IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 September 2022, accepted 13 October 2022, date of publication 14 October 2022, date of current version 20 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3214974

RESEARCH ARTICLE

Reduction of the Delays Within an Intrusion Detection System (IDS) Based on Software Defined Networking (SDN)

ALESSANDRO FAUSTO[1,2], (Student Member, IEEE), GIOVANNI GAGGERO[1], (Member, IEEE), FABIO PATRONE[1], (Member, IEEE), AND MARIO MARCHESE[1], (Senior Member, IEEE)
[1]Department of Electrical, Electronics and Telecommunications Engineering and Naval Architecture—DITEN, University of Genoa, 16145 Genoa, Italy
[2]Nozomi Networks Italia, 20156 Milan, Italy

Corresponding author: Mario Marchese (mario.marchese@unige.it)

ABSTRACT Software Defined Networking (SDN) is a very useful tool not only to manage networks but also to increase network security, in particular by implementing Intrusion Detection Systems (IDS) directly into the SDN architecture. The implementation of IDS within the SDN paradigm can simplify the implementation, speed up incident responses, and, in general, allow to promptly react to cyber attacks through proper countermeasures. Nevertheless, embedding IDS within SDN also introduces delays that cannot be tolerated in specific network environments, like industrial control systems. This paper focuses on the implementation of an IDS based on Machine Learning (ML) algorithms into an SDN architecture and proposes a very practical approach to reduce the delay by using the sequential implementation of prototypes of increasing software and hardware complexity so allowing quick tests to highlight the main problems, solve them and pass to the next operative step. A fully validated performance evaluation is then shown by exploiting all the presented solutions and by using further improved hardware features. The overall performance is very good and compliant with most, even if not yet all, industrial control systems constraints. Results show how the proposed solutions provide a significant improvement of the latency so opening the door to a real implementation in the field.

INDEX TERMS Cybersecurity, intrusion detection system (IDS), software defined networking (SDN), OpenFlow, key performance indicators (KPI).

I. INTRODUCTION

A cybersecurity technology pillar is represented by Intrusion Detection Systems (IDSes), which are hardware/software components or groups of devices designed to monitor networks and systems to detect malicious activities. There are several IDS classification models, depending on, for example: 1) the position of the detector: network-based intrusion detection systems (NIDS) vs host-based intrusion detection systems (HIDS); 2) the performed action: active IDS, also called Intrusion Prevention Systems (IPS), versus passive IDS; 3) the detection approach: Signature-based IDS, which look for specific data patterns, vs Anomaly-based IDS, which identify the anomalies with respect to a behaviour considered normal; 4) the analysis performed on network traffic: IDS based on deep packet inspection implemented at application layer where the application data are analysed in detail vs statistical fingerprint based IDS, typically implemented at TCP/IP layer by exploiting network statistics.

Intrusion detection systems are nowadays making a large use of machine Learning technologies [1]. They can use traditional algorithms, or more sophisticated deep learning-based algorithms [2], depending on the complexity of data. The proposal reported in this paper derives from a research work started with paper [3] that introduces an IDS based on statistical analysis which, after extracting a fingerprint for each

The associate editor coordinating the review of this manuscript and approving it for publication was Engang Tian.

network flow, uses a Machine Learning (ML) classifier to decide whether a network flow is affected or not by Botnet Malware.

In parallel with the evolution of IDSes, the need to simplify network management has led to the development of the Software Defined Networking (SDN) paradigm which is based on the decoupling of data and control planes. Data forwarding functions are located inside devices (switches, routers, gateways) called SDN switches, while control functions are concentrated in the SDN controllers. The communication between SDN controller and switches is handled through the OpenFlow (OF) signalling protocol. The SDN architecture is also a very useful tool to enhance cybersecurity [4] and has several applications in the Industrial Control System (ICS) field, including the power sector [5], especially to improve incident response. SDN allows increasing the control system resiliency, thanks to the possibility to dynamically re-configure the network after the detection of a fault or of a compromised device, allowing to operate even in degraded conditions. Resilience is particularly useful for control networks within critical infrastructures, which require an extremely high availability over time. Embedding an IDS within an SDN architecture would allow to simplify network configurations, speed up responses and reactions to attacks, and improve system efficiency and resilience. In this context, an highly relevant issue is the delay that the implementation of SDN brings [6]. The introduced latency is particularly critical in industrial control systems, in which we can find severe constraints in terms of Quality of Service (QoS) [7]. In more detail, the implementation of an IDS within an SDN controller may lead to the introduction of delays in the process of packet forwarding, which may be not tolerable for QoS to be assured in some specific networks, like the ones dedicated to an ICS, in particular in the electrical sector where packets have to respect very stringent requirements in terms of latency. For example, the packets of the GOOSE protocol, which belongs to the IEC 61850 suite for electrical substation automation, have to reach its destination within 10 milliseconds. It becomes therefore necessary to deeply analyse the delays introduced by an IDS in a SDN-based network so to check the feasibility of the implementation in such environments, also in view of a future implementation in 5G scenarios whose one of the pillars is just SDN. From the point of view of the authors of this paper, the original idea to implement a statistical fingerprint (SF) based IDS detector for Botnet Malware, specifically an adaptation of the one in [3], within an SDN controller is contained in [8]. The SF-IDS and the entire SDN-based architectural solution is identified as SDN-SF-IDS and is summarised in the next Section for the sake of completeness.

The aim of the present paper is to analyse and reduce the delays introduced by the SDN infrastructure in a Ethernet-based network, in order to allow the implementation over industrial environments such as the ones based on SCADA systems, which usually have severe constraints in terms of allowed latency.

The paper is structured as follows. Section II frames the work presented in this paper in the context of the state of the art. Section III contains a summary of the IDS proposed in [3] and its adaptation to be embedded in the SDN architecture [8]. Section IV summarises the implementation of the SDN-SF-IDS used as a testbed for this paper. The series of tests made to identify the main causes of delays and introduce improvements are shown in Section V. A full performance evaluation is reported in Section VI together with a discussion about the results and some considerations over possible future developments. Section VII contains the conclusions.

II. RELATED WORKS

The use of Machine Learning in Intrusion Detection Systems is useful in different scenarios and is significantly growing in importance. [9] and [10] propose a method to detect malicious Bot-IoT traffic in IoT Network, highlighting the importance of the feature selection phase in the design of the algorithm, which is significant also in the context of smart cities [11]. [12] proposes a new framework model and a hybrid algorithm for the selection of effective machine learning algorithms to detect Bot-IoT attacks in IoT environments over smart cities. [13] discusses how SDN and Network Function Virtualisation (NFV) technologies can help design automatic incident-response mechanisms for an ICS. [14] proposes an attack detection and localisation algorithm and designs an intervention strategy in the networked robot control field. A software-defined approach to secure field zones in an ICS is shown in [15]. The implementation of IDS within SDN controllers is proposed in [16], [17], [18], and [19] that provides a survey on SDN based network intrusion detection systems based on machine learning approaches. Few papers measure the impact of an IDS implemented over SDN architectures: [20] presents a QoS comparison of two open source network intrusion detection systems (Snort IDS and Bro IDS) in a SDN architecture. A preliminary analysis of the delay introduced by the implementation of IDS over SDN is presented by the same authors of this paper in [21].The analysis reported in [21] is limited to the results contained in Section V and omits many implementation details and comments reported here. All the results reported in Section VI are totally new and not discussed before.

This paper presents a full prototype based methodology to understand and tackle the numerical impact of the IDS implementation over SDN, reports all the implementation details, and reports a full performance evaluation, which allows to consider this paper a real step towards a real implementation in the field.

