap. In this context, BitMatrix will occupy $m \times n$ bits of

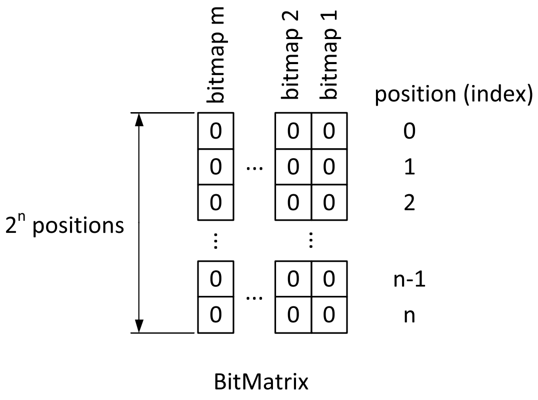

Fig. 1 BitMatrix represented as a set of *n* bitmaps

memory in the network device. Actually, bitmaps can be created with any number of positions. Unfortunately, owing to the actual limitations of the P4 architecture, to modify this number it is necessary to rewrite the source code and recompile and embed it into the network device. Figure 1 shows the Bitmatrix structure, which is also represented in Eq. 3. Because bitmaps will be allocated in the memory of the network device, a power of 2 was selected to count the number of positions in BitMatrix and achieve an easier memory allocation and management.

$$BitMatrix = \{bitmap\_1, bitmap\_2, bitmap\_3, \dots, bitmap\_m\} \tag{3}$$

The main benefit of this method is the ability to segment the packet counting using a single probabilistic structure. This segmentation can occur in different ways, such as per sub-network, per network or transport protocol, or per application port. In this paper, in a multi-tenant network scenario, BitMatrix uses a per tenant segmentation approach, where every packet originated by a specific tenant, identified by the packet source IP address, will be marked in the same bitmap in BitMatrix. The number of bitmaps in BitMatrix will increase linearly according to the number of tenants being monitored in the network, and will determine how many bits BitMatrix will use in each position.

As an example, considering a case in which BitMatrix is used to monitor traffic of four tenants, consequently, it has 4 bits (one for each tenant). The first bit is for bitmap_1, second bit is for bitmap_2, third bit is for bitmap_3, and the fourth bit is for bitmap_4. Still, in this example, each line of BitMatrix has a single value that can vary from 0 to 15 ($2^m - 1$, where m is the number of tenants): if there are no packet traces hashed in that position in any of the bitmaps, the value will be 0 (0000_2); if all bitmaps have packets hashed in the same position, the value will be 15 (1111_2). As a consequence of this design, BitMatrix grows linearly according to the number of tenants, i.e., if the monitored infrastructure has m tenants, our BitMatrix will have m bitmaps stored in it.

Thus, for the marking procedure, it is insufficient to determine the position of the packet (its index in a bitmap) in BitMatrix. It is also necessary to determine which bitmap should be used to hash the fingerprint (a bit) of this packet. We populate the bitmaps by mapping the IP addresses to binary numbers depending on the quantity of the tenants. In the example mentioned above, with four tenants (e.g., IP A, IP B, IP C, and IP D), we mapped IP address A to bitmap_1 with a value of 0001, IP address B to bitmap_2 with a value of 0010, IP address C to bitmap_3 with a value of 0100, and IP address D to bitmap_4 with a value of 1000. This mapping is conducted manually in the control plane and is then loaded by Algorithm 2, as shown in line 3. The IP source address of every packet entering in the switch is stored in the *pkt* variable (line 4 in Algorithm 2) and then compared with the bitmap vectors containing all IP addresses of the tenants previously mapped (lines 12–17). Algorithm 2 shows a simplified version of the procedure for finding and marking the correct bitmap in BitMatrix.

These values are written in the BitMatrix position by using a logical disjunction operator OR (line 22 in Algorithm 2), avoiding any loss of packets previously marked in the other bitmaps (tenants). To summarize, to map a packet to a bitmap in BitMatrix we use a combination of Algorithm 1 (to determine the position in the bitmap) and Algorithm 2 (to determine in which bitmap the packet will be mapped).

Algorithm 2 Finding and marking the tenant's bitmap in BitMatrix

```
1: procedure BitMatrix vector population
2:     Input: First 28 invariant Bytes of the packet (i)
3:     Input: All externally populated bitmaps: bitmap_1, bitmap_2...bitmap_n
4:     Input: Tenant identificator of the current packet (pkt)
   Output: Corresponding bit in BitMatrix associated position set to 1
5:     $hash\_value = hash\_function(i)$
6:     if $hash\_value > BitMatrix\_length$ then
7:         $BitMatrix\_position = \mod (BitMatrix\_length, hash\_value)$
8:     else
9:         $BitMatrix\_position = hash\_value$
10:    end if
11:    $BitMatrix\_Stored\_Value = BitMatrix[BitMatrix\_position]$
12:    if $pkt = bitmap\_1$ then
13:        $BitMatrix\_value = 1$ [0001]
14:    else if $bitmap = bitmap\_2$ then
15:        $BitMatrix\_value = 2$ [0010]
16:    else if $bitmap = bitmap\_3$ then
17:        $BitMatrix\_value = 4$ [0100]
18:        ...
19:    else
20:        $BitMatrix\_value = m$
21:    end if
22:    $BitMatrix[BitMatrix\_position] = BitMatrix\_Stored\_Value$ OR $BitMatrix\_value$
23: end procedure
```

Using this algorithm, it is possible to segment the packets stored in BitMatrix by creating a process to distinguish between different tenants. By doing so, our solution can identify which tenant originated the stored fingerprints of the packets.

3.2 Using Data from Bitmatrix

The data stored in BitMatrix is of paramount importance for network monitoring and provide a powerful tool to obtain interesting information from the network to be used by applications in the control plane. Some examples of information that can be extracted from Bitmatrix are as follows:

- The number of unique packets crossing the network;
- Their origin and destination;
- The network devices responsible for routing such packets;
- The heavy-hitters and which tenant they belong to.

BitMatrix, allied with the network topology information, is sufficient to conduct the proposed monitoring and a further analysis. Given an identical BitMatrix on every device in the network, it is possible to have the same packet fingerprint marked precisely at the same position of BitMatrix installed on each device responsible for forwarding such packets.

Within a fixed time period, the control plane should collect BitMatrix with all hashed data stored on it from all network devices. To retrieve BitMatrix, a command line interface (CLI), available in the Behavioral Model version 2 (BMv2) software switch is used. In our testbed, the CLI is connected to the Thrift RPC server running in each emulated network device.

Once the control plane has collected the data, several analyses can be conducted. The number of packets processed in a specific device, per tenant and in total, can be estimated by analyzing the information from BitMatrix by summing the bits equal to 1, which represent packet fingerprints stored in BitMatrix. The number of packets counted in BitMatrix will always be less than the number of packets processed by the device, given the probability of a hash collision occurring, as discussed later in Sect. 3.3.

To estimate the number of packets for a specific tenant, it is necessary to select the corresponding bitmap in BitMatrix, and then conduct the calculation. The total number of packets processed by a specific network device can be estimated by counting the number of bits marked in the entire BitMatrix. Assuming that all packets from Tenant_A are marked in bitmap_1, Eq. 4 represents the total number of packets sent from Tenant_A, where n is the size of the bitmap for Tenant_A.

$$\text{Total number of packets from Tenant\_A} = \sum_{i=1}^{n} bitmap\_1[i] \tag{4}$$

Equation 5 expresses the total number of packets processed by the network device from where BitMatrix was collected. Here, m is the number of tenants in the network, and n is the size of the bitmap for each tenant.

$$\text{Total number of packets from all tenants} = \sum_{j=1}^{m} \sum_{i=1}^{n} BitMatrix[j][i] \tag{5}$$

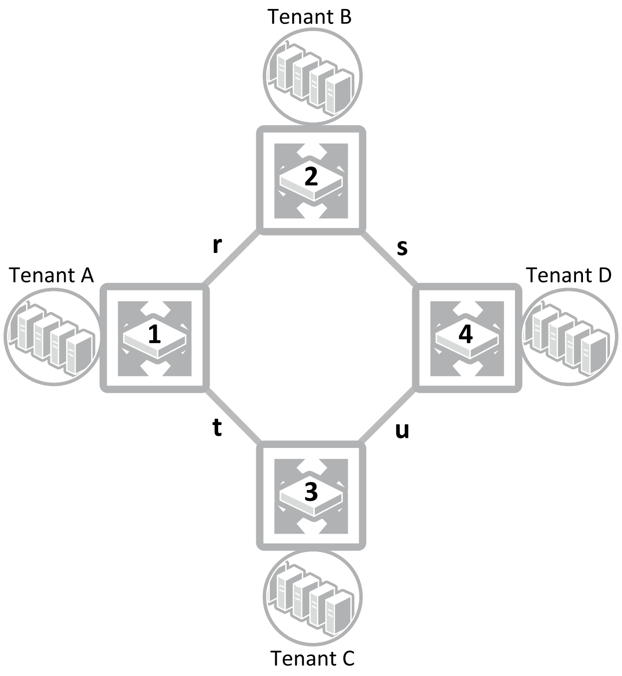

Fig. 2 Network proposed to illustrate the task of analyzing the BitMatrix data

To obtain the number of unique packets processed in the network, per tenant and in total, a logical disjunction operator (OR) can be used in the bitmaps related to each tenant, collected from BitMatrices across the network devices during the same time period. The logical operator OR will avoid counting the same packet more than once. Equation 6 shows the sum of the bits resulting from the logical disjunction of all bitmaps correspondent to Tenant_A, collected from all BitMatrices in the network. Here, n is the size of the bitmap. The total number of unique packets processed in the network can be obtained by summing the total number of unique packets of all tenants.

