Using Probabilistic Data Structures for Monitoring of Multi-tenant P4-based Networks

Regis F. T. Martins, Fábio L. Verdi
Computing Department
Federal University of São Carlos
Sorocaba-SP, Brazil
regisftm@gmail.com,fverdi@gmail.com

Rodolfo Villaça, Luis Fernando U. Garcia
Department of Industrial Technology
Federal University of Espírito Santo
Vitória-ES, Brazil
rodolfo.villaca@ufes.br,luferu@gmail.com

***Abstract*—With the evolution of the processing capacity of the network devices and the appearing of novel programmable network hardware, new methods and techniques for traffic monitoring have been proposed. Among these new techniques, the use of sketches has been increased. Sketches are probabilistic data structures capable of summarizing information about the network with two main advantages over flow-based counters: low memory usage and adjustable accuracy. Considering this scenario, this paper proposes the use of probabilistic data structures implemented in P4 (Programming Protocol-Independent Packet Processors) to monitor multi-tenant networks. While it is common to see similar solutions for monitoring non-shared networks, as far as we know this is the first work that deals multi-tenant scenarios. We implemented sketches for each tenant so that an isolated network monitoring can be done. The solution, named BitMatrix, was created taking into account two probabilistic structures: bitmaps and counter-arrays.**

***Index Terms*—probabilistic structures, counter-array, bitmaps, P4, monitoring.**

I. Introduction

The exponential growth of network traffic emphasizes the importance of network monitoring, management, planning and traffic engineering. It is important for data centers to have accurate and fine-grained monitoring to operate efficiently, considering the use of shared resources by multiple tenants.

In this context, sketches are probabilistic data structures used to store summarized information about the network. There are several applications in measurement tasks for sketches, such as heavy-hitter detection, traffic pattern change detection, traffic matrix estimation, flow-size distribution estimation, and others [1].

As an evolution of Software Defined Networking (SDN), P4 (Programming Protocol-Independent Packet Processor) was presented as a high-level programming language for packet forwarding devices [2]. In order to use a program written in P4, it has to be compiled to the target device, which can be a hardware or software based system. P4 allows the creation of several mechanisms for traffic measurement [3] [4], among them, the probabilistic data structures, or sketches.

The main goal of this paper is to propose and implement a solution based on the P4 language for multi-tenant network monitoring, based on the well-known bitmap and counter-array sketches. A new compact and probabilistic structure, named BitMatrix, is presented. BitMatrix was created to support the multi-tenant monitoring.

The rest of this paper is organized as follows: Section II shortly discusses important concepts about well-known probabilistic compact structures. Section III presents BitMatrix and how it is used for multi-tenant network monitoring. Section IV shows the usage of our solution for collecting network information taking into account a shared multi-tenant scenario. Section V concludes the paper and some future work are proposed.

II. Background

It is well known that network monitoring is a widely studied topic in the literature, presenting several solutions. Sketches [5] allow to reduce the computational cost associated with the detailed collecting of information in the network [6] and can get traffic statistics requiring a fixed size memory with controlled accuracy. In this paper the measurement strategy is based on a probabilistic approach for packet counting as defined by Zhao [6]. In particular, we use a probabilistic structure called BitMatrix, which is based on bitmaps and counter-arrays.

The bitmap data structure is simple: an array of bits initialized with zeros on each monitoring node. For each packet arrival at a participating node, the node simply sets the bit to 1, indexed by the hash value of the header's invariant fields plus some bytes of the payload of this packet [6]. That is, the value resulted from the hash operation will be used to select which position of the bitmap will be set as 1 and will represent each packet in a unique way. Even by using different values from packet fields and payload for hash calculation, it is possible that two hashes with different parameters results in the same value. This is called hash collision and creates a gap between the amount of packets actually forwarded by the element and the total number of positions marked in the bitmap. There are three main factors that can cause variation in the number of hash collision in the proposed scenario: a) the size of the bitmap or, in other words, the number of positions available in the array; b) the occupation of the bitmap when marking a new position; c) the type of the hash funtion used to generate the hash value.

Bitmaps can be used to estimate the volume of traffic between two observing network devices in any period of time during the monitoring activity. To do this, it is required to retrieve the set of bitmaps marked during this interval and then compare the two sets of bitmaps. The more common bits in the same positions the more common packets have passed through these elements. If the intersection of bits in a same position is small, it is possible to conclude that few common packets have passed by these elements.

The size of the bitmap and the size of hash value are related, in a sense that there is no point in setting a bitmap with more positions than the maximiun 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 resulted hash value in order to determine the offset for the present packet. We also need to consider the bitmap occupancy as the more occupied is a bitmap, bigger are the chances of a hash collision.