III. SDN-SF-IDS ARCHITECTURE

The network-based IDS, called SF-IDS (statistical fingerprint-IDS), originally introduced in [3], is aimed at deciding whether an IP flow is malware affected or not. SF-IDS uses the typical flow definition of TCP/IP networks: IP Source and Destination (IP SRC and DST), TCP/UDP Source and Destination Ports (SRC and DST Ports), and

Protocol field in the IP packet header. It is structured into a training phase developed by using a ground truth of known flows and an operative classification and decision phase. Both training and classification/decision phases are based on the definition and extraction of a group of statistical parameters related to each IP flow, which represent the statistical fingerprint of the flow, and on machine learning-based classifiers devoted to distinguish normal from malicious traffic. The key idea is that the statistical fingerprint associated to each flow is enough to infer the possible malicious nature of the flow. [3] defines a statistical fingerprint composed of the following 14 parameters indicated as features of each flow:

- Number of packets
- Number of bytes
- Duration of the flow in seconds
- Byte rate
- Packet rate
- Average inter-arrival time of packets
- Standard deviation of inter-arrival time
- ''Entropy'' of the packet lengths
- Total number of subsets of packets having the same length divided by the total number of packets of the flow
- Length of the first packet
- Length of the longest packet
- Length of the shortest packet
- Average packet length
- Standard deviation of the packet length

[3] compares 15 machine learning-based classifiers by using a group of Botnet malware such as Cutwail, Purple Haze, Ramnit, Tbot, Zeus, ZeroAccess, AlienspyRAT, Kuluoz, Sality, together with traffic classified as Normal. Tests allow the evaluation of each single classifier and the choice of the best ones by using a metric composed of the sum of false alarms and missed detections. Random Forest is the best one but also J48 and PART provide excellent results. Random Forest assures a null percentage of false negatives and false positives for Kuluoz, Tbot, and ZeroAccess, and very close to null percentage for Sality and Zeus. It also assures satisfying results for Cutwail and is very efficient to recognise AlienspyRAT and Ramnit but it has some difficulties to identify normal traffic, often interpreted as AlienspyRAT or Ramnit. The first problem to tackle towards the integration of SF-IDS and SDN is that the SDN standard does not allow to get all 14 parameters used in [3] and listed above. [22] proposes the reduction of the features involved for malware detection to be compliant with the SDN-OpenFlow standard and also with the features that most SDN switches available in the market can really measure. The reduced statistical fingerprint is composed of the following 7 parameters:

- Number of packets
- Number of bytes
- Duration of the flow in seconds
- Byte rate
- Packet rate
- Average packet length

Again the tree based classifiers Random Forest and J48 achieve excellent performance in terms of accuracy, not so far from the one obtained by using the 14 component statistical fingerprint.

As said, the original idea to implement the SDN-SF-IDS architecture is contained in [8] where the proposed system exploits the 7-features Random Forest based SF-IDS described above to detect the possible presence of malware inside the network. [8] describes the design and preliminary implementation within a Ryu-based SDN controller. The system is tested by using the following Botnet malware: Asprox, Cutwail, Darkness, Madness, Purplehaze, Zeus, Alienspy, Kuluoz, Neris, Ramnit, Tbot, and Zeroaccess. The results obtained through a large simulation campaign show the effectiveness and robustness of the proposed system, which reaches an accuracy level ranging from 88% to 97%. The effect of the delay introduced by the SDN infrastructure, fundamental to go towards a real testbed, is totally ignored in this phase and, as said, is the object of this paper.

IV. SDN-SF-IDS IMPLEMENTATION

The description of the architecture of the proposed solution is reported hereinafter. Section IV-A describes the basic structure of the SDN-SF-IDS, while IV-B contains the architecture of the testbed used for the collection of the measures shown in Section VI.

A. ARCHITECTURE OF SDN-SF-IDS

Conforming the SDN architecture, the SDN-SF-IDS prototype used as a testbed consists of two logically and physically separated parts: the SDN controller and SDN switch (Figure 1).

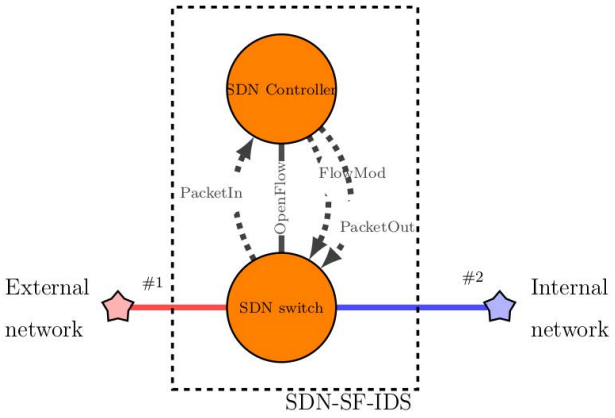

FIGURE 1. Structure of the SDN-SF-IDS prototype.

Open Source software and Operating System (OS) GNU/Linux are used for its implementation in order to be able to operate, if necessary, on the source code of each system component. The SDN controller, starting from the description reported in Section III, is built by using Ryu Open Source software and contains the 7-features SF-IDS implemented by applying the ''Random Forest'' machine-learning

scheme provided by the Scikit-learn Python library. The SDN switch routes packets on the monitored network and works with the SDN controller to perform the statistical analysis of the flows. The SDN switch can be implemented by:

- Software switch: a computer running OS GNU/Linux and Open virtual Switch (OvS) software that simulates the operation of an SDN switch by managing the routing of the network packets received by the interfaces.
- Hardware switch: a network device (switch, router, etc).

In both cases the communication between the switch and controller takes place by the OpenFlow protocol. The performance evaluation is essentially carried out through software switches whose features and functionalities are improved in each test. Only a group of final tests are carried out by using a hardware switch.

For each packet of a given flow received from interfaces #1 and #2, the SDN switch checks if its OF flow table contains a rule concerning the packets of that flow. If a rule is present, the SDN switch applies it to the entering packet. For example, the packet can be forwarded or dropped. When fully operational, OF flow tables contain the management rules for each flow whose packets are routed through the switch. Otherwise, if no rule is present, the SDN switch sends a OpenFlow "PacketIn" message to the SDN controller to ask how to process it. The SDN controller receives this message and sends it to the IDS code, implemented within the SDN controller, which stores the data related to the new flow within its own data structure, analyses the packet, and then, by using the OpenFlow "FlowMod" message, adds the appropriate rule or set of rules in the OF flow table of the SDN switch to monitor and forward the data flow. As a last step, the SDN controller sends the OpenFlow "PacketOut" message to the switch to allow the packet to be forwarded. Figure 2 shows the sequence of message exchange within the SDN-SF-IDS infrastructure when the SDN switch receives a new flow. Figure 3 shows the case when the flow is already present in the OF flow table.

The training of the ML algorithm is performed by using a labelled data set of network flows that last about three days and contain both traffic affected by Botnet Malware and normal network traffic used in [8]. During this operation, the SDN-SF-IDS system calculates and stores the statistics and, once an experimentally chosen threshold in terms of number of analysed flows is reached, starts the training phase of the "Random Forest" algorithm as described before. Once the training is finished, the IDS code saves the obtained parameters inside a special file loaded during the SDN-SF-IDS execution. When the class ("Botnet Malware or "Normal") of each monitored flow is decided, a drop rule may be added to the OF flow table of the SDN switch for each flow classified as a "Botnet Malware". If necessary, it is possible to change this rule in order to forward the flow catalogued as a malware to a specific network port of the SDN switch where additional analysis systems can be connected. Each of these

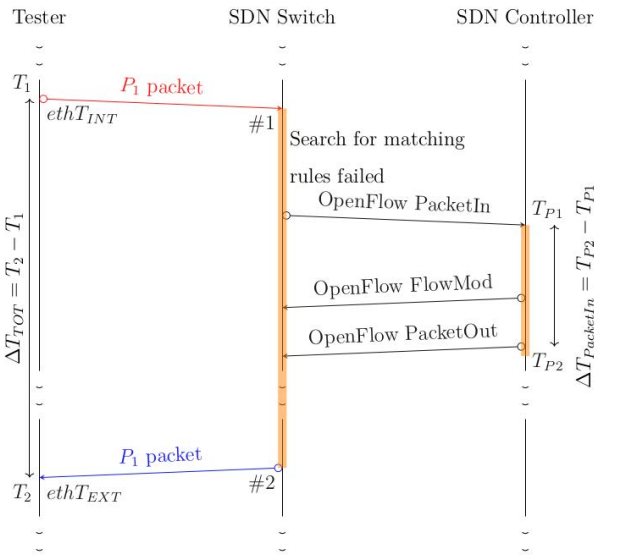

FIGURE 2. Timing diagram of the Openflow message exchange following the arrival of the first packet of a new flow.