$$\text{Total number of unique packets of Tenant\_}A = \sum_{i=1}^{n} BitMatrix[A][i] \tag{6}$$

Because each bitmap in BitMatrix corresponds to a different tenant, the source of a packet can be determined according to the corresponding bitmap in which the packet fingerprint is marked. The destination can be estimated by analyzing which network devices have stored the packet fingerprint at the same position in BitMatrix. To do so, the network topology knowledge becomes necessary. Given such information, it is possible to create a traffic matrix between tenants and the network devices. As an example, consider the network from Fig. 2. In this context, based on the collected

BitMatrix from all network devices, given a fixed and known time period, it is possible to answer the following questions:[2]

- How many packets were exchanged between Tenant_A and Tenant_B?
- What was the network device with the highest load?
- What link had the higher throughput? In addition, what was its average throughput?

3.2.1 How Many Packets are Exchanged Between Tenant_A and Tenant_B?

To answer this question, two different analyses must be conducted. The first aims to identify the packets sent from Tenant_A to Tenant_B, and the second aims to identify packets sent from Tenant_B to Tenant_A. To identify packets sent from Tenant_A to Tenant_B, once knowing the network topology and assuming the shortest path between them, we can infer that these packets are the sum of the bits in the corresponding bitmaps of BitMatrix from devices 1 and 2 but are not present in devices 3 and 4. Equation 7 shows this logical operation. In this equation, bitmap$_{x,y}$ represents the bitmap of tenant x in the device y, and n is the size of the bitmap (in bits). As an example, bitmap$_{A,1}$ in Eq. 7 means the bitmap of Tenant_A in Router 1.

$$\text{Total A} \rightarrow \text{B} = \sum_{i=1}^{n} \{bitmap_{A,1}[i] \wedge bitmap_{A,2}[i] \wedge \neg bitmap_{A,3}[i] \wedge \neg bitmap_{A,4}[i]\} \tag{7}$$

Second, we use the same algorithm to estimate the number of packets sent from Tenant_B to Tenant_A using the bitmaps associated with Tenant_B. Equation 8 shows these logical operations.

$$\text{Total B} \rightarrow \text{A} = \sum_{i=1}^{n} \{bitmap_{B,1}[i] \wedge bitmap_{B,2}[i] \wedge \neg bitmap_{B,3}[i] \wedge \neg bitmap_{B,4}[i]\} \tag{8}$$

Finally, to answer the question, we need to determine the total number of packets that flow between Tenant_A and Tenant_B. In this case, we need to sum the results from Eqs. 7 and 8. Equation 9 shows the final result and the answer to the above question.

$$\text{Total A} \leftrightarrow \text{B} = (\text{Total A} \rightarrow \text{B}) + (\text{Total B} \rightarrow \text{A}) \tag{9}$$

3.2.2 What Network Device has the Highest Traffic Load?

It is possible to estimate the network device with the highest traffic load by estimating the number of packets processed for each during the observation time. To obtain

[2] Equations 7, 8, 9, 10, and 11 are expressed based on the topology shown in Fig. 2 for sake of the readers. However, they can be generalized to any tenant, device, or link.

this estimation, it is necessary to sum the bits of all bitmaps in the BitMatrix of all routers/switches in the network. Equation 10 demonstrates how to estimate the total number of packets for a specific device (Dev_1). In this equation, *m* is the number of tenants and *n* is the size of the bitmap in BitMatrix.

$$\text{Total number of packets in Dev\_1 } = \sum_{j=1}^{m} \sum_{i=1}^{n} BitMatrix[j][i] \tag{10}$$

The same procedure is used to estimate the total number of packets of all other devices in the network. Once calculated, we can sort them by the total number of packets, determining which was the network device with the highest load in the network during the observation time period.

3.2.3 What is the Link with the Highest Throughput? In Addition, What was its Average Throughput?

To determine which link had the highest throughput, we need to identify the packet fingerprints present in the BitMatrices of a pair of connected devices during the same time period. In other words, all packets present in the BitMatrices of both Dev_1 and Dev_2 may have used the link *r* (Fig. 2) to travel from one device to another. It is possible to determine the packet fingerprints that are present in both devices by using a logical conjunction operation (AND) between the two BitMatrices. Thus, to find the total number of packets that crossed this link, we need to sum the bits of BitMatrix generated by the AND operation. Equation 11 shows the operation used to find the total number of packets that crossed the link `r`. In the equation, BitMatrix_1 is the BitMatrix of Dev_1 and BitMatrix_2 is the BitMatrix of Dev_2, where *m* is the number of tenants and *n* is the size of the bitmap of both BitMatrices.

$$\begin{aligned} &\text{Total number of packets between Dev\_1 and Dev\_2} \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} \{BitMatrix_1[j][i] \wedge BitMatrix_2[j][i]\} \end{aligned} \tag{11}$$

Although it is possible to go from Dev_1 to Dev_2 through links *t*, *u*, and *s* due to an abnormal situation, e.g., a link failure, this will not happen. In this example, we do not considered such a possibility, although the same logical conjunction operation in the BitMatrices of Dev_1, Dev_2, Dev_3, and Dev_4 (all devices in this alternative path) may be used to sum the bits of the resulting BitMatrix. This sum is an estimate of the number of packets traveling from Dev_1 to Dev_2 by using the links *t*, *u*, and *s*.

To estimate the average throughput of each link, the counter array sketch must be used. As discussed before, once the network device hashes the fingerprint of the packets (a bit) in BitMatrix, it uses the same index to store the total length of the packet in the correspondent counter array. Both counter arrays of Dev_1 and Dev_2 store the packet length of each packet that passes through link *r*. To estimate the total amount of bytes crossing this link, it is possible to use the resulting BitMatrix of the logical conjunction operation to find what positions to read and what values to sum in the counter array. Finally, given the total packet length from the counter array, it is possible to calculate the average throughput by dividing the total amount of bytes by the time frame of the observation.

3.3 Caveats and Limitations

Even using the invariant bytes of a packet for a hash calculation, it is possible that the hashing of two different packets will result in the same value. This occurrence is called a hash collision and creates a gap between the number of packets actually forwarded by the network device and total number of positions marked (equal to ‘1’) in the bitmap. There are three main factors that can cause a variation in the number of hash collisions in the proposed scenario: (a) the size of the bitmap, or in other words the number of positions available in the vector, (b) the occupation of the bitmap when marking a new position, and (c) the type of hash function used to generate the hash value.

The size of the bitmap and size of the hash value are related in the sense that there is no point in setting a bitmap with more positions than a hash value. Positions beyond the maximum hash value will never be used. Nevertheless, setting a bitmap smaller than the maximum hash value will demand a modulo operation, using the bitmap size and the resulting hash value to determine the offset for the present packet. Finally, we need to consider the bitmap occupancy. The more occupied a bitmap is, the greater the chances of a hash collision. In Sect. 4.3, we detail the mechanism designed to adjust the hash collision.

The overhead related to the usage of Bitmatrix may be the focus of future analysis. We can state that the main overhead will come from the hash algorithm used to determine the position in BitMatrix. In this context, there are other studies that have conducted extensive analyses related to the performance of P4 hashing algorithms in both software and hardware. We point the readers to[46, 54] and the references therein as example studies to achieve a better comprehension regarding the overhead of the probabilistic structures.

4 Evaluation and Results

As stated before, the P4 implementation of BitMatrix and its respective evaluation were presented in our previous study[33]. This paper presents a simulator whose purpose is to scale up the previous results by simulating network devices with a BitMatrix sketch in a real-world network topology with traffic traces from the passive equinix-sanjose CAIDA[3] dataset. The simulator is designed in Python and receives the packet traces as input, as well as the network topology and the data collected from BitMatrix. The output generated by the simulator is a set of CSV files imported into Tableau Software[55], where all the analyses are

[3] http://www.caida.org/data/passive/passive_2012_dataset.xml.