In addition to the bitmap probabilistic structure, counter-arrays can be used to count the number of bytes of each arrival in a network device. This sketch is basically an array of counters and is operation and counting is very similar to bitmap, but it counts, not only marks a position in the array. In the next section, we present the use of our proposed data structure, called BitMatrix, and how it is associated with bitmaps and counter-arrays in order to estimate the amount of packets and bytes that passes through a node, per tenant and, in addition, understand the path taken for those packets inside the P4 network.

III. P4 BitMatrix Design and Implementation

The main idea is to implement a bitmap using available commands and structures in the P4 language. To do that, the bitmap is implemented as a P4 register. Its length is the register's parameter *instance_count* and the register's parameter *width* is set to 1, as it will receive only values 0 and 1. The other possibility, regarding the usage of bitmap in P4, is that it is possible to create multiple arrays using only one register with a larger width. This will result in what we called BitMatrix. The BitMatrix is the usage of a unique P4 register to host several bit arrays, each one used to measure traffic from a different tenant identified by its Source IP subnetwork.

The goal is to use BitMatrix associated with counter-arrays to estimate the amount of packets and bytes transmitted for each tenant and, in addiction, understand the path taken by those packets inside the P4 network. Each packet received by the P4 switch computes a hash value. This value is used to determine which position will be set in the BitMatrix. In this paper we used BitMatrix to count packets according to its origin (tenant). In this way, it is possible to determine which tenant is responsible for each packet in the network.

Figure 1 illustrates BitMatrix structure and how each bitmap is accommodated into it. In this paper, as a proof of concept, a BitMatrix composed by three bitmaps is presented. This resulted in a P4 register with a width equal to 3. Thus, it is possible to segment traffic from up to three different tenants, setting different bitmaps inside the BitMatrix according to which tenant originated the packet. Using a P4 table, a value for each tenant can be assigned, according to its Source IP subnetwork: 1 to tenant A, 2 to tenant B and 4 to tenant C. Once the hash value of each packet is computed, a modulo operation is applied to the value to determine what position should be set in the BitMatrix. This is achieved by using the P4 primitive action *modify_field_with_hash_based_offset* [7]. As each position has three bits, we used another P4 primitive action named *bit_or* to set the correct bit in that position of BitMatrix by performing a logical OR operation using the current value for the selected position and the tenant value.

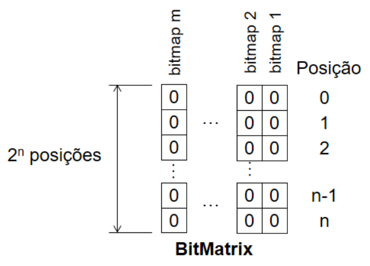

Figure 1. BitMatrix structure and its bitmaps.

To count bytes it was not possible to use the same P4 structure for all tenants, so we created one P4 register to work as a counter-array for each tenant. Finding the position for where to sum the current packet bytes is achieved by using the same hash value computed for BitMatrix. In this way, when retrieving packets, it is possible to know how many bytes were carried by that packet.

IV. Evaluation

Towards to determine what is the more efficient setup for the bitmap size and maximum occupancy, tests were done using a fixed hash value size and varying the bit array size and the percentage of occupancy. The hash algorithm used was the checksum 16 (csum16), which generates a value of 16 bits length. The bit array sizes tested were 2048, 4096, 8192, 16384, 32768 and 65536 bits length. We did not control the occupation itself, instead, we processed an amount of packets approximately 5%, 10%, 25%, 50% and 100% of the bit array size. The output was the occupation smaller than the amount of packets processed due the hash collision inherent in the process.

To continue the BitMatrix evaluation the percentage of hash collision was target to keep under 10%, which implied a bitmap occupation around 15%. The hash collision was calculated by dividing the total number of positions marked in the bit array by the number of processed packets. This approach resulted in an epoch of 60s, a bandwidth limited to 1Mbps and bitmap size of 16384 positions. The setup for this experiment was constructed using Mininet [8] network emulator customized in order to enable P4 switch in the emulated network.

The topology used was composed of three hosts and four P4 switches. Each of the hosts received an IP address from a different subnetwork, emulating different tenants. The topology is presented in Fig. 2.