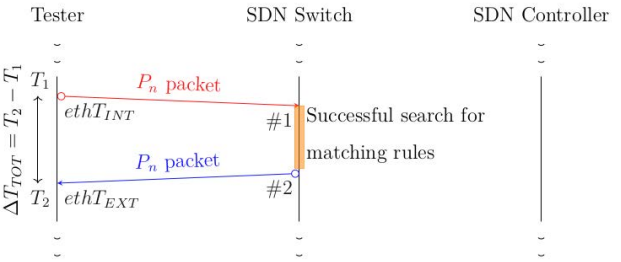

FIGURE 3. Timing diagram of the Openflow message exchange following the arrival of the n-th packet of a previously detected flow.

operations introduces delays in the forwarding of the packets and impacts on the QoS.

B. ARCHITECTURE OF THE TESTBED

In order to perform the evaluation of delays, the testbed in Figure 1is extended by including a measurement system. The laboratory tests are carried out by using the logic scheme presented in Figure 4. #1 and #2 interfaces of the SDN-SF-IDS are connected to a special system "Tester" that can generate and receive a stream of Ethernet frames (each of them encapsulating a single IP packet) and estimate the delay suffered by each individual Ethernet frame (IP packet).

The Tester allows performing the delay measures but does not influence the testbed behaviour. The Tester sends the network flow to be filtered from the internal Ethernet interface $ethT_{INT}$ to the port #1 of the SDN switch and receives the filtered flow outgoing from port #2 of the SDN switch on the interface $ethT_{EXT}$. If T_1 is the instant when the frame leaves the $ethT_{INT}$ interface and T_2 the instant when the frame returns from the $ethT_{EXT}$ interface, it is possible to estimate the total delay ΔT_{TOT} generated by the SDN-SF-IDS through

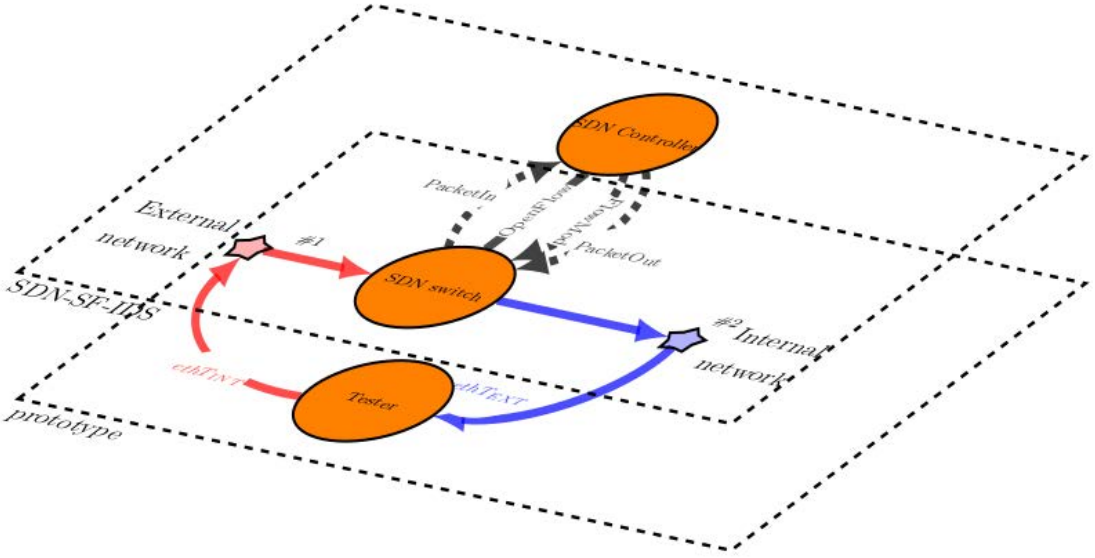

FIGURE 4. Logical connection of SDN-SF-IDS interfaces with the Tester.

the trivial subtraction in (1).

$$\Delta T_{TOT} = T_2 - T_1 \tag{1}$$

V. DELAY ANALYSIS AND PROPOSED SOLUTIONS

The SDN-SF-IDS system performs different operations each introducing peculiar delays. Particularly significant are the delays related to the exchange of OF messages between SDN switch and controller as well as the internal delays within the two systems. Of course delays are greater as they involve packet forwarding to the SDN controller and their analysis. As a first example delays depend on whether the received packet belongs to an already known flow, i.e. to a flow that has rules already present in the OpenFlow flow table, or not, because, for each new flow, the SDN controller must send two OF messages (''FlowMod'' and ''PacketOut'') to the SDN switch. Delays depend also on the Operating System (OS) of the SDN-SF-IDS infrastructure because, to perform the analysis of a single network packet, many software components must communicate and interact with each other and many lines of code come into play such as Ethernet driver reception and transmission queues, context switch between kernel and user space. Each of these interactions adds a delay that needs to be reduced as much as possible. Of course delays depend also on the hardware features of the involved SDN controllers and switches.

Prototypes of increasing complexity are created to overcome the bottlenecks gradually highlighted by the tests carried out in order to improve the performance in terms of latency. Following the software switch concept presented before, the two logical parts of the SDN-SF-IDS infrastructure are created by using two computers: the first (identified as A) acts as an SDN controller; the second (identified as B) as an SDN switch. As the tests progress, five different versions of the SDN-SF-IDS infrastructure are implemented. For the first three configurations the same controller A_1 is used by varying only the SDN switch B_i so getting configurations A_1B_1, A_1B_2, and A_1B_3, with the aim of obtaining increasingly reduced latency at the cost of software complexity and hardware requirements. The hardware features of A_1, B_1, B_2, and B_3 are shown in Table 1.

The first set of tests aim to determine the influence of the SDN switch/controller computational power and features on the delays suffered by the analysed packets and to introduce modifications to software and hardware to limit the problems. This phase is not identified as performance evaluation since the evaluation period is limited to tens of minutes and does not extend to tens of days as done later. The results of this phase have been carried out through the quick implementation of prototypes allowing a rapid practical evaluation so to pass to the next operative steps.

A. A_1B_1

The A_1B_1 configuration brings out two SDN-SF-IDS problems related to each other and to the SDN controller: the implementation introduces delays both in the management phase of the new flows and in the phase of acquiring the statistics, carried out periodically, aimed at classifying and filtering the flows. With reference to the management of new flows, peculiar delays are introduced for the first packet of each new flow. An important part of this delay is linked to the fact that the first packet of each new flow does not find a dedicated rule within the OF flow table of the SDN switch. The first tests are based only on the measurement of the time $\Delta T_{PacketIn}$, defined in Figure 2, needed to process a single OF ''PacketIn'' message. An example behaviour of $\Delta T_{PacketIn}$ over time is shown in Figure 5, where shown minimum,

TABLE 1. Hardware features of A_1, B_1, B_2, and B_3.

Controller A 1 : CPU Intel ® CoreTM i5-3340 running at 3.10 GHz, 16 GByte RAM, 4 Eth 1000 Mbit/s	
Testbed	description of SDN switch B_n software (SW) o hardware (HW)
A1B1	SW: CPU AMD Geode running at 266 Mhz, RAM 128 MByte, 3 Eth 10/100 Mbit/s
A1B2	SW: CPU Intel Q6600 running at 2.4 Ghz, 4 GByte RAM, 3 Eth 10/100/1000 Mbit/s
A1B3	SW: CPU Intel i5-7400 running at 3.00 GHz, 16 GByte RAM, 3 Eth 10/100/1000 Mbit/s

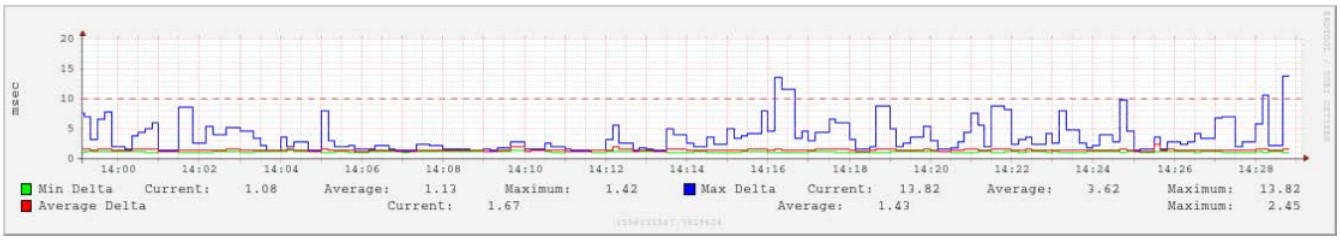

FIGURE 5. $\Delta T_{PacketIn}$ over time - single "PacketIN" (IDS with array structure, prototype A_1B_1).

maximum and average times are averaged over a window of 10.00 [s]. Maximum measured delay peak is 13.82 [ms] in this example test that highlights the slowness of the search and insertion phases of the flows within the data structure used for their storage, based on arrays.