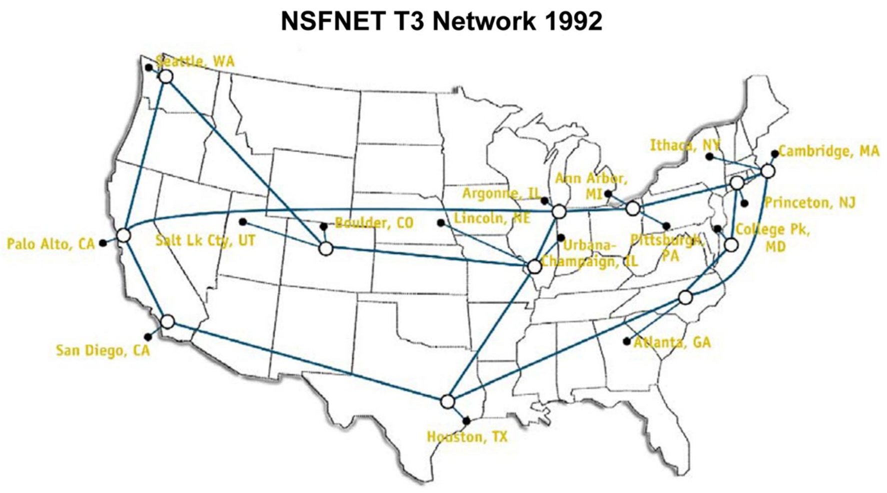

Fig. 3 NSFNet'92 network used in the evaluation of the BitMatrix sketch

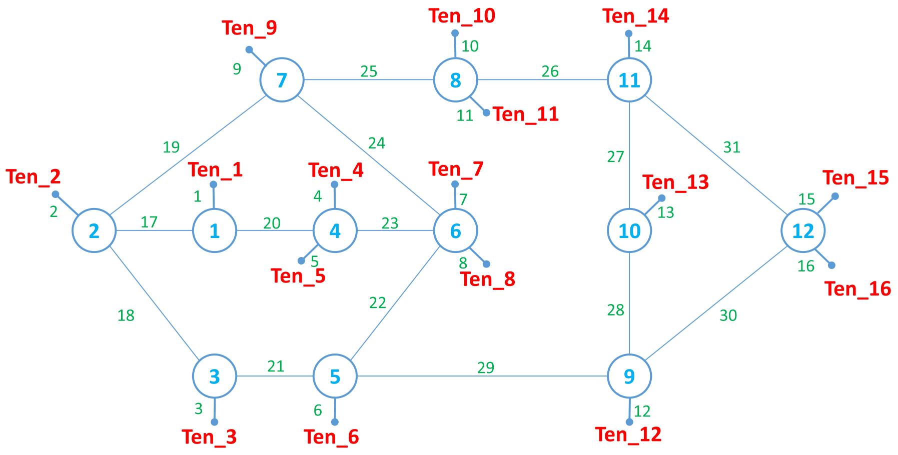

Fig. 4 Network topology showing tenants, routers, and links

conducted. We used a computer with an i9 9900k CPU and 64 GB of RAM for running the simulator.

To create a more realistic model for the tests, a network topology based on NSFNet'92 was used, as shown in Fig. 3.

The NSFNet'92 network topology is represented as a graph for the simulation, as shown in Fig. 4. The NSFNet'92 sites become tenants, numbered from 1 to 16. Routers and links are also numbered, having a total of 12 routers and 31 links interconnecting the tenants and routers. This network topology was taken as reference in an evaluation of BitMatrix presented herein.

Table 2 Tenants and their assigned group of network prefixes

Tenant	Network prefixes assigned to the tenants
Tenant_1	180.0.0.0/8, 128.0.0.0/8
Tenant_2	223.0.0.0/8, 208.0.0.0/8, 55.0.0.0/8, 108.0.0.0/8
Tenant_3	48.1.159.0/24, 151.0.0.0/8, 48.0.0.0/8
Tenant_4	145.0.0.0/8, 158.0.0.0/8, 54.0.0.0/8
Tenant_5	61.0.0.0/8, 184.0.0.0/8, 181.0.0.0/8, 203.0.0.0/8
Tenant_6	48.1.136.0/24, 132.0.0.0/8, 177.0.0.0/8, 186.0.0.0/8, 197.0.0.0/8
Tenant_7	48.1.137.0/24, 83.0.0.0/8, 34.0.0.0/8, 141.0.0.0/8, 116.0.0.0/8
Tenant_8	48.2.0.0/8, 155.0.0.0/8, 49.0.0.0/8, 187.0.0.0/8, 37.0.0.0/8
Tenant_9	135.0.0.0/8, 144.0.0.0/8, 60.0.0.0/8, 220.0.0.0/8, 236.0.0.0/8, 118.0.0.0/8, 113.0.0.0/8
Tenant_10	41.0.0.0/8, 142.0.0.0/8, 147.0.0.0/8, 247.0.0.0/8, 50.0.0.0/8, 32.0.0.0/8, 125.0.0.0/8
Tenant_11	178.0.0.0/8, 70.0.0.0/8, 221.0.0.0/8, 148.0.0.0/8, 248.0.0.0/8, 219.0.0.0/8, 152.0.0.0/8, 138.0.0.0/8, 115.0.0.0/8
Tenant_12	53.0.0.0/8, 150.0.0.0/8, 48.1.156.0/24, 201.0.0.0/8, 42.0.0.0/8, 228.0.0.0/8, 68.0.0.0/8, 104.0.0.0/8, 35.0.0.0/8, 85.0.0.0/8
Tenant_13	39.0.0.0/8, 159.0.0.0/8, 183.0.0.0/8, 36.0.0.0/8, 33.0.0.0/8, 112.0.0.0/8, 182.0.0.0/8, 242.0.0.0/8
Tenant_14	143.0.0.0/8, 218.0.0.0/8, 79.0.0.0/8, 78.0.0.0/8, 77.0.0.0/8, 253.0.0.0/8, 254.0.0.0/8, 163.0.0.0/8, 98.0.0.0/8, 109.0.0.0/8, 105.0.0.0/8
Tenant_15	176.0.0.0/8, 40.0.0.0/8, 140.0.0.0/8, 190.0.0.0/8, 149.0.0.0/8, 43.0.0.0/8, 146.0.0.0/8, 231.0.0.0/8, 174.0.0.0/8, 48.1.226.0/24, 80.0.0.0/8, 84.0.0.0/8, 134.0.0.0/8, 131.0.0.0/8, 210.0.0.0/8
Tenant_16	0.0.0.0/0

In addition, a routing table was installed in the routers to specify the path that a packet needs to travel across the network to go from a source to a destination. The same path is used for the upstream and downstream traffic. In this table, the shortest path was used for all situations.

A different dataset and a different topology were not used because they are not considered as variables in our proposal. The hash is calculated using the invariant portion of the packet and the first 8 bytes of the payload. Using another dataset or topology, when considering the same methodology to assign an ingress and egress point in the network, the measured values will change; however, neither the algorithm used to estimate the route followed by each packet nor the ML algorithm applied to improve the accuracy of the measurements will be altered.

4.1 BitMatrix Setup

Once the network topology is selected and the routing table is configured, the next step is to distribute packets from the trace to the tenants. source The IP addresses of the packets were distributed in 16 groups of networks (one group per tenant) in such a way that each group has a similar number of packets to balance the traffic between routers. Table 2 shows the network prefixes assigned to each tenant. Packets with a

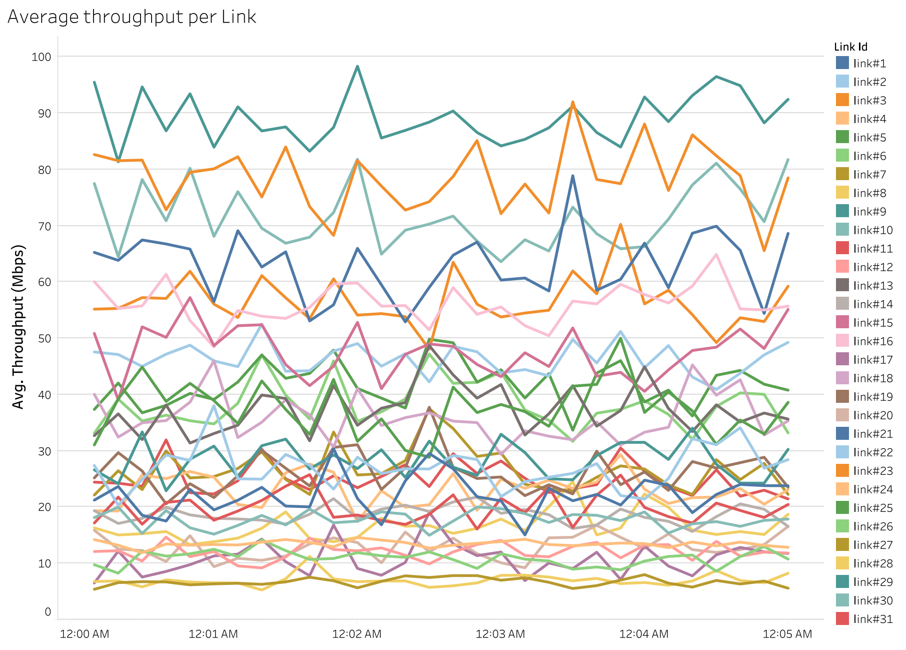

Fig. 5 Average throughput per link. The monitoring time frame is set to 10 s

source IP address not belonging to any of the prefixes assigned from Tenant_1 to Tenant_15 were assigned to Tenant_16, which is a type of "catch-all" tenant.

