Advances in Science and Technology
Research Journal

Advances in Science and Technology Research Journal, 18(7), 176–191
https://doi.org/10.12913/22998624/192641
ISSN 2299-8624, License CC-BY 4.0

Received: 2024.06.14
Accepted: 2024.09.18
Published: 2024.10.06

Detection of Incidents and Anomalies in Software-Defined Network – Based Implementations of Critical Infrastructure Resulting in Adaptive System Changes

Patryk Organiściak[1], Paweł Kuraś[1*], Dominik Strzalka[1], Andrzej Paszkiewicz[1], Marek Bolanowski[1], Bartosz Kowal[1], Michał Ćmil[1], Paweł Dymora[1], Mirosław Mazurek[1], Veronika Vanivska[1]

[1] Department of Complex Systems, The Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, ul. MC Skłodowskiej 8, 35-036 Rzeszów, Poland

* Corresponding author's e-mail: p.kuras@prz.edu.pl

ABSTRACT

In the paper an example of an integrated software-defined network (SDN) system with heterogeneous technological instances based on the Linux platform will be shown. For this purpose, two research testing stands with a POX controller and OVS (Open vSwitch) switches were used. In the first testing stand, the research based on the ICMP traffic was done while in the second one, MQTT traffic was analysed. The capabilities of these systems were examined in terms of responding to detected incidents and traffic anomalies. In particular, their appropriate responses to anomalies were tested, as well as the possibility of continuous monitoring of packet transfer between separate network components. The aim of the paper is to investigate the effectiveness of SDN in enhancing the security and adaptability of critical infrastructure systems. For isolation and optimised resource management, some components, such as POX or the MQTT broker, were run in Docker containers. The test environment used both hardware cases and prepared software, enabling comprehensive design and testing of networks based on the OpenFlow protocol used in SDN architecture, enabling the separation of control from traffic in computer networks. The results of this research make it possible to implement anomaly detection solutions in critical infrastructure systems that will adapt on the fly to changing conditions that arise, for example, in the case of an attack on such infrastructure or physical damage to it at a selected node.

Keywords: anomaly detection, software-defined network, open vSwitch, open flow protocol, adaptive system changes.

INTRODUCTION

SDN systems are a new interesting and still very challenging paradigm that nowadays is used in many important practical applications [1, 2, 3]. The main idea assumes that in the case of these networks, a control plane (routing process) is separated from the data plane, i.e. network packets forwarding process. The first approaches to this idea in computer networks can be dated back to 2004 (see RFC 3746 document for instance [4]), however, the whole concept is not a new one, since earlier it was proposed and used in the case of switched telephone networks. As a result, the intelligence of classical, static traditional networks architecture is improved within one or more network components (e.g. SDN switches). Such an approach is seen as a paradigm change in computer engineering [5, 6, 7].

In this paper the authors propose a detailed practical approach where some development and applied research actions were taken and, as a final result, there were obtained some specific results that can be useful for those who are working with SDN networks and their possible applications in operational technology (OT) networks and critical infrastructure. 4 different scenarios were tested giving the evidences how some actions should be done and what kind of final result can be expected with specific network configurations.

Remark: because a significant part of the paper is related to several important listings with codes of programs, the reader is requested to visit an external repository (pastebin.com) as it is given by References [17–25]. The multicitation [17–25] includes references to various code snippets and configurations essential for setting up and managing the POX controller and handling MQTT (Message Queue Telemetry Transport) traffic in a Docker environment. Each of these references details specific steps and methods crucial for the implementation of the test environment, such as preparing the Docker image for POX with building and running an image [17, 18], handling incoming packets [19], constructing MQTT containers based on eclipse-mosquitto image [20], recognizing and processing MQTT packets [21], and managing traffic anomalies [22–25]. This information is vital for the accurate replication and understanding of the experimental setup and results discussed in the paper.

The aim of the paper is to investigate the effectiveness of SDN in enhancing the security and adaptability of critical infrastructure systems. By examining the detection of incidents and anomalies within SDN-based implementations, the study seeks to demonstrate how adaptive system changes can be implemented in response to various challenges. Through detailed testing setups and analysis, the paper aims to provide practical insights and evidence on the benefits of integrating SDN solutions in operational technology networks, ultimately contributing to the advancement of secure and resilient network infrastructures.

The motivation for this paper comes from two major sources. The first one is related to the rapid evolution of SDN into critical infrastructure systems, which underscores the need for advanced mechanisms to ensure their security, reliability, and adaptive capabilities. This paper addresses this need by exploring the detection of incidents and anomalies in SDN-based implementations, highlighting adaptive system changes in response to these challenges. The second one is related to the fact that authors take active part in the project CRINET, Critical Network SDN Security System where the main challenge was to test and use SDN solutions for distributed OT systems where communication between subsystems is done via IT networks thus convergence and packets encapsulation is done.

THE RELATED WORKS

Security in IT solutions has a strategic role in ensuring the integrity, confidentiality and availability of data, which is essential for the operational continuity and resilience of IT systems [34]. Additionally, a critical aspect in information systems is the protection and access to the information they contain [33]. The growing interest in SDN [35], industrial internet of things (IIoT) [36], internet of things (IoT) [37], intrusion detection system (IDS) and the MQTT protocol in IoT environments has resulted in a large amount of research aimed at improving network security and performance [38]. This section reviews the existing literature on SDN applications in IoT, focusing on anomaly detection, lightweight messaging protocols and machine learning-based intrusion detection systems, highlighting advances and challenges in these areas. Examples of practical applications of the analyzed solutions are also described.

The paper [39] presents an intelligent lightweight scheme for detecting LR-DDoS attacks in software-defined IoT environments, based on the MQTT protocol. The scheme uses four machine learning models to analyse a state-of-the-art LRDDoS-MQTT-2022 dataset and achieves high detection accuracy, with the best results obtained by a decision tree classifier (DTC) with 99.5% accuracy. The research [40] also proposed a learning-based detection approach that implements learning algorithms and uses Openflow packets to identify attack traffic in the SDN control and data planes. The proposed approach and experimental results show that the system accurately identified low-speed DDoS attack traffic with low imposition on performance of the system.

The article [41] describes a new denial of service (DoS) attack on the MQTT protocol, called ‘Slow Subscribers,’ which can turn MQTT servers into single points of failure, and presents solutions to increase resilience to such attacks. The attack has been shown to be effective in disrupting MQTT servers, causing significant delays and potentially bringing them to a complete halt. The authors proposed methods to detect and prevent such attacks that significantly improve the resilience of servers to such threats.

The analysis [42] presents an SDN-based solution for detecting and mitigating DoS and DDoS attacks in IoT networks, using an entropy approach to detect network traffic anomalies. The authors conducted experiments in three different scenarios, demonstrating the effectiveness of the proposed method in real IoT traffic conditions. The results show that the entropy-based method can effectively detect and mitigate DoS and DDoS attacks, minimising the impact on legitimate network traffic.

This study [43] discusses how SDN overcomes many shortcomings of wireless sensor networks by proposing an SDN-IoT framework where sensor nodes work within an SDN environment, sending their transmission data to the SDN controller using a Mosquitto broker. The data is analyzed by a Python application to compute packet rate and bandwidth, identifying potential DDOS attacks if these metrics exceed thresholds, with plans to integrate Machine Learning for enhanced DDOS detection in the future.

Attack-resistant IoT systems may have practical utility. The study [44] presents a communication system for battlefield UAV (unmanned aerial vehicle) swarms based on SDN and the MQTT protocol. This practical application aims to enhance the efficiency and reliability of communication in the dynamic and demanding conditions of the battlefield. The researchers implemented the system by integrating SDN controllers to manage network resources dynamically and using the MQTT protocol to facilitate real-time, low-latency communication between UAVs. Additionally, they proposed a QoS-based (Quality of Service) multi-path routing framework, which calculates multiple disjointed paths from sources to destinations to enhance network performance. Another practical example is paper [4] which proposes an SDN-controlled MANET (Mobile Ad Hoc Network) swarm for mobile monitoring, utilizing Raspberry Pi-equipped MANET nodes integrated with cameras and sensors, networked through ad hoc protocols, and managed via centralized SDN control to achieve flexible, robust, and efficient monitoring, with experimental results proving the feasibility of this architecture.

Experiment set-up

For research and testing, an integrated system of diverse technological instances based on the Linux platform was developed. A POX controller and OVS (Open vSwitch) switches were utilized for this purpose. Two research testing stands were set up as part of the Internet of Everything research station within the Department of Complex Systems at Rzeszow University of Technology [14]. The architecture of the first testing stand, used for ICMP traffic research, is illustrated in Fig. 1, and the devices along with their configuration parameters are detailed in Table 1. The second testing stand, intended for

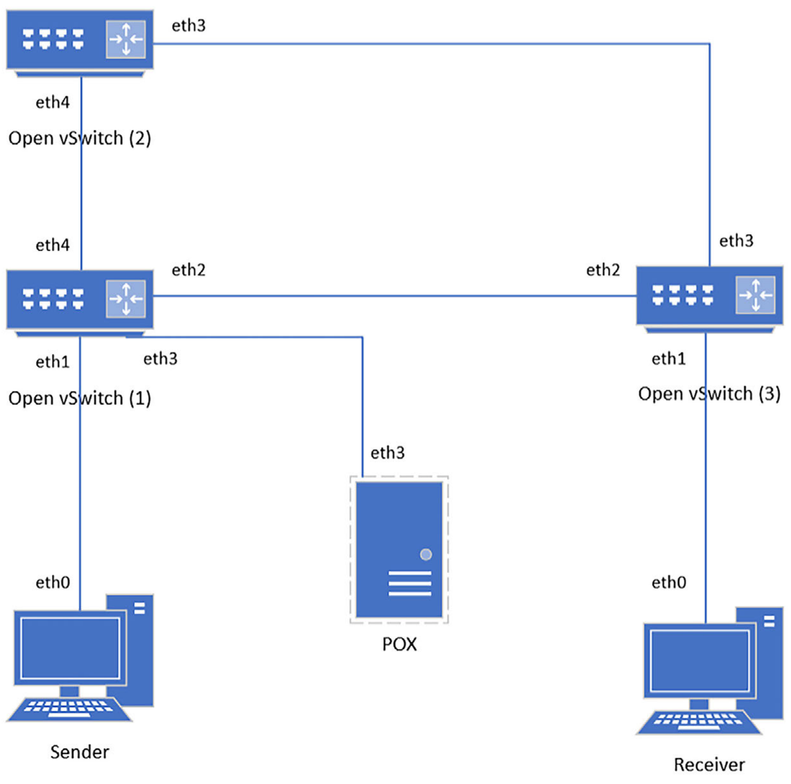


Figure 1. Testing stand for ICMP traffic experiments

Table 1. List and configuration parameters of devices

Device	Ports and adresses
POX	eth3 192.168.0.100
Open vSwitch (1)	eth3 192.168.0.150
Open vSwitch (2)	-
Open vSwitch (3)	-
Sender (PC)	eth0 192.168.0.2
Receiver (PC)	eth0 192.168.0.3

Message Queue Telemetry Transport (MQTT) traffic research, is depicted in Fig. 2, with device configurations listed in Table 2. To ensure isolation and optimize resource management, components such as POX and the MQTT broker were executed in Docker containers.

The test environment comprised both hardware and software, facilitating the comprehensive design and testing of networks utilizing the OpenFlow protocol within the SDN architecture. This adaptable framework allows for dynamic network traffic management, data flow programming, and adjustment to changing requirements [9, 10, 11]. The infrastructure also includes three instances of Open vSwitch (OVS), a versatile software switch for SDN environments, functioning as a multi-functional network bridge supporting the OpenFlow protocol for programmable traffic control. According to official documentation, POX supports OpenFlow 1.0 and offers specific support for Open vSwitch/Nicira extensions [15]. The framework requires Python version 3 and is available on GitHub for download via the git tool [16].

The initial phase of research concentrated on analyzing irregularities in ICMP packet transport and evaluating the responses to these issues (scenarios 1 and 2). Devices used for testing are described in Table 1, and their topology is shown in Figure 1.

Additional infrastructure components (Table1) include Linux-based computers and Open vSwitch network switches. The computers, named ‘Sender’ and ‘Receiver’, run on Ubuntu 20.04 and are equipped with tools for generating ICMP traffic and testing host communication. Open vSwitch switches can be managed directly through commands or Python scripts using the POX environment. In this study, the switches were configured, with one instance connected to the POX controller. Not all switches needed to be connected to the controller, as they operate in automatic mode after configuration, forwarding packets correctly between interfaces. Subsequent research focused on MQTT traffic (scenarios #3 and #4). For this purpose, the

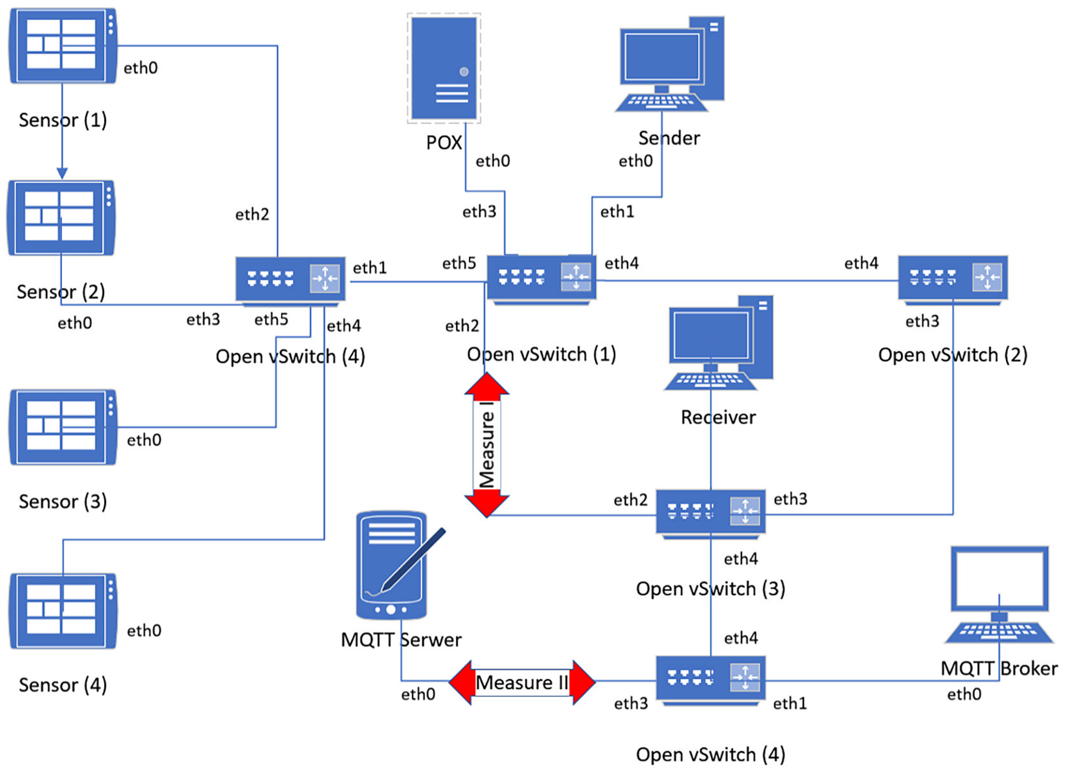


Figure 2. MQTT traffic testing stand

infrastructure was expanded with additional elements (Table 2), and their topologies are shown in Fig. 2. The test setup was augmented with instances essential for MQTT communication:

- broker - intermediary for communication between various MQTT clients,
- clients - four instances acting as sensors, publishing messages,
- subscriber - the unit receiving messages from the broker, marked as MQTT Server.

To verify the communication and functionality of the infrastructure, PING and Wireshark tools were employed. Measurement points where these tools were used are indicated in Fig. 2 as Measurement I and Measurement II.

Table 2. List and configuration parameters of MQTT traffic testing devices

Devices	Ports and adresses
POX	eth0 192.168.0.100
Open vSwitch (1)	eth3 192.168.0.150
Open vSwitch (2)	-
Open vSwitch (3)	-
Open vSwitch (4)	-
Sender (PC)	eth0 192.168.0.2
Receiver (PC)	eth0 192.168.0.3
MQTT Broker	eth0 192.168.0.5
MQTT Server (Subscriber)	eth0 192.168.0.10
MQTT Sensor (1)	eth0 192.168.0.11
MQTT Sensor (2)	eth0 192.168.0.12
MQTT Sensor (3)	eth0 192.168.0.13
MQTT Sensor (4)	eth0 192.168.0.14

Preparation and configuration of a multiplatform POX instance

POX, written in Python, was utilized as the SDN controller, necessitating the preparation of a system instance with the controller software. POX offers several stable versions, including angler, betta, carp, dart, eel, fangtooth and halosaur. The test environment employed a Docker image prepared based on the program detailed in [17]. This image is built and launched using the script provided in [18].

OVS configuration from the CLI level

Open vSwitch facilitates network management at layers 2 and 3 directly through the console, offering extensive functionality and supporting multiple protocols. The primary tool for managing Open vSwitch from the console is the *ovs-vsctl* command. This command enables switch configuration, adding ports, defining bridges, setting port attributes, and establishing connections to the SDN controller. The functions and commands are detailed in Table 3.

Controlling OVS with POX in Python

POX is an excellent solution for continuously controlling Open vSwitch switches according to programmed rules. Certain basic POX methods are common to all introduced processes. The minimal runtime code that manages OVS devices requires creating a file with the base controller class. A sample of this code is shown in [19].

The _handle_PacketIn function in the POX controller is triggered when the controller receives a packet from the switch, reported by the PacketIn event. This function is crucial for network traffic processing in an SDN environment. The _handle_ConnectionUp method is invoked when a connection is established. This method is important because it enables the controller to react to the connection establishment, allowing for actions such as initialization, retrieving the switch's state information, or assigning specific resources to it.

Preparing the infrastructure to support the MQTT protocol

Measurement data sent remotely from sensors is known as telemetry data. Devices connected to the IoT network transmit information to a central system, where it is collected and processed. OT/IoT devices typically have limited computing power, making lightweight protocols like MQTT ideal [12, 13]. MQTT is designed to provide high data throughput in near real-time with minimal resource usage. It operates on a publish-subscribe communication model, with a connection broker service managing links and communication channels (topics). The topic is an identifier indicating which publication or subscription channel a given message is assigned to, and the broker acts as an intermediary between the sender and recipient, as illustrated in Fig. 3. For advanced communication or security requirements, the Advanced Message Queuing Protocol (AMQP) can be considered as an alternative [12, 13].

Data transmitted from IoT devices is processed by central systems (Fig. 3). Any network-connected device that implements the TCP/IP stack and supports the MQTT protocol can function as an

Table 3. Description of subsequent functions used to configure OVS

1	ovs-vsctl del-br br0	Removes the bridge named „br0" (if it exists).
2	ovs-vsctl del-br br1	Removes the bridge named „br1" (if it exists).
3	ovs-vsctl del-br br2	Removes the bridge named „br2" (if it exists).
4	ovs-vsctl del-br br3	Removes the bridge named „br3" (if it exists).
5	ovs-vsctl del-br mybridge	Removes the bridge named „mybridge" (if it exists).
6	ovs-vsctl add-br mybridge	Adds a new bridge called „mybridge".
7	ovs-vsctl add-port mybridge eth1 -- set Interface eth1 ofport_request=1	Adds a port named „eth1" to the bridge „mybridge" and sets the port number to 1.
8	ovs-vsctl add-port mybridge eth2 -- set Interface eth2 ofport_request=2	Adds a port named „eth2" to the bridge „mybridge" and sets the port number to 2.
9	ovs-vsctl add-port mybridge eth4 -- set Interface eth4 ofport_request=4	Adds a port named „eth4" to the bridge „mybridge" and sets the port number to 4.
10	ovs-ofctl add-flow mybridge in_port=1,actions=output:2	Adds a flow rule that says that if a packet enters through port 1, it will be sent to port 2.
11	ovs-ofctl add-flow mybridge in_port=2,actions=output:1	Adds a flow rule that says that if a packet enters through port 2, it will be sent to port 1.
12	ovs-vsctl set-controller mybridge tcp:192.168.0.100:6633	Sets the OpenFlow controller for the „mybridge" bridge to the IP address 192.168.0.100 and port 6633. These settings result from the configuration and address of the POX itself.
13	ovs-vsctl show	Displays details about all bridges managed by ovs-vswitchd along with controller data

Note: the OVS devices used in the research have port connections prepared by default.

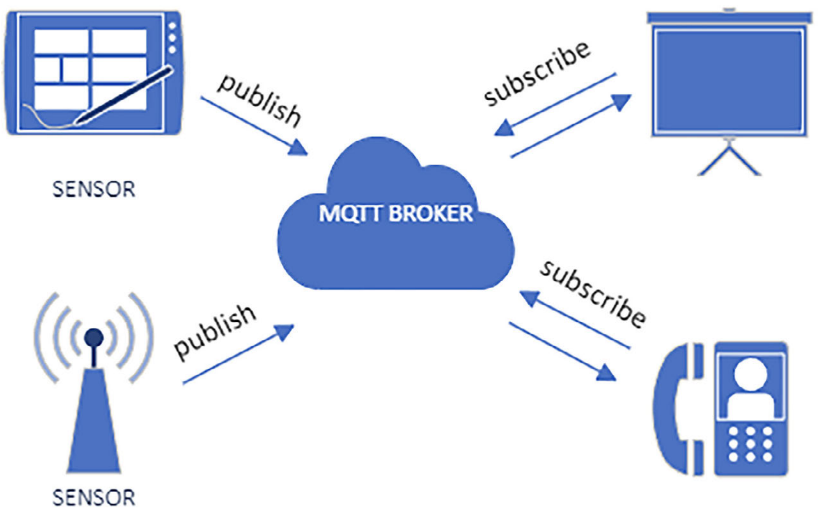


Figure 3. MQTT protocol operation diagram

MQTT client, capable of acting both as a subscriber and a sender, facilitating two-way communication. The Docker tool was utilized to implement selected infrastructure components for the MQTT communication testing environment. The port and access settings are specified in the "mosquitto.conf" file. The configuration file [20] is responsible for creating the Eclipse Mosquitto image, an open-source software that provides an MQTT Broker instance to mediate communication. The flow of packets through the proposed infrastructure was verified. All clients' packets reach the MQTT gateway, while the subscriber communicates solely with this gateway, adhering to the MQTT standard. The script [21] distinguishes MQTT packets from others. Upon receiving a packet, the code checks if it is a TCP packet containing IP data. If the packet is directed to port 1883, designated for the MQTT protocol, the is_mqtt flag is set, and the handle_mqtt function is called to process the MQTT packet. If not, the packet is forwarded to other ports. The entire process is monitored by logging information about the packet type and the size of the transferred MQTT data.

Scenario #1 – ICMP traffic filtering