The attempt to improve is to change the data structure based on arrays (list) into a structure based on hash tables (dictionary) in the controller. Even if, as said, these results are just quick examples, the result is quite clear in Figure 6 where the maximum delay peak is reduced to 3.22 [ms] in operating conditions very similar to the ones experienced in Figure 5.

From deeper analysis, it emerges that the frequency of new flows detected by the SDN switch is not constant over time but there are peaks of new flows and consequently peaks of OpenFlow "PacketIn" messages sent to the SDN controller. "PacketIn" messages are processed in sequence so the delays add up with consequent rapid extension of response times.

Concerning the total delay ΔT_{TOT}, including the statistics acquisition, a sequence of samples over time concerning the A_1B_1 prototype by using the hash structure is shown in Figure 7. Again the minimum, maximum and average times averaged over a window of 10.00 [s] are shown. The average of maximum delays is 66.48 [ms] but it is possible to note a number of peaks well above 100 [ms] up to a maximum of 189.61 [ms], largely incompatible with many SCADA applications.

B. A_1B_2

Always using the hash structure, the impact of more performing system resources is tested by replacing the SDN B_1 switch with the B_2 one. The shown results are of the same type used in the previous subsection: measured ΔT_{TOT} samples over approximately 30 minutes are shown in 8.

The maximum delay peak is now 54.29 [ms]. The average maximum delay is 5.96 [ms] and it is visible a limited number of delays above 10 [ms].

C. PROCESSES ACTING IN PARALLEL TO IMPROVE IDS PERFORMANCE

A portion of ΔT_{TOT} is due to the statistics acquisition phase. Statistics computations and cataloguing action performed by the ML algorithm are carried out within the main Ryu process. For this reason, Ryu cannot process other requests, including the management of new flows, until the end of the cataloguing phase. Required time is not negligible. To solve this problem, the analysis of the flow statistics and their classification need to be moved to a special thread which can process them in parallel so that Ryu main process is free to handle other OpenFlow messages sent by the SDN switch. In more detail, the mentioned problem is strictly related to the structure of the Ryu software which is based on a single process programming model. The management of each received OpenFlow packet is done sequentially, so parsing the OpenFlow "$STAT_{REQ}$" message blocks Ryu execution until the operation is finished. Meanwhile other OF messages sent to the SDN Controller A_1 remain stuck in the system queue of the socket associated with OpenFlow port 6633 without being processed. This problem occurs for each received OF message but it is particularly critical during the reception of the OpenFlow "$STAT_{RESP}$" message as, after receiving the message, the IDS module performs the extraction of the statistical features and the cataloguing of the flows. Furthermore, the statistics calculation procedure is repeated at regular time intervals so blocking Ryu main process for a variable time depending on the number of packets to be catalogued. This aspect causes additional congestion in the handling of OpenFlow "PacketIn" messages and consequent delay in the management of new flows. To tackle the problem, the IDS module of the SDN-SF-IDS is enhanced by adding two running threads acting in parallel with the main Ryu process identified as "*Ryu*": the thread "*Monitor*" that cyclically wakes up and requests the SDN switch to send the statistics of the flows it monitors and then returns to sleep, and the thread to compute statistics identified as "*Statistics*"

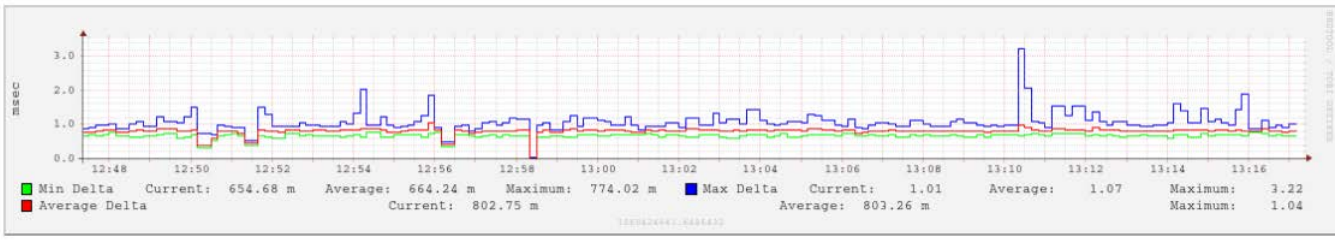

FIGURE 6. $\Delta T_{PacketIn}$ over time - single "PacketIN" (IDS with hash structure, prototype A_1B_1).

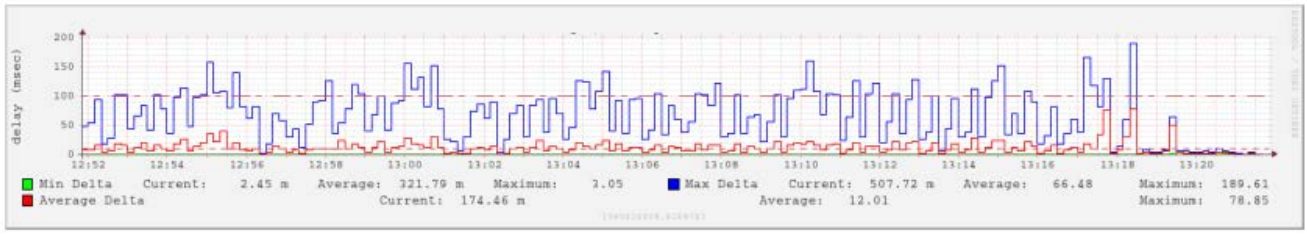

FIGURE 7. Packet delays ΔT_{TOT}, prototype A_1B_1, hash structure.

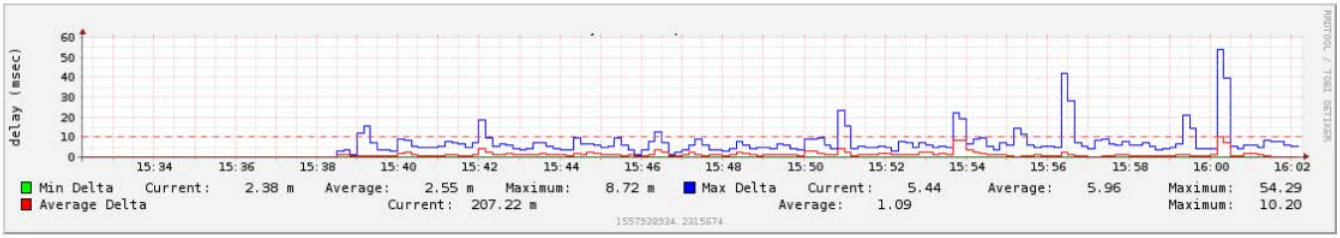

FIGURE 8. Packet delays ΔT_{TOT}, prototype A_1B_2.

that is awakened by the main process when the statistics are ready to be computed. "*Statistics*" moves the necessary data within a data structure reserved for it. In this way, "*Statistics*" and "*Ryu*" act in parallel without concurrent access to data. "*Ryu*" signals the presence of new data to "*Statistics*" through a special Semaphore and immediately starts managing the OpenFlow messages again, finding the data structure empty and ready to store the data of new flows. "*Statistics*" computes the statistics of the flow and performs the cataloguing by using the "Random Forest" algorithm, then, if necessary, it proceeds by adding drop rules in the switch and finally goes back to rest. The execution time is limited to few milliseconds at each statistics request interval, configurable by the user and set to 220 [s] in all shown tests. Figure 9 shows a simplified diagram of the interaction "*Ryu*" and parallel "*Monitor*" and "*Statistics*" threads.