To determine how many packets should be processed before collecting the statistics and resetting the counters, it is necessary to configure the simulator. Once the bandwidth of the links was arbitrarily defined as 100 Mbps, the question of how many packets per second per tenant does the simulator need to process the respective the bandwidth of the links was derived. The monitoring time frame for BitMatrix was configured as 10 s, and the total simulation time was set to 30 min. To calibrate the simulation and answer the previous question, an empirical evaluation of the maximum rate of *pps* for each tenant was conducted, and Fig. 5 shows the result. On average, the simulator processed a batch of 430,000 packets every 10 s resulting in a rate of 43,000 pps. With this number of packets processed, we can maintain the simulated traffic under a pre-determined link capacity of 100 *Mbps*.

The next configuration step is related to adjusting the BitMatrix parameters on each router: its length (in bits) and the epoch (monitoring time frame). According to the previous evaluation, the epoch was set to 10 s, or 430,000 packets for all tenants. Based on this evaluation, the length of the bitmaps of the BitMatrix was set to 65,536 bits, thus, the total size of each Bitmatrix in each device is $16 \times 65{,}536$ bits (131 KB), which is quite acceptable considering the size of the memory used in current physical forwarding devices. This size was chosen to maintain a low level of occupancy of each BitMatrix as well as to avoid a high number of hash collisions.

It is important to note here that *m*, i.e., the number of tenants in the network, does not have an effect on the results, and will only influence the amount of memory used

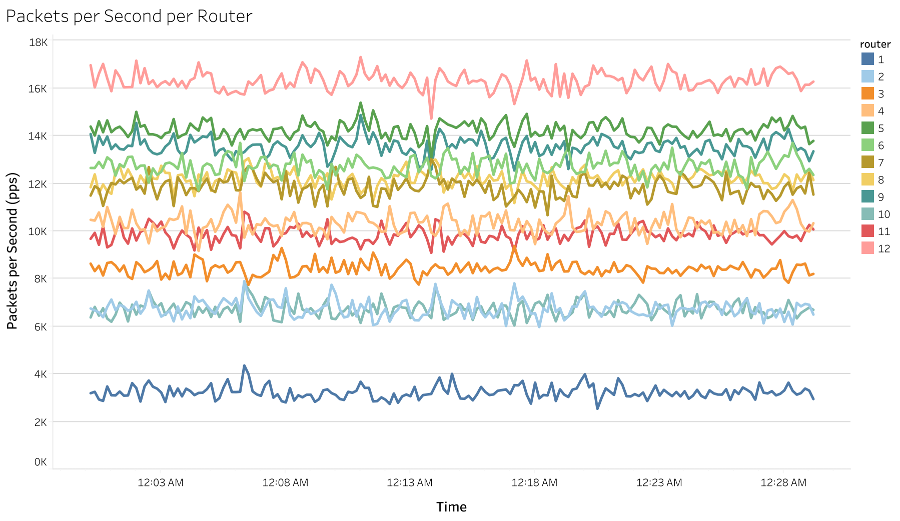

Fig. 6 Average throughput per router, in *pps*. The monitoring time frame is set to 10 s

by BitMatrix because a bitmap exists for each tenant. In summary, m affects only the total size of Bitmatrix. Regarding n, i.e., the size of the bitmaps in BitMatrix, it will have an influence on the hashing collision rate because the lower the size of the bitmap, the greater the probability of having a hashing collision to mark the packets into the bitmap. Increasing the collision rate will increase the error in our counting estimation and reduce the accuracy of our measurements. We conducted an evaluation on the occupation versus collision rate in two previous studies[33, 56].

In both[33, 56] an evaluation of the relation between three variables was conducted, namely, the size of the bitmap, its occupation, and the throughput of the monitored network. These three variables are tied to the collision rate of the hashing function as follows: the smaller the size of the bitmap, the greater the probability of having collisions marking the packets in the bitmap; the greater the occupancy of the bitmap, the greater the chance of collisions occurring; and finally, the greater the throughput of the network, the faster the occupancy of the bitmap. It is possible to control the size of the bitmap and the desired occupancy for any monitoring system based on sketches, despite the throughput of the network being beyond the control of the network manager, as it follows different loading along the time. For this reason, it is not possible to generate a fixed equation that points to the best configuration n for a specific network. The network manager must maintain these variables and continuously monitor the accuracy of the measurements and modify its BitMatrix configuration.

4.2 BitMatrix Evaluation Overview

To analyze the statistics generated by the BitMatrices on each router, we used Tableau software. The information extracted from BitMatrix using CLI and the Thrift RPC server available in each software switch was compared to the real information

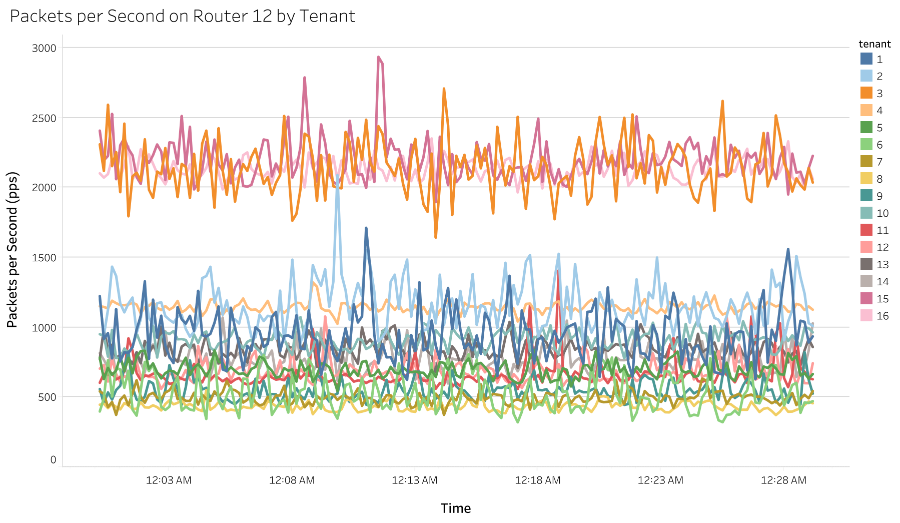

Fig. 7 Average throughput per tenant on Router_12, in *pps*. The monitoring time frame is set to 10 s

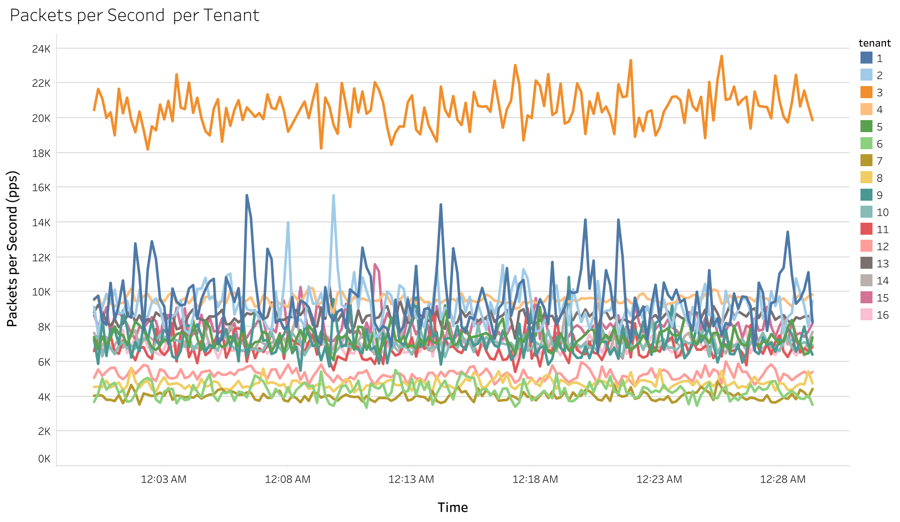

Fig. 8 Average throughput per tenant (all routers), in *pps*. The monitoring time frame is set to 10 s

obtained from packets and byte counters in the simulator. This comparison is presented in Sect. 4.3. The simulator processed a total of 77,400,000 packets from the trace. Figure 6 shows the average *pps* rate per router during the 30 min simulation.

Analyzing Fig. 6, the router with a higher *pps* rate is Router_12. To observe the traffic in Router_12, the details of the traffic in this router, broken down by tenant, are shown in Fig. 7.

From Fig. 7, it is possible to identify that tenants 3, 15, and 16 are the heavy hitters of router_12, i.e., the tenants that send/receive more traffic (measured in *pps*) to/

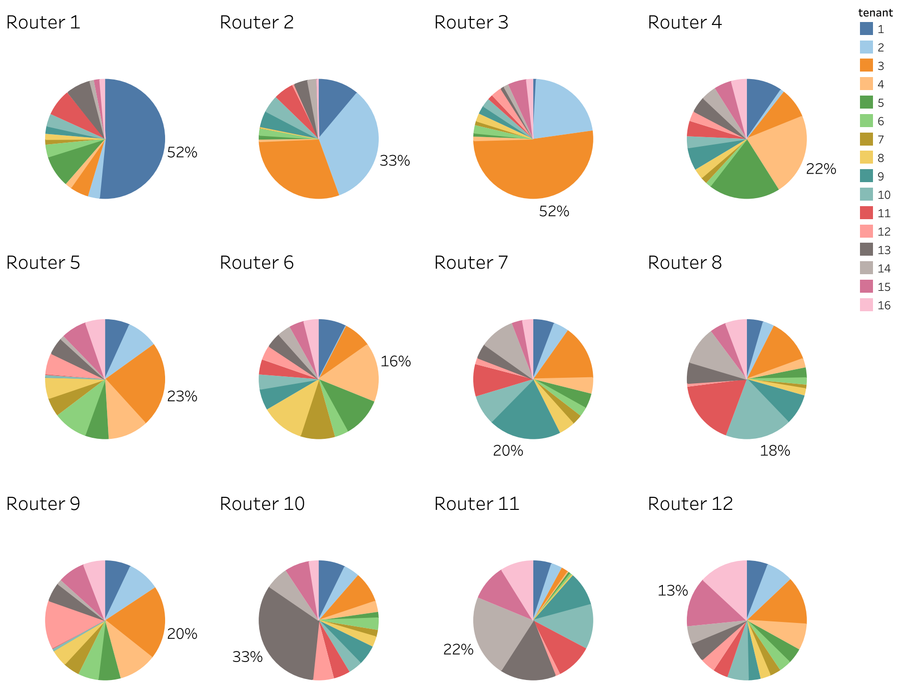

Fig. 9 Dashboard showing the traffic contribution of each tenant per router

from router_12. router_12 is one of two possible paths to all other tenants used to reach tenants 15 and 16, and thus a higher *pps* in this router is observed (see Fig. 4). Tenant 3 was identified as a heavy hitter, and thus it also has a high *pps*.

As an assumption when exemplifying the power of BitMatrix, it is possible to see from Fig. 4 that both tenants 15 and 16 are directly connected to router_12, and thus the unexpected traffic load is to/from tenant_3. Let us analyze the traffic per tenant in the network to better understand the traffic profile of each tenant. Figure 8 shows the total traffic, per tenant, in the network.

From Fig. 8, it can be seen that the higher traffic in the network is to/from tenant_3, which can be identified as the biggest offender in the network. Figure 9 presents a dashboard with a complete view representing the total traffic contribution, per tenant, on each router. For an efficient visualization, the percentage is only presented for the tenant with the highest contribution.

Several other analyses can be applied. Using the simulator, it was possible to obtain insight and better understand the power when using the packet digested (sketch) information from BitMatrix. It is important to highlight that, because the information is collected every 10 s during this evaluation, a near real-time perspective regarding the network behavior and performance analysis is offered to the network administrator.

However, because BitMatrix has a probabilistic data structure, hash collisions may introduce errors in the measurements conducted when using the data collected from the sketches. In the next section (Sect. 4.3), we explore statistics regarding

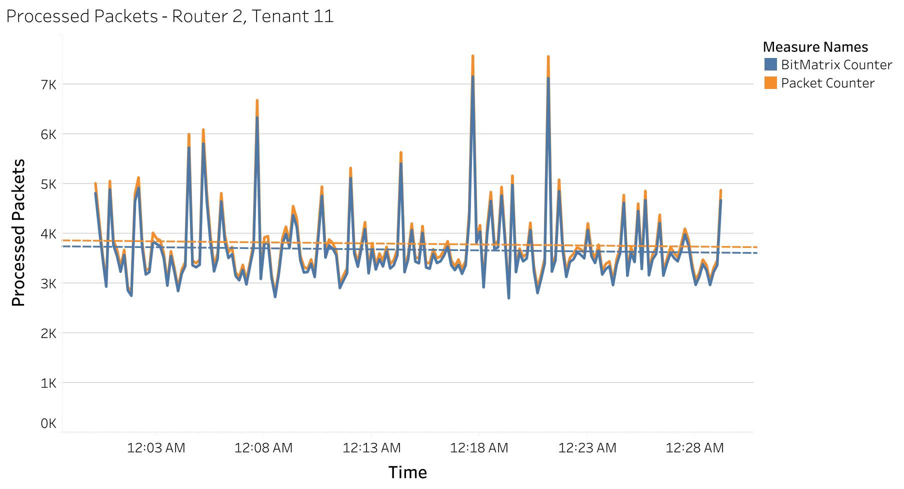

Fig. 10 Number of packets measured by BitMatrix and the packet counter for router_2 and tenant_11. The monitoring time frame is set to 10 s

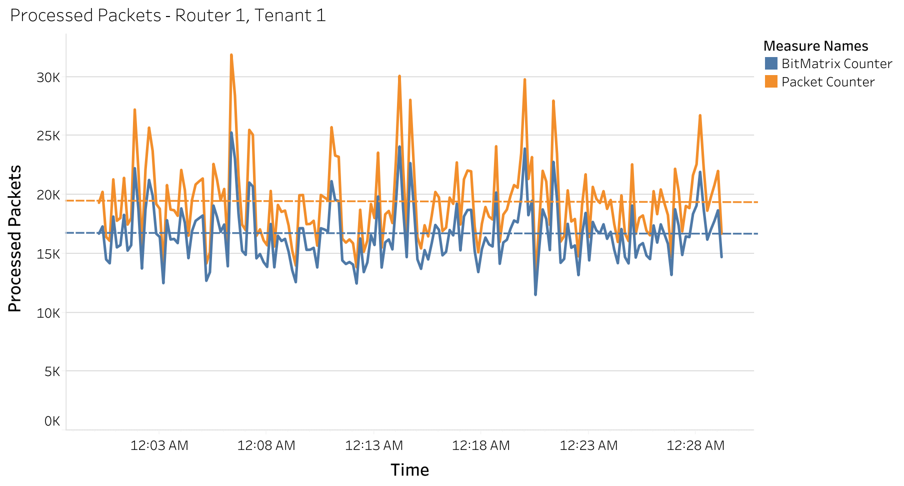

Fig. 11 Number of packets measured by BitMatrix and the packet counter for router_1 and tenant_1. The monitoring time frame is set to 10 s

collisions and the bitmap occupation, and propose a model based on a polynomial regression to adjust the results compensating the measurement errors from hash collisions.

4.3 BitMatrix Measurement Evaluation

The simulator generates one BitMatrix containing 16 bitmaps and 16 counter arrays (one per tenant), per router, at every 10 s or for every 430,000 packets

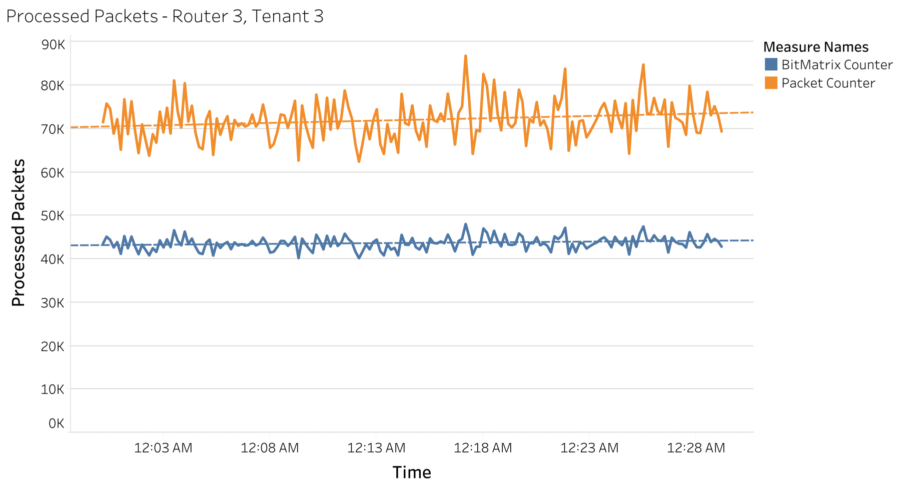

Fig. 12 Number of packets measured by BitMatrix and the packet counter for router_3 and tenant_3. The monitoring time frame is set to 10 s

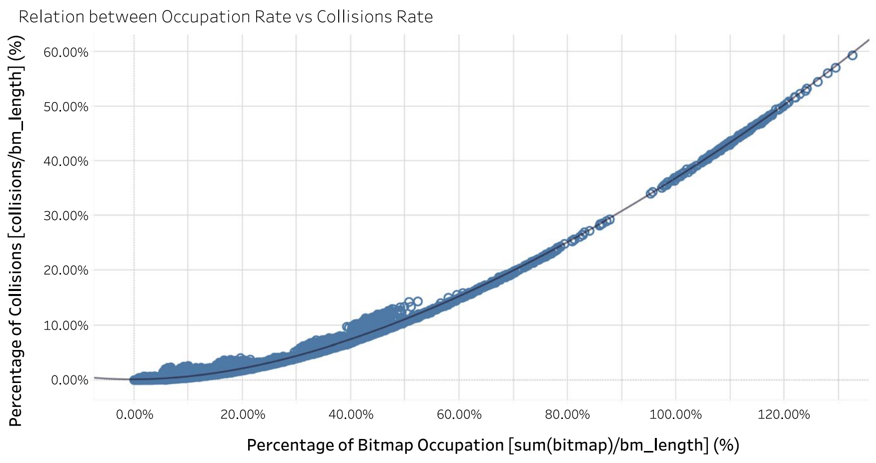

Fig. 13 Percentage of bitmap occupation versus the percentage of collisions, per bitmap

processed on average. In total, after processing the packets to simulate 30 min of traffic, it creates approximately 34,500 bitmaps and counter arrays. In addition to generating BitMatrices, the simulator was also used to create real packets and byte counters to evaluate the quality of the information provided by BitMatrix. When comparing the measurements from the bitmaps and these real counters, a gap between the two can be seen.