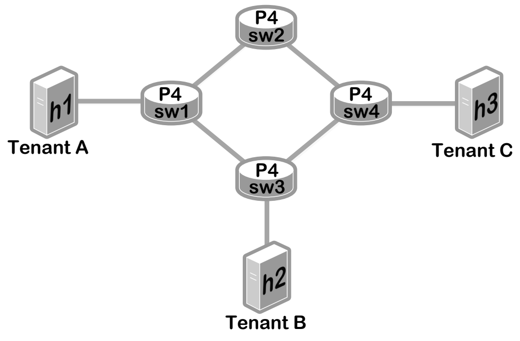

Figure 2. Mininet emulated network topology with P4-enabled forwarding.

The paths between tenants were arbitrarily defined as shown in Table I.

Table I
TRAFFIC PATH BETWEEN TENANTS

Tenant Pairs	Sw hop by hop	Tenant Pairs	Sw hop by hop
A to B	sw1 - sw3	C to A	sw4 - sw2 - sw1
B to A	sw3 - sw1	B to C	sw3 - sw4
A to C	sw1 - sw2 - sw4	C to B	sw4 - sw3

Every packet processed by the P4 switches generates entries in its corresponding probabilistic data structures (instantiated in each switch). Our P4 implementation consists in processing packets for the BitMatrix and can be described in the following general steps:

- Completely parse the packet headers of Layer 2, 3, 4 and the first 8 bytes of the payload;
- Select the packet headers to be used in the hashing algorithm;
- Determine the position in the bitmap;
- Determine the position (using the same hashing value) in the counter-array to sum the total bytes of the current packet with the previous ones;
- Forward the packet to the next hop.

A. *Parsing the Packet Header*

The P4 [2] implementation used in this work enabled the P4 switch to completely parse the packet headers from layer 2 to layer 4 and also the first 8 bytes (64 bits) of the payload. We implemented the parser for TCP, UDP and ICMP protocols. As the TCP layer usually brings optional headers, we used a variable length header to accommodate it. Ignoring TCP optional header would mislead the parsing of the payload. The parsing of payload was done by creating an one field header to receive the 8 bytes (64 bits) subsequent to the Layer 4 header.

B. *Hashing and Hash Inputs*

Although using the same hashing algorithms, the input for them must not vary and need also to be sufficient to identify a packet as unique across all hops in a network. Duffield and Gross-glauser [9] define that the IPv4 fields with low entropy are those which do not vary along the forwarding path for a given packet. Then, to have a low entropy, in this work we used the invariant IPv4 header fields (version, header length, total length, identification, flags, fragment offset, protocol, source address and destination address) and the first 8 bytes of the payload as input for the hash algorithm. According to Snoeren [10], those inputs are sufficient to differentiate unique packets.

C. *Retrieving and Processing BitMatrix*

To understand what was the path of a certain packet, it is necessary to compare the data structures gathered from different devices in the network. The data structures to be compared need to belong to the same epoch (monitoring interval). An epoch is the time frame in which the P4 switch stored information in the data structure. From time to time, the data structures need to be collected and reset. This determines the beginning of a new epoch.

The P4 switch is not in charge of collecting and storing the data structures. This task was performed by a centralized server, who collected the values from the P4 registers and reset them to start a new epoch. For this experiment, the time frame for each epoch was set to 60 seconds.

By processing data from bitmaps and counter-arrays, it is possible not only to obtain information regarding the amount of packets and bytes processed by each element per tenant, but also to identify how may packets and bytes per tenant went through a specific path in the network.

D. *Counting Results*

, The amount of packets, during a specific period, per tenant can be visualized in Figure 3. This metric was a result of the sum of all set positions in the bit array corresponding to each tenant in the BitMatrix in each epoch.

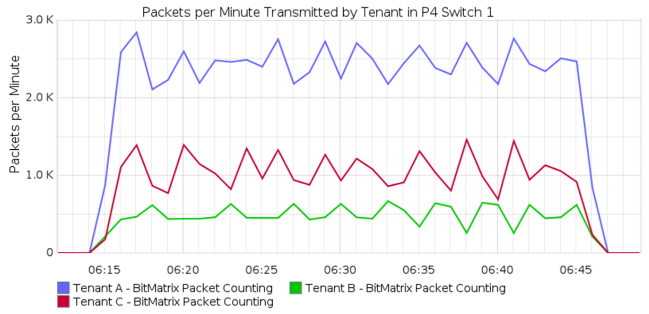

Figure 3. Amount of packets per tenant in P4 switch 1.

By counting the positions with bits set to 1 from all bitmaps of the BitMatrix, we obtain the approximated total number of packets processed by each P4 switch. This value reflects less packets than each P4 switch actually processed due to the hash collisions.