The first attempts to implement the solution in Figure 9 over the testbed A_1B_2 were not successful: a performance limit emerged due to the software nature of the SDN switch running inside a general purpose OS, such as GNU/Linux. An analysis of the state of the art [23] shows that the presence of a scheduler that manages the execution of parallel processes and other strictly technical elements introduces additional waiting times that cannot be reduced. Data Plane Development Kit (DPDK) can be used to overcome this problem. This development kit has been designed to ensure faster handling of packets received by Ethernet cards at the price of a significant increase in complexity. The DPDK libraries work in close synergy with the Linux kernel and require the presence of the most advanced architectural solutions offered by today's CPUs (Hugepages, IOMMU, VT-x, VT-d, etc.).

D. A_1B_3

B_2 switch does not support the minimum hardware requirements to use DPDK5, so B_3 switch equipped as detailed in Table 1 is exploited. The DPDK library version 19.05.0 6 is compiled and installed on system B_3. The OvS software, shown in Figure 9, version 2.11.1 8, is compiled to take advantage of the DPDK libraries. The shown results are of the same type used in Figures 7 and 8. Figure 10 contains ΔT_{TOT} values over approximately 30 minutes time.

The maximum delay peak is 11.76 [ms] and, in this small example period, only two peaks over 10 [ms] are measured. The average maximum delay is 4.59 [ms].

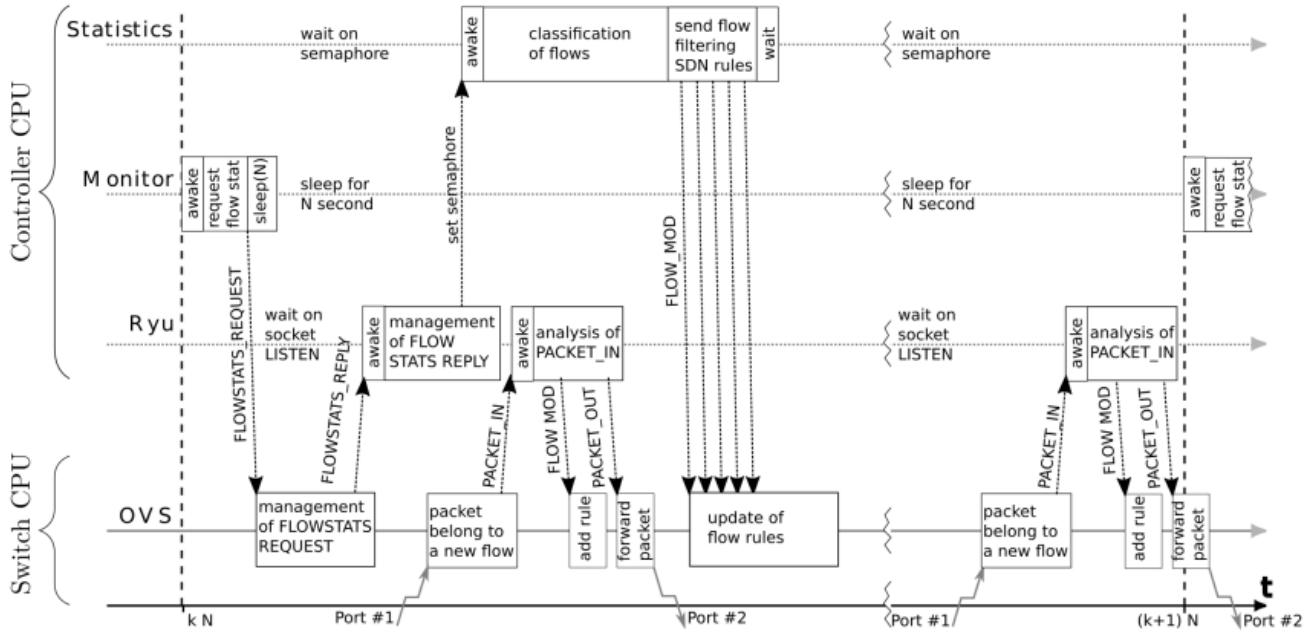

FIGURE 9. Simplified diagram of the interaction between main process "*Ryu*", "*Monitor*" and "*Statistics*" threads, acting in parallel.

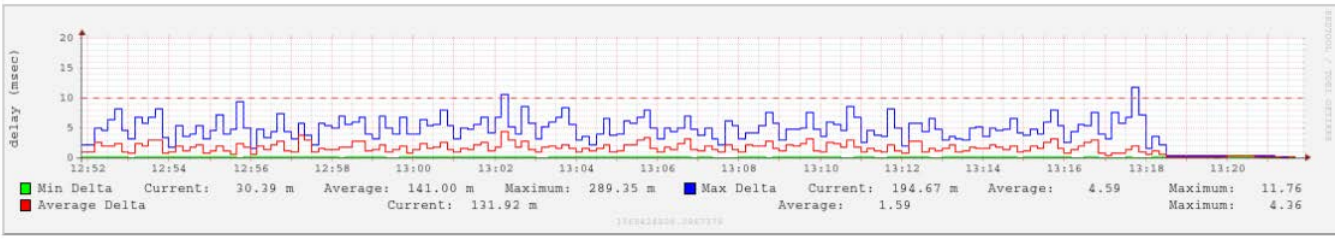

FIGURE 10. Packet delays ΔT_{TOT}, prototype A_1B_3.

TABLE 2. Hardware features of the controller A_2 as well as of the switches B_4, and B_5 and related measured ΔT_{TOT}.

A 2 : CPU Intel Xeon E3110 running at 3.00 GHz, 2 GByte of RAM, 3 Eth 1000 Mbit/s		
Testbed	description of SDN switch B_n software (SW) o hardware (HW)	ΔT_{TOT}
A_2B_4	software switch CPU Intel Core i5-3450S running at 2.80GHz 8 GByte RAM, 3 Eth 10/100/1000 Mbit/s	95.36% of packets <=10ms, peak of 150ms
A_2B_5	hardware switch HPE Aruba M2940-48G WC.16.07.0003	98% of packets <=10ms, peak of 150ms

VI. PERFORMANCE EVALUATION

Performed tests highlight the main problems and are a fundamental step to determine the solutions for the design of the SDN-SF-IDS. It is now time to present a fully validated performance evaluation by exploiting all the presented solutions and also by using further improved hardware features. Table 2 contains the hardware features of the controller A_2 as well as of the software switch B_4, and of the hardware switch B_5. The test data set includes months of traffic from the LAN of our research lab and all the Pcaps at our disposal containing Botnet. In detail, 82,999,983 samples have been taken over a 2,377,497 [s] (between 27 and 28 days) monitoring phase. The statistics request interval is again set to 220 seconds in the tests shown in this Section. The results have the aim of checking if the IDS is suitable to be applied in industrial networks characterised by severe KPIs. Table 2 shows in its last column a summary of the ΔT_{TOT} obtained by using the configurations A_2B_4, mainly used in the remainder of the paper, and A_2B_5. A high level software switch such as B_4 allows keeping 95.36% of packets below 10 [ms]. An hardware switch allows improving: 98.1% of packets are below 10 [ms]. Maximum measured delay is 150 [ms] in both case. The average results in Table 2 may be further investigated by looking at Figure 11 which shows the delay absolute frequency distribution histogram by using the classes in the abscissa axis for the A_2B_4 entire performed tests in the

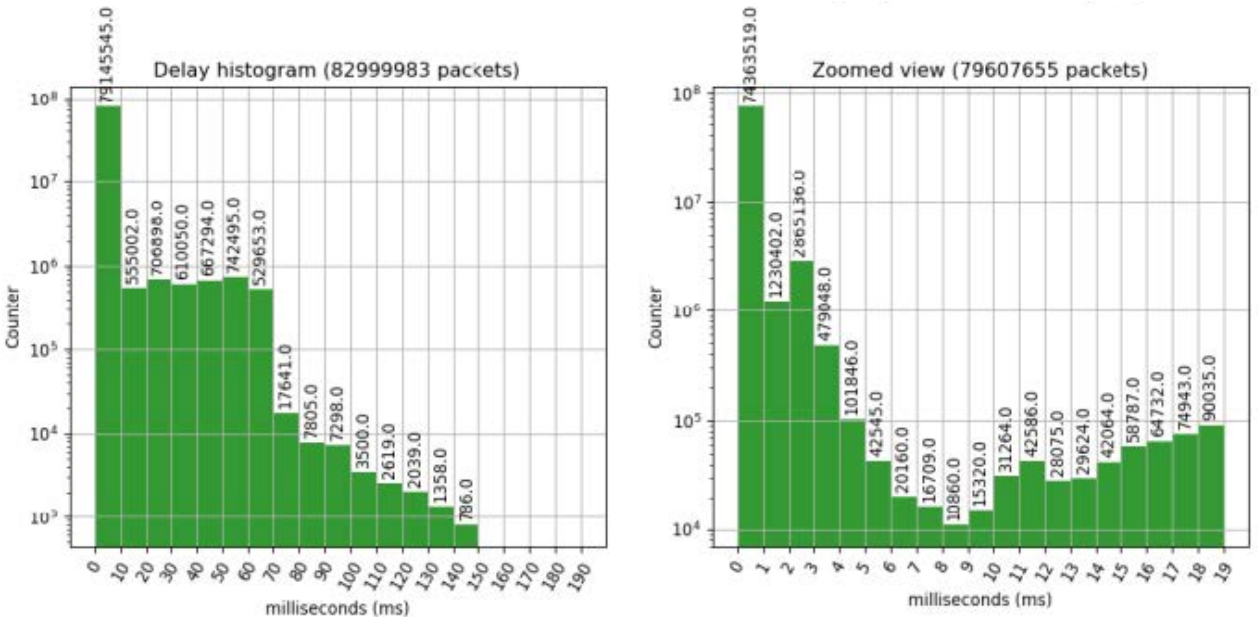

FIGURE 11. Delay absolute frequency distribution histogram for the performed tests in the configuration A_2B_4.

TABLE 3. Delay relative frequency distribution for the performed tests in the configuration A_2B_4.

Delay classes [ms]	Relative frequency distribution
0-10	0.953560978
10-20	0.006686772
20-30	0.008516845
30-40	0.007350002
40-50	0.008039688
50-60	0.008945725
60-70	0.006381363
70-80	0.000212542
80-90	9.40362E-05
90-100	8.79277E-05
100-110	4.21687E-05
110-120	3.15542E-05
120-130	2.45663E-05
130-140	1.63614E-05
140-150	9.46988E-06
above 150	0.0

TABLE 4. Cumulative relative delay frequency distribution for the performed tests in the configuration A_2B_4.

Delay classes [ms]	Cumulative relative delay frequency distribution values
0-10	0.953560978
0-20	0.960247751
0-30	0.968764596
0-40	0.976114598
0-50	0.984154286
0-60	0.993100011
0-70	0.999481373
0-80	0.999693916
0-90	0.999787952
0-100	0.999875879
0-110	0.999918048
0-120	0.999949602
0-130	0.999974169
0-140	0.99999053
0-150	1.0

left part of the figure and its magnifying glass in the range [0 - 19]i [ms]. Table 3 shows the relative frequency distribution of the delay versus the delay classes used in Figure 11. Table 4 shows the cumulative relative frequency distribution of the delay in the same conditions. Values contained in Table 4 are very meaningful: associating relative frequencies to probabilities, the probability to have a delay below 10 [ms] is 0.9536, below 20 [ms] 0.9602, below 50 [ms] 0.9842, below 80 [ms] 0.9997, below 100 [ms] 0.9999, and below 150 [ms] is 1. SCADA and GOOSE-based industrial networks have very stringent KPIs for maximum delays [24]. These KPIs imply device-to-device transfer times below 20 [ms] for non-tripping and P2/P3 class messages and below 100 [ms] for non-tripping and P1 class messages. Concerning these classes the results obtained in this paper may be satisfying: the probability that the delay is below 20 [ms] is above 0.96 and the probability that the delay is below 100 [ms] is close to 1. The KPI of tripping messages and intervention class P1 messages require transfer times below 10 ms. As far as this class is concerned, the probability we can assure is 0.95. Required KPIs for tripping and P2/P3 intervention class messages must be below 3 [ms]. Concerning these messages, the left part of Figure 11 may be of help: the probability to have a delay below 1 [ms] is 0.8959, below 2 [ms] 0.9108, and below 3 [ms] 0.9453. The obtained result is somehow compatible also with a delay requirement fixed to 3 [ms] even if it would be recommendable to further reduce the latency of the SDN-SF-IDS system in this case. These long tests, lasting almost one month, composed of 82,999,983 samples, allow additional investigation by differentiating the protocol encapsulated in IPv4 through the Protocol field in the IP

TABLE 5. Number of samples, ethertype, Protocol field (if any or, alternatively, Protocol encapsulated in Ethernet), maximum measured delay for delay class, delay class, percentage of packets in the given delay class - configuration A_2B_4.

Number of samples	Ethertype	Protocol field (if any)	Maximum delay [ms]	Delay class [ms]	% of packets in the delay class
146	0x0800	0x00 IPv6 Hop by Hop	1	0-1	100.00
411,485	0x0800	0x01 ICMP	80	0-6	99.38
195,775	0x0800	0x02 IGMP	110	0-10	99.82
75,353,718	0x0800	0x06 TCP	150	0-10	96.96
2,639,983	0x0800	0x11 UDP	140	0-40	97.70
1,188,637	0x0800	0x27 TP++	8	0-9	9.99
3	0x0800	0x29 IPv6 encapsulation	3	2-3	100.00
2	0x0800	0x2F GRE	18	2-18	100.00
1,183,910	0x0800	0x32 ESP	7	0-1	99.99
237,477	0x0800	0x59 OSPF	110	1-3	99.47
80,419	0x0800	0x67 PIM	100	1-3	99.52
1,680,234	0x0806	ARP	9	0-1	99.99
28,194	0x86dd	IPv6	2	0-1	99.98
82,999,983	all	all	150	0-10	95.36

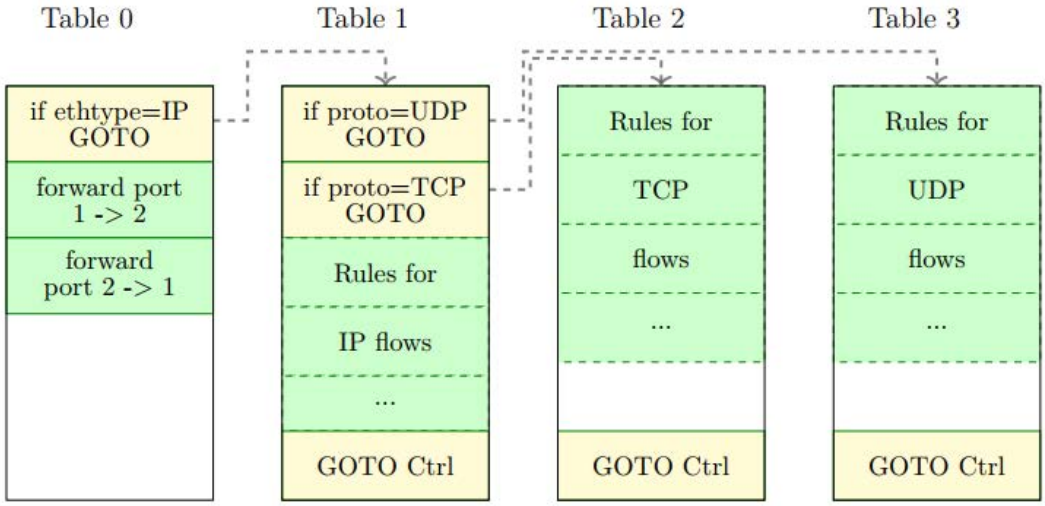

FIGURE 12. Logical structure of the OF rules as used by the A_2 controller.

header or, alternatively, if no IPv4 packet is transported over Ethernet, either ARP or IPv6. Table 5 shows: number of samples, ethertype values, Protocol field or Protocol encapsulated in Ethernet, Maximum measured delay for delay class, Delay Class, and Percentage of packets having the delay in the given class. There is a clear dependence between the delay suffered and the used protocol. This dependence is linked to the number of OpenFlow rules checked before packet forwarding, as depicted in Figure 12, and by the necessary interactions with the SDN sontroller. Non-IPv4-based protocols (ARP and IPv6 in Table 5) are forwarded after checking the rules of the first OpenFlow flow table with minimal delays. IPv4 protocols other than UDP and TCP are forwarded once the second OpenFlow flow table is reached and suffer longer delays; TCP packets are forwarded once the second OpenFlow flow table is reached, while UDP packets once the third OpenFlow flow table is reached, assuming they have already been analysed. If not, OpenFlow messages are exchanged with the controller causing longer delays. In more numerical detail, ICMP experiences delays up to 80 [ms] but 99.38% of packets is below 6 [ms], IGMP up to 110 [ms] but with 99.82% of packets below 10 [ms], OSPF up to 110 [ms] but with 99.47% of packets below 3 [ms], and PIM up to 100 [ms] but with 99.52% of packets below 3 [ms]. All the other protocols experience really low delays. TCP and UDP deserve additional attention. Figure 13 shows the delay absolute frequency distribution histogram by using the classes in the abscissa axis, for TCP packets, and its magnifying glass in the range [0 - 19] ms, as previously done in Figure 11. Figure 14 contains the same quantities concerning UDP packets and shows many samples between 0-4 [ms], less samples between 4-30 [ms], and a new increment between 30-70 (2.88%). Peaks between 0 and 4 [ms] are linked to the packets belonging to a flow already present in the OpenFlow flow table while the second peak between 30-70 [ms] is linked to the interaction with the controller for the insertion of the relative rule in the OpenFlow flow table. TCP packet delay behaves not so differently from the general histogram in Figure 11. Table 6 shows the cumulative relative frequency distribution of the delay for TCP packets and the reported

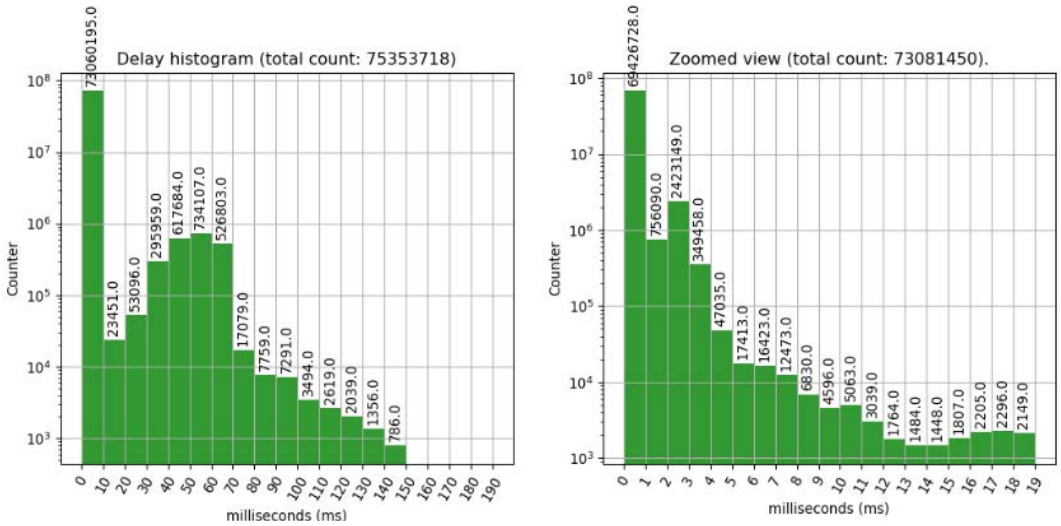

FIGURE 13. Delay absolute frequency distribution histogram for the performed tests in the configuration A_2B_4, TCP packets.

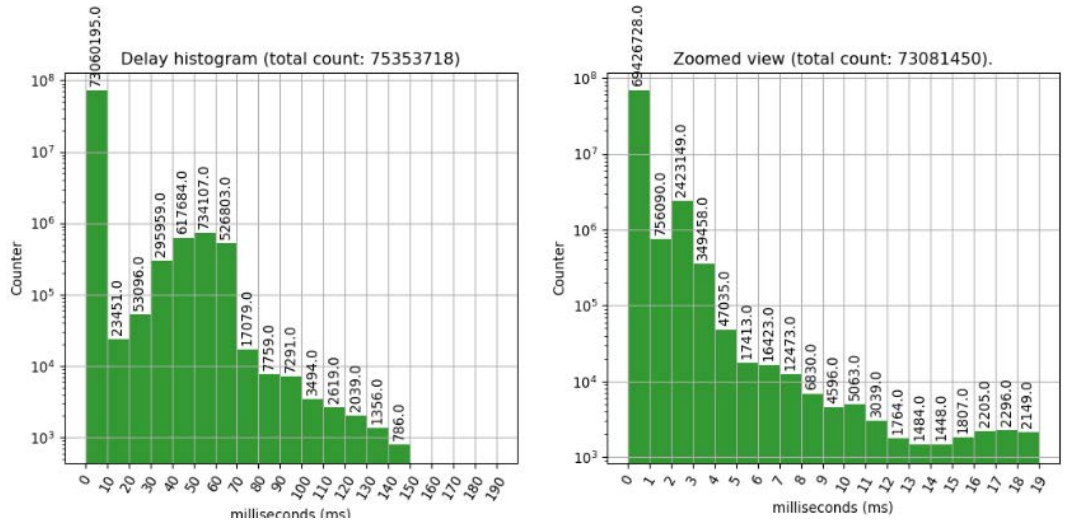

FIGURE 14. Delay absolute frequency distribution histogram for the performed tests in the configuration A_2B_4, UDP packets.

values are similar to the ones in Table 4. This may be also expected because 90.79% of the traffic is represented by the TCP one. UDP packet delay behaves differently. Table 7 shows the cumulative relative frequency distribution of the delay for UDP packets: again associating relative frequencies to probabilities, the probability to have a delay below 10 [ms] is 0.4094 (very low), below 20 [ms] 0.6105 (still very low for an industrial network), and below 30 [ms] 0.8581 (still far from industrial network requirements). The probability to have a delay below 40 [ms] is 0.9770, which is the value shown in Table 5. For delays above 40 [ms] the values are comparable with the general behaviour in Table 4. Actually, the histogram of the delays of the UDP protocol in Figure 14 shows a sort of plateau between 0-40 [ms] which rapidly decreases after 40 [ms], reaching negligible valus after 80 [ms] delays. The connectionless nature of UDP linked to the fact that UDP ports practically change for each packet causes a high interaction with the controller that has to catalogue new flows and add the appropriate rules. This action leads to noticeable increases in the delay.

The next steps to improve the performance may be in the optimisation of the Ryu SDN controller source code or in its replacement with a more efficient software. Alternatively it is possible to exploit the features of an hardware switch such as B_5 in Table 2. The corresponding configuration is A_2B_5. As already shown in Table 2 the probability to have a delay in the range [0-10] [ms] is 0.981, value that must be compared with 0.9536 obtained by using B_4. B_5 hardware switch has slightly better performance than B_4 software switch improved with DPDK but the performance is comparable. The effect of

TABLE 6. Cumulative relative delay frequency distribution for the performed tests in the configuration A_2B_4 - TCP traffic.

Delay classes [ms]	Cumulative relative delay frequency distribution values for TCP traffic
0-10	0.96956324
0-20	0.969874453
0-30	0.970579076
0-40	0.974506673
0-50	0.9827038
0-60	0.992445947
0-70	0.999437015
0-80	0.999663666
0-90	0.999766634
0-100	0.999863391
0-110	0.999909759
0-120	0.999944515
0-130	0.999971574
0-140	0.999989569
0-150	1.0

TABLE 7. Cumulative relative delay frequency distribution for the performed tests in the configuration A_2B_4 - UDP traffic.

Delay classes [ms]	Cumulative relative delay frequency distribution values for UDP traffic
0-10	0.409440894
0-20	0.61054825
0-30	0.858114995
0-40	0.977021822
0-50	0.995678381
0-60	0.998772719
0-70	0.999773483
0-80	0.999980303
0-90	0.999996212
0-100	0.999997727
0-110	0.999999242
0-120	0.999999242
0-130	0,999999242
0-140	1.0

hardware switching, anyway, should be further investigated. To date, the cost of a B_4 computer configured to operate as a high performance SDN switch is less than the cost of a hardware SDN switch B_5. Attention needs to be given to software, concerning unexpected software upgrade, CPU overload and memory exhaustion leading to swap memory on disk, which can lower the performance of the system.

VII. CONCLUSION

The present paper focuses on the reduction of delays introduced by the implementation of a statistical-based Intrusion Detection System within a Software Defined Networking infrastructure. The main causes of delays are experimentally identified together with the main solutions to be addressed. These solutions are implemented step by step and finally tested over a high quality software switch that guarantees very good performance compliant with most industrial control systems constraints:

1) industrial system device-to-device transfer time for non-tripping and P2/P3 class messages must be below 20 [ms] and for non-tripping and P1 class messages below 100 [ms]. Referring to all exchanged protocols in the reported tests, the probability that the delay is below 20 [ms] is above 0.96 and the probability that the delay is below 100 [ms] is close to 1.

2) KPI of tripping messages and intervention class P1 messages require transfer times below 10 [ms]. Concerning reported tests, the probability that the delay is below 10 [ms] is 0.95.

3) KPIs for tripping and P2/P3 intervention class messages require a delay below 3 [ms]. Concerning the obtained results, the probability to have a delay below 1 [ms] is 0.8959, below 2 [ms] 0.9108, and below 3 [ms] 0.9453.

The obtained results are somehow compatible also with a delay requirement fixed to 3 [ms] even if it would be recommendable to further reduce the latency of the SDN-SF-IDS system.

A chance to increase the performance is to use a hardware switch, which is tested and offers slightly better performance than the one of software switch improved with DPDK. The effect of hardware switching needs to be investigated in full detail as done for the software switch.

REFERENCES

[1] N. F. Haq, A. R. Onik, M. A. K. Hridoy, M. Rafni, F. M. Shah, and D. M. Farid, "Application of machine learning approaches in intrusion detection system: A survey," *Int. J. Adv. Res. Artif. Intell.*, vol. 4, no. 3, pp. 9–18, 2015.

[2] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[3] L. Boero, M. Cello, M. Marchese, E. Mariconti, T. Naqash, and S. Zappatore, "Statistical fingerprint-based intrusion detection system (SF-IDS)," *Int. J. Commun. Syst.*, vol. 30, no. 10, Jul. 2017, Art. no. e3225.
[4] R. Sahay, W. Meng, and C. D. Jensen, "The application of software defined networking on securing computer networks: A survey," *J. Netw. Comput. Appl.*, vol. 131, pp. 89–108, Apr. 2019.
[5] G. B. Gaggero, P. Girdinio, and M. Marchese, "Advancements and research trends in microgrids cybersecurity," *Appl. Sci.*, vol. 11, no. 16, p. 7363, Aug. 2021.
[6] D. Chefrour, "One-way delay measurement from traditional networks to SDN: A survey," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–35, Sep. 2022.
[7] S. Soucek and T. Sauter, "Quality of service concerns in IP-based control systems," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1249–1258, Dec. 2004.
[8] F. Bigotto, L. Boero, M. Marchese, and S. Zappatore, "Statistical fingerprint-based ids in SDN architecture," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst. (SPECTS)*, Jul. 2018, pp. 1–12.
[9] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "IoT malicious traffic identification using wrapper-based feature selection mechanisms," *Comput. Secur.*, vol. 94, Jul. 2020, Art. no. 101863.
[10] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: A malicious bot-IoT traffic detection method in IoT network using machine-learning techniques," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3242–3254, Mar. 2021.
[11] M. Shafiq, Z. Tian, A. K. Bashir, A. Jolfaei, and X. Yu, "Data mining and machine learning methods for sustainable smart cities traffic classification: A survey," *Sustain. Cities Soc.*, vol. 60, Sep. 2020, Art. no. 102177.
[12] M. Shafiq, Z. Tian, Y. Sun, X. Du, and M. Guizani, "Selection of effective machine learning algorithm and bot-IoT attacks traffic identification for Internet of Things in smart city," *Future Gener. Comput. Syst.*, vol. 107, pp. 433–442, Jun. 2020.
[13] A. F. M. Piedrahita, V. Gaur, J. Giraldo, A. A. Cárdenas, and S. J. Rueda, "Leveraging software-defined networking for incident response in industrial control systems," *IEEE Softw.*, vol. 35, no. 1, pp. 44–50, Jan./Feb. 2018.
[14] H. Sándor, B. Genge, Z. Szántó, L. Márton, and P. Haller, "Cyber attack detection and mitigation: Software defined survivable industrial control systems," *Int. J. Crit. Infrastruct. Protection*, vol. 25, pp. 152–168, Jun. 2019.
[15] J. Yang, C. Zhou, Y.-C. Tian, and S.-H. Yang, "A software-defined security approach for securing field zones in industrial control systems," *IEEE Access*, vol. 7, pp. 87002–87016, 2019.
[16] S. Seeber, L. Stiemert, and G. D. Rodosek, "Towards an SDN-enabled IDS environment," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 751–752.
[17] G. A. Ajaeiya, N. Adalian, I. H. Elhajj, A. Kayssi, and A. Chehab, "Flow-based intrusion detection system for SDN," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 787–793.
[18] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in *Proc. 7th Int. Conf. Emerg. Secur. Technol. (EST)*, Sep. 2017, pp. 138–143.
[19] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-Peer Netw. Appl.*, vol. 12, no. 2, pp. 493–501, Jan. 2019.
[20] H. Hendrawan, P. Sukarno, and M. A. Nugroho, "Quality of service (QoS) comparison analysis of snort IDS and bro IDS application in software define network (SDN) architecture," in *Proc. 7th Int. Conf. Inf. Commun. Technol. (ICoICT)*, Jul. 2019, pp. 1–7.
[21] A. Fausto and M. Marchese, "Implementation details to reduce the latency of an SDN statistical fingerprint-based IDS," in *Proc. Int. Symp. Adv. Electr. Commun. Technol. (ISAECT)*, Nov. 2019, pp. 1–6.
[22] L. Boero, M. Marchese, and S. Zappatore, "Support vector machine meets software defined networking in IDS domain," in *Proc. 29th Int. Teletraffic Congr. (ITC 29)*, Sep. 2017, pp. 25–30.
[23] R. Giller. *Open Vswitch With DPDK Overview*. Accessed: Mar. 30, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/open-vswitch-with-dpdk-overview.html
[24] S. Joshi, "Utilization of GOOSE in MV substation," in *Proc. 16th Nat. Power Syst. Conf.*, 2010, pp. 323–328.

ALESSANDRO FAUSTO (Student Member, IEEE) received the Laurea degree in computer engineering, in 2006, the master's degree *(cum laude)* in cyber security and data protection, in 2018, and the Ph.D. degree in electronic and telecommunication engineering from the University of Genoa, in 2022. Since 2010, he has been an ICT Technician with the Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa. From 2015 to 2016, he was a part of the 31st Italian Expedition in Antarctica and Winterover DC12 as an ICT, Radio and Telecommunication Technician at "Concordia" International Research Station on the Antarctic Plateau. Since 2022, he has also been a Security Researcher at Nozomi Networks Italia. His research interests include network security, intrusion detection systems, software-defined networks, and software-defined radio.

GIOVANNI GAGGERO (Member, IEEE) received the M.Sc. degree in electrical engineering and the Ph.D. degree in electronic and telecommunication engineering from the University of Genoa. He is currently a Postdoctoral Research Fellow at the Satellite Communications and Heterogeneous Networking Laboratory, University of Genoa. His research interests include network security of industrial control systems, microgrids, and smart grids.

FABIO PATRONE (Member, IEEE) is currently an Assistant Professor with the Satellite Communications and Heterogeneous Networking Laboratory, University of Genoa. His research interests include routing, scheduling, and congestion control algorithms in satellite, vehicular, and sensor networks, and the employment of networking technologies, such as network function virtualization and software defined networking for the integration of these networks with the terrestrial infrastructure within 5G.

MARIO MARCHESE (Senior Member, IEEE) received the Laurea degree *(cum laude)*, in 1992, and the Ph.D. degree in telecommunications from the University of Genoa, in 1997. From 1999 to January 2005, he worked with the Italian Consortium of Telecommunications (CNIT), where he was the Head of Research. From February 2005 to January 2016, he was an Associate Professor, and since February 2016, he has been a Full Professor at the University of Genoa, where he has been a Rector's Delegate to doctoral studies, since December 2020. He is the author of the book *Quality of Service over Heterogeneous Networks* (John Wiley & Sons, Chichester, 2007), and the author/coauthor of more than 300 scientific works, including international magazines, international conferences, and book chapters. His research interests include the field of networking, quality of service over heterogeneous networks, software-defined networking, satellite networks, network security, critical infrastructure security, and intrusion detection systems.