Figures 10, 11, and 12 show the difference between the measurements based on BitMatrix and the packet/byte counter implemented in the simulator. The

Table 3 Database partitioning for k-fold cross validation

Partition #	Training set	Test set
Partition 1	[Index] < 26881	[Index] > 26880
Partition 2	[Index] > 6720	[Index] < 6721
Partition 3	[Index] < 6721 OR [Index] > 13440	[Index] > 6720 AND [Index] < 13441
Partition 4	[Index] < 13441 OR [Index] > 20160	[Index] > 13440 AND [Index] < 20161
Partition 5	[Index] < 20161 OR [Index] > 26880	[Index] > 20160 AND [Index] < 26881

packet counter reports the real number of packets/bytes processed, and BitMatrix estimates its number by using data from the sketches.

From these figures, it can be seen that the measurement error of BitMatrix increases as more packets are processed. The difference shown in Fig. 11 is greater than that in Fig. 10; in addition, the difference in Fig. 12 is greater than that in Fig. 11. This occurs as a result of the increase in the hash collisions as more packets are processed, and because the size of BitMatrix is the same in all routers. Thus, the chances of a collision increase as BitMatrix becomes more occupied.

Aiming to find an algorithm based on the historical data to apply an adjustment to the BitMatrix measurements, approximating it to real values, the relation between the BitMatrix occupation and the percentage of hash collisions was used to train and test the ML model. Figure 13 shows the relation between these two measurements.

To estimate the occupation of BitMatrix and the percentage of hash collisions, the size of the bitmap in BitMatrix was used as a reference in Eqs. 12 and 13, respectively. Here, n is the size of the bitmap, in bits. For the sake of simplification, in Eq. 12 the index of the tenant is represented as "any," [], i.e., the bitmaps are from the same tenant. As stated in this section, BitMatrix was set to 65,536 bits and consequently the bitmaps in BitMatrix have the same size. The representation shown in Fig. 13 also uses this approach. During the simulation, if the simulator counts 2000 collisions when storing the packets for the same tenant in BitMatrix, the percentage of collisions will be 2000 divided by 65,536, which is 3.051758%.

$$\%\text{ of occupation} = \left(\sum_{i=1}^{n} BitMatrix[][i] \right) / n \tag{12}$$

$$\%\text{ of collisions} = \text{total number of collisions}/n \tag{13}$$

In a more realistic scenario, it is not desirable to have one more counter in the data plane to count the number of hash collisions. Thus, the idea is to use the bitmap occupation rate and estimate the number of collisions. This idea was used to create an ML model, and using this prediction it is possible to adjust the BitMatrix measurements, which incurs a smaller error and better approximation into the real values.

Table 4 Average MSE and average standard deviation (StdDev) for the testing database

Method	Average mean squared error	Average std. deviation
Polynomial degree 4	1.76E−05	0.039634
Polynomial degree 3	1.84E−05	0.039674
Polynomial degree 2	3.66E−05	0.039381
Linear	4.95E−04	0.034805
Logarithmic	1.30E−03	0.025877
Exponential	5.96E−00	2.466112

4.4 Use of ML Algorithms to Predict the Number of Hash Collisions

To predict the hash collision rate in the network, linear and polynomial regression algorithms were evaluated. First, they were trained as a part of an ML technique. For the training phase, a bitmap database with 33,600 samples of hash collisions in a real network implemented in P4 was used. This database was divided into five partitions of 6720 samples each for the k-fold validation. In the k-fold validation, we used four partitions for training the algorithm and one partition for testing. Each field of the database has its own index, from 1 to 33,600. The index field was used for the field selection, as shown in Table 3.

Table 4 shows the results for the k-fold cross-validation process. The mean squared error (MSE) is the average for the five partitions, and may help in selecting the best polynomial algorithm for application to the prediction problem. The lower the error that occurs, the better the result. The table is sorted in ascending order, with the best results shown first.

As a result of the training and testing phases, Eq. 14 expresses the number of collisions as a function of the occupation (*occupation*), in percent (%). In this same equation, n is the bitmap size in BitMatrix. It is important to highlight that Eq. 14 applies to any other traffic dataset as well, the only requirement being that with any other ML technique, the algorithm must be retrained allowing new input values for the equation to be generated. We can even vary the size of the bitmaps (n) to have greater or less accuracy as desired.

$$\begin{aligned}\text{number of collisions} =\, & n * (0.0600287 * occupation^4) \\ & + (-0.226063 * occupation^3) + (0.531893 * occupation^2) \\ & + (0.0019715 * occupation^1) + 0.000701056)\end{aligned} \tag{14}$$

Using this result, the adjusted bitmap is used as a more accurate data source for the measurements when using BitMatrix. Equation 15 expresses the estimated number of packets in the adjusted bitmap as a function of the predicted number of collisions. Here, n is the size of the bitmap, in bits.

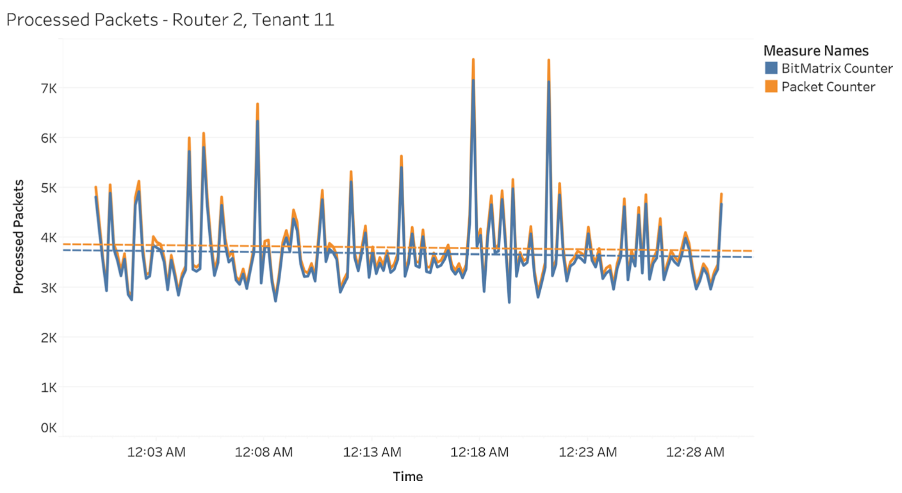

Fig. 14 Number of packets processed as measured by BitMatrix counter, packet counter, and BitMatrix adjusted for router_2 and tenant_11. The monitoring time frame is set to 10 s

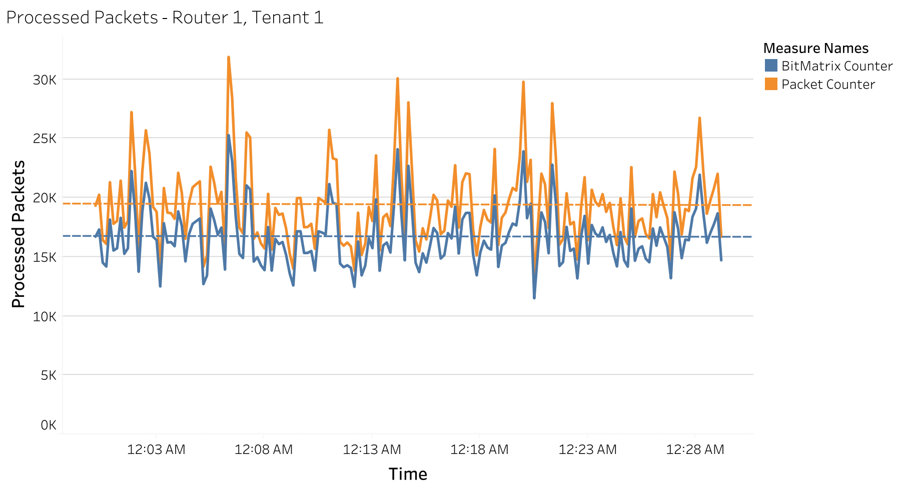

Fig. 15 Number of packets processed as measured by BitMatrix counter, packet counter, and BitMatrix adjusted for router_1 and tenant_1. The monitoring time frame is set to 10 s

$$\begin{aligned}&\text{number of packets in the adjusted bitmap}\\&\quad=\text{number of collisions}+\sum_{i=1}^{n} bitmap[i]\end{aligned}\tag{15}$$

Figures 14, 15, and 16 show three values for comparison:

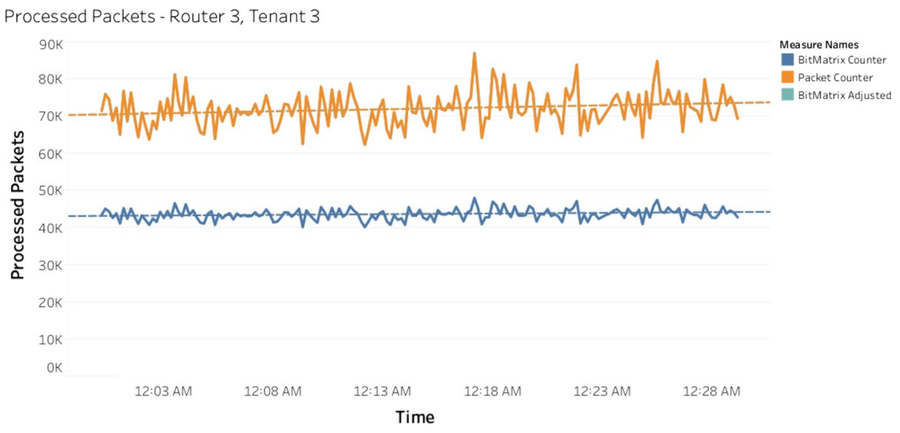

Fig. 16 Number of packets processed as measured by BitMatrix counter, packet counter, and BitMatrix adjusted for router_3 and tenant_3. The monitoring time frame is set to 10 s

1. BitMatrix counter: This is the result of the counted packets by summarizing the number of bits in the original bitmap.
2. Packet counter: This is the real number of packets counted by a counter in the simulator and is used as a reference.
3. BitMatrix adjusted: This is the result of the packets counted by summarizing the number of bits in the bitmap, and adjusting this value by using the polynomial regression algorithm based on the bitmap occupation. This line (in green) in the graph is overlapped by the packet counter line (in orange) and is not well perceptible.

The adjusted BitMatrix measurement has a mean absolute percentage error of $\pm 6.14\%$. It is also possible to observe that, even under a low or high occupation, the performance of the algorithm does not decrease.

5 Conclusion and Future Studies

Network monitoring is a crucial task for a network operator. Information provided by the monitoring tools offers intelligence for the decision-making process for capacity planning and traffic engineering. The solution proposed in this paper, a BitMatrix sketch, goes further than a traditional monitoring process. In addition to general statistics, it enables detailed analyses to be segmented per tenant. The solution presented in this paper differs from previous solutions in the sense that several data analyses can be conducted in the control plane by means of the sketches collected through BitMatrix. We found that the counters collected are not useful if not properly observed through a proper correlation of data, which must be applied in the control plane. This hidden information cannot be obtained by looking at raw counters.

To process real traffic captured in a simulated network, we used CAIDA traces to produce a considerable number of traffic statistics for routers, broken down by a tenant. Moreover, another key contribution of this study is that using the generated statistics from this simulation, we created an algorithm, using supervised ML, to reduce the errors introduced by hash collisions.

The model presented in this paper is a powerful mechanism to minimize the measurement errors caused by hash collisions, and may be applied to other scenarios, not only those related to packet counting. In this study, linear and polynomial regression algorithms are used; however, other algorithms may be trained and tested using the same traces.

The development and evaluation of BitMatrix can be improved in many ways. A more user-friendly interface is required for creating rules to generate specific information from data stored in BitMatrix. Another task will be to speed up the statistics module, using an extract, transform, load tool to retrieve data periodically from BitMatrix, and aggregate the metrics (number of packets and bytes) based on the network device and tenant dimensions. We also believe that the BitMatrix structure is a first step toward the segmented monitoring of slices, which has gained significant attention in recent studies related to 5G, cloud computing, and IoT verticals.

Acknowledgements The authors would like to thank CAPES, CNPq, FAPES, NECOS and FAPESP for partially supporting this research.

References

1. CISCO: 2020 global networking trends report. Tech. rep., CISCO (2019). https://engage2demand.cisco.com/LP=18332?ccid=cc001244&oid=rpten018612. Accessed 6 Jan 2020
2. Dimitropoulos, X., Hurley, P., Kind, A.: Probabilistic lossy counting: an efficient algorithm for finding heavy hitters. Comput. Commun. Rev. **38**, 5 (2008)
3. Moshref, M., Yu, M., Govindan, R., Vahdat, A.: Scream: sketch resource allocation for software-defined measurement. In: Proceedings of the 11th ACM conference on emerging networking experiments and technologies, CoNEXT ’15, pp. 14:1–14:13. ACM, Heidelberg (2015). https://doi.org/10.1145/2716281.2836099
4. Moshref, M., Yu, M.Y., Govindan, R., Vahdat, A.: DREAM: Dynamic resource allocation for software-defined measurement . Proceedings of the 2014 ACM SIGCOMM conference (2014)
5. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with opensketch. In: Proceedings of the 10th USENIX conference on networked systems design and implementation, NSDI’13, pp. 29–42. USENIX Association, Lombard (2013). http://dl.acm.org/citation.cfm?id=2482626.2482631
6. Claise, B.: Cisco systems NetFlow services export version 9. RFC 3954, Cisco Systems (2004). https://tools.ietf.org/html/rfc3954. Accessed 6 Jan 2020
7. sflow-rt (2019). https://sflow-rt.com/. Accessed 9 Aug 2019
8. Gibbons, P.B., Matias, Y.: New sampling-based summary statistics for improving approximate query answers. ACM SIGMOD Rec. (1999). https://doi.org/10.1145/276304.276334
9. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Möhring, R., Raman, R. (eds.) Algorithms—ESA 2002, pp. 348–360. Springer, Berlin (2002)
10. Kamiyama, N., Mori, T.: Simple and accurate identification of high-rate flows by packet sampling. In: Proceedings IEEE INFOCOM 2006. In: 25TH IEEE international conference on computer communications, pp. 1–13 (2006). https://doi.org/10.1109/INFOCOM.2006.324
11. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2003). https://doi.org/10.1145/872757.872764

12. Zhao, Q.G., Kumar, A., Wang, J., Xu, J.J.: Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. In: Proceedings of the 2005 ACM sigmetrics international conference on measurement and modeling of computer systems, SIGMETRICS '05, pp. 350–361. ACM, Banff (2005). https://doi.org/10.1145/1064212.1064258
13. Bandi, N., Metwally, A., Agrawal, D., El Abbadi, A.: Fast data stream algorithms using associative memories. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data, SIGMOD '07, pp. 247–256. ACM, Beijing (2007). https://doi.org/10.1145/1247480.1247510
14. Mathew, R., Katkar, V.: Survey of low rate dos attack detection mechanisms. In: Proceedings of the international conference & 38; workshop on emerging trends in technology, ICWET '11, pp. 955–958. ACM, Mumbai (2011). https://doi.org/10.1145/1980022.1980227
15. Krishnamurthy, B., Sen, S., Zhang, Y., Chen, Y.: Sketch-based change detection: methods, evaluation, and applications. In: Proceedings of the 3rd ACM SIGCOMM conference on internet measurement, IMC '03, pp. 234–247. ACM, Miami Beach (2003). https://doi.org/10.1145/948205.948236
16. Schweller, R., Gupta, A., Parsons, E., Chen, Y.: Reversible sketches for efficient and accurate change detection over network data streams. In: Proceedings of the 4th ACM SIGCOMM conference on internet measurement, IMC '04, pp. 207–212. ACM, Taormina (2004). https://doi.org/10.1145/1028788.1028814
17. Duffield, N., Lund, C., Thorup, M.: Estimating flow distributions from sampled flow statistics. IEEE/ACM Trans. Netw. **13**(5), 933–946 (2005). https://doi.org/10.1109/TNET.2005.852874
18. Kumar, A., Sung, M., Xu, J.J., Wang, J.: Data streaming algorithms for efficient and accurate estimation of flow size distribution. SIGMETRICS Perform. Eval. Rev. **32**(1), 177–188 (2004). https://doi.org/10.1145/1012888.1005709
19. Guanyao Huang, Lall, A., Chuah, C., Jun Xu: Uncovering global icebergs in distributed monitors. In: 2009 17th international workshop on quality of service, pp. 1–9 (2009)
20. Sanjuàs-Cuxart, J., Barlet-Ros, P., Duffield, N., Kompella, R.: Sketching the delay: tracking temporally uncorrelated flow-level latencies. Proceedings of the ACM SIGCOMM internet measurement conference, IMC (2011). https://doi.org/10.1145/2068816.2068861
21. Zhang, Y., Singh, S., Sen, S., Duffield, N., Lund, C.: Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In: Proceedings of the 4th ACM SIGCOMM conference on internet measurement, IMC '04, p. 101–114. Association for Computing Machinery, New York (2004). https://doi.org/10.1145/1028788.1028802
22. Li, X., Bian, F., Crovella, M., Diot, C., Govindan, R., Iannaccone, G., Lakhina, A.: Detection and identification of network anomalies using sketch subspaces. In: Proceedings of the 6th ACM sigcomm conference on internet measurement, IMC '06, p. 147–152. Association for Computing Machinery, New York (2006). https://doi.org/10.1145/1177080.1177099
23. Huang, Q., Lee, P.P.: A hybrid local and distributed sketching design for accurate and scalable heavy key detection in network data streams. Comput. Netw. **91**(C), 298–315 (2015). https://doi.org/10.1016/j.comnet.2015.08.025
24. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005). https://doi.org/10.1016/j.jalgor.2003.12.001
25. Estan, C., Varghese, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Trans. Comput. Syst. **21**, 270–313 (2003)
26. Mitzenmacher, M., Pagh, R., Pham, N.: Efficient estimation for high similarities using odd sketches. In: Proceedings of the 23rd international conference on world wide web, WWW '14, pp. 109–118. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2566486.2568017
27. Vieira, M.A.M., Castanho, M.S., Pacífico, R.D.G., Santos, E.R.S., Júnior, E.P.M.C., Vieira, L.F.M.: Fast packet processing with ebpf and xdp: concepts, code, challenges, and applications. ACM Comput. Surv. (2020). https://doi.org/10.1145/3371038
28. Pacífico, R.D.G., Silva, L.B., Coelho, G.R., Silva, P.G., Vieira, A.B., Vieira, M.A.M., Ítalo, F.S.C., Vieira, L.F.M., Nacif, J.A.M.: Bloomtime: space-efficient stateful tracking of time-dependent network performance metrics. Telecommun. Syst. (2020). https://doi.org/10.1007/s11235-020-00653-1
29. Li, Y., Miao, R., Kim, C., Yu, M.: Flowradar: a better netflow for data centers. In: 13th USENIX symposium on networked systems design and implementation (NSDI 16), pp. 311–324. USENIX Association, Santa Clara (2016)
30. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., Walker, D.: P4: programming protocol-independent packet processors. SIGCOMM Comput. Commun. Rev. **44**(3), 87–95 (2014). https://doi.org/10.1145/2656877.2656890

31. Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., Wobker, L.J.: In-band network telemetry via programmable dataplanes. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15. ACM, Santa Clara (2015)
32. Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., Rexford, J.: Heavy-hitter detection entirely in the data plane. In: Proceedings of the symposium on SDN research, SOSR '17, pp. 164–176. ACM, Santa Clara (2017). https://doi.org/10.1145/3050220.3063772
33. Martins, R., Garcia, L.F., Villaça, R., Verdi, F.L.: Using probabilistic data structures for monitoring of multi-tenant p4-based networks. In: Proceedings of the IEEE symposium on computers and communications, ICC '18. IEEE (2018). https://doi.org/10.1109/ISCC.2018.8538352
34. Zhang, Y.: An adaptive flow counting method for anomaly detection in sdn. In: Proceedings of the 9th ACM conference on emerging networking experiments and technologies, CoNEXT '13, pp. 25–30. ACM, Santa Barbara (2013). https://doi.org/10.1145/2535372.2535411
35. Xie, Y., Sekar, V., Maltz, D.A., Reiter, M.K., Zhang, H.: Worm origin identification using random moonwalks. In: 2005 IEEE symposium on security and privacy (S P'05), pp. 242–256. IEEE, Oakland (2005). https://doi.org/10.1109/SP.2005.23
36. Benson, T., Anand, A., Akella, A., Zhang, M.: Microte: fine grained traffic engineering for data centers. In: Proceedings of the seventh conference on emerging networking experiments and technologies, CoNEXT '11, pp. 8:1–8:12. ACM, Tokyo (2011). https://doi.org/10.1145/2079296.2079304
37. Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., True, F.: Deriving traffic demands for operational ip networks: methodology and experience. IEEE/ACM Trans. Netw. **9**(3), 265–280 (2001). https://doi.org/10.1109/90.929850
38. Wang, N., Ho, K., Pavlou, G., Howarth, M.: An overview of routing optimization for internet traffic engineering. Commun. Surveys Tuts. **10**(1), 36–56 (2008). https://doi.org/10.1109/COMST.2008.4483669
39. Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., Rexford, J.: Heavy-hitter detection entirely in the data plane. In: Proceedings of the symposium on SDN research, SOSR '17, p. 164–176. Association for computing machinery, New York, NY, USA (2017). https://doi.org/10.1145/3050220.3063772
40. Kim, J., Sim, A.: A new approach to multivariate network traffic analysis. J. Comput. Sci. Technol. **34**(2), 388–402 (2019). https://doi.org/10.1007/s11390-019-1915-y
41. Phaal, P., Panchen, A.S., McKee, N.: InMon corporation's sFlow: a method for monitoring traffic in switched and routed networks. RFC 3176, internet engineering task force (IETF) (2001). https://tools.ietf.org/html/rfc3176
42. Estan, C., Varghese, G.: New directions in traffic measurement and accounting. In: Proceedings of the 1st ACM SIGCOMM workshop on Internet Measurement, IMW '01, pp. 75–80. ACM, San Francisco (2001). https://doi.org/10.1145/505202.505212
43. Ramachandran, A., Seetharaman, S., Feamster, N., Vazirani, V.: Fast monitoring of traffic subpopulations. In: Proceedings of the 8th ACM SIGCOMM conference on internet measurement, IMC '08, pp. 257–270. ACM, Vouliagmeni (2008). https://doi.org/10.1145/1452520.1452551
44. Braverman, V., Liu, Z., Singh, T., Vinodchandran, N.V., Yang, L.F.: New bounds for the CLIQUE-GAP problem using graph decomposition theory. In: Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24–28, 2015, Proceedings, Part II, pp. 151–162 (2015)
45. Lall, A., Sekar, V., Ogihara, M., Xu, J., Zhang, H.: Data streaming algorithms for estimating entropy of network traffic. SIGMETRICS Perform. Eval. Rev. **34**(1), 145–156 (2006). https://doi.org/10.1145/1140103.1140295
46. Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., Braverman, V.: One sketch to rule them all: Rethinking network flow monitoring with univmon. In: Proceedings of the 2016 ACM SIGCOMM conference, SIGCOMM '16, pp. 101–114. ACM, Florianopolis (2016). https://doi.org/10.1145/2934872.2934906
47. Wellem, T., Lai, Y., Huang, C., Chung, W.: A flexible sketch-based network traffic monitoring infrastructure. IEEE Access **7**, 92476–92498 (2019)
48. Huang, Q., Jin, X., Lee, P.P.C., Li, R., Tang, L., Chen, Y.C., Zhang, G.: Sketchvisor: Robust network measurement for software packet processing. In: Proceedings of the conference of the ACM special interest group on data communication, SIGCOMM '17, pp. 113–126. ACM, Los Angeles (2017). https://doi.org/10.1145/3098822.3098831

49. Shahbaz, M., Choi, S., Pfaff, B., Kim, C., Feamster, N., McKeown, N., Rexford, J.: Pisces: A programmable, protocol-independent software switch. In: Proceedings of the 2016 ACM SIGCOMM conference, SIGCOMM '16, pp. 525–538. ACM, Florianopolis (2016). https://doi.org/10.1145/2934872.2934886
50. Dang, H.T., Canini, M., Pedone, F., Soulé, R.: Paxos made switch-y. SIGCOMM Comput. Commun. Rev. **46**(2), 18–24 (2016). https://doi.org/10.1145/2935634.2935638
51. Sivaraman, A., Kim, C., Krishnamoorthy, R., Dixit, A., Budiu, M.: Dc.p4: Programming the forwarding plane of a data-center switch. In: Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research, SOSR '15, pp. 2:1–2:8. ACM, Santa Clara (2015). https://doi.org/10.1145/2774993.2775007
52. Snoeren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Kent, S.T., Strayer, W.T.: Hash-based ip traceback. SIGCOMM Comput. Commun. Rev. **31**(4), 3–14 (2001). https://doi.org/10.1145/964723.383060
53. NETRONOME: Netronome Agilio SmartNIC. https://www.netronome.com/products/agilio-cx/ (2020). Accessed 18 Mar 2020
54. Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., Uhlig, S.: Elastic sketch: adaptive and fast network-wide measurements. In: Proceedings of the 2018 ACM SIGCOMM conference, SIGCOMM '18. ACM (2018)
55. Tableau: Tableau Software. https://www.tableau.com/ (2020). Accessed 02 May 2020
56. Martins, R.: Packet routing analyses using probabilistic data structures in Multi-Tenant Networks based on programmable devices. Master Thesis. Federal University of São Carlos, UFSCar (2018). https://repositorio.ufscar.br/handle/ufscar/11892

Regis Francisco Teles Martins is graduated in Electrical Engineering from Universidade Santa Úrsula (2008) and Master in Computer Science from Universidade Federal de São Carlos—Campus Sorocaba (2019). He is currently a pre-sales engineer at Sandvine Incorporated ULC and has experience in the field of Electrical Engineering, with an emphasis on Telecommunications Systems.

Rodolfo da Silva Villaça is an assistant professor at the Industrial Technology Department (DTI) of the Federal University of Espirito Santo (Ufes). He received his Ph.D. in Computer Engineering in 2013 at the University of Campinas (Unicamp). He also holds a Computer Engineering degree and a M.Sc. in Electrical Engineering, both from the Federal University of Espirito Santo (Ufes). His main research Interests are in the Computing Systems area. He was the Coordinator of the Point of Presence of the Brazilian National Education and Research Network (RNP) in the state of Espírito Santo (PoP-ES) from August/2016 to February/2020.

Fábio L. Verdi is an Associate Professor at the Computing Department in Federal University of São Carlos (UFSCar) campus Sorocaba. He received his Master degree in Computer Science and Ph.D. degree in Electrical Engineering both from State University of Campinas (UNICAMP). Fábio has been working with data centers, cloud computing and SDN. He is the coordinator of the LERIS Research Group and has been leading projects in the area of monitoring of virtual resources and cloud infrastructures. Currently, he is member of the NECOS Project (BR-EU joint call) and FIXP Project (MCTIC/FAPESP).