The amount of bytes sent by tenants was computed by counting the total values of each position from every tenant-correspondent counter-array. The total number of bytes related to packets forwarded by a particular P4 switch is computed by counting the values from each position in every tenant's counter-array. As counter-arrays indeed count the number of bytes, there is no hash collision and this result will reflect exactly the volume of bytes processed by the P4 switches.

By computing bitmaps from different network devices, it is possible to determine what was the path a packet took through the network. In Fig. 4, it is possible to see the amount of packets per minute originated by tenant A with destination to tenant B and packets originated from tenant B with destination to tenant A. Those metrics were calculated using logical operations with the bitmaps for tenants A and B from every P4 switch in charge of forwarding packets between those two tenants. Considering the information in Table I, it is possible to state that packets going from tenant A to tenant B will take the path passing through P4 switch 1 and then switch 3, and packets going from tenant B to tenant A will take the reverse route, passing firstly through P4 switch 3 and finally through P4 switch 1. With this information in hands, it is possible to determine which packets flowed from tenant A to tenant B. To do that, we used the logical expression ((sw1_A & sw3_A) & !sw4_A) where sw1_A is the bitmap corresponding to tenant A from P4 switch 1, sw3_A is the bitmap corresponding to tenant A from P4 switch 3, sw4_A is the bitmap corresponding to tenant A from P4 switch 4. Similar logic was used to determine which packets sent from tenant B went through P4 switch 3 and P4 switch 1, towards tenant A. This logic indicates what position was set by packets exchanged between tenants A and B.

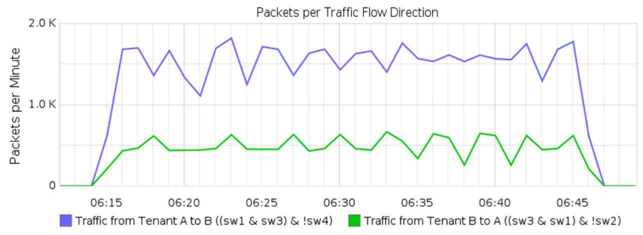

Figure 4. Amount of packets on path AB+BA.

Once these positions are known, we were able to count how many bytes were involved in the data transfer between tenants A and B.

V. Conclusion

In this paper we presented BitMatrix, a matrix of bitmaps used to measure traffic information in a multi-tenant network. BitMatrix information is complemented with the use of counter-arrays to register the amount of bytes carried by each packet in the network. The data structures were implemented using P4 language in the native P4 software switch BMv2 [11] in a network emulated with Mininet. Using an external server, acting as a collector. It was possible to retrieve information from BitMatrix of each P4 switch and store it to be processed producing relevant information according to the network administrator needs.

The amount of possibilities in terms of processing and mining the collected information is immense. Once the BitMatrix and counter-array data is obtained, it is just a matter of making logic operations, counting and crossing the bit streams to generate a huge amount of network information for each tenant and as a whole. Future work includes performance adjusts on the BitMatrix P4 implementation and the inclusion of new sketches to gather more information about the network in a compact way with adjustable accuracy.

Acknowledgment

Grant 2015/19766-9, São Paulo Research Foundation (FAPESP) and NECOS - Novel Enablers for Cloud Slicing - Project 777067 H2020 EU-Brazil Joint Call EUB-01-2017.

References

[1] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'13. Lombard, IL: USENIX Association, 2013, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482631

[2] 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, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[3] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band Network Telemetry via Programmable Dataplanes," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. Santa Clara, CA, USA: ACM, june 2015, Demo.

[4] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. Santa Clara, CA, USA: ACM, 2017, pp. 164–176. [Online]. Available: http://doi.acm.org/10.1145/3050220.3063772

[5] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Sep. 1985. [Online]. Available: http://dx.doi.org/10.1016/0022-0000(85)90041-8

[6] Q. G. Zhao, A. Kumar, J. Wang, and J. J. Xu, "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*, ser. SIGMETRICS '05. Banff, Alberta, Canada: ACM, 2005, pp. 350–361. [Online]. Available: http://doi.acm.org/10.1145/1064212.1064258

[7] "The P4 Language Specification - version 1.0.4," The P4 Language Consortium, https://p4.org, Specification, may 2017.

[8] Minitet project. [Online]. Available: http://mininet.org.

[9] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 280–292, Jun. 2001. [Online]. Available: http://dx.doi.org/10.1109/90.929851

[10] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based ip traceback," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 3–14, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/964723.383060

[11] The P4 Language Consortium. P4 Switch Behavioral Model. [Online]. Available: https: //github.com/p4lang/behavioral-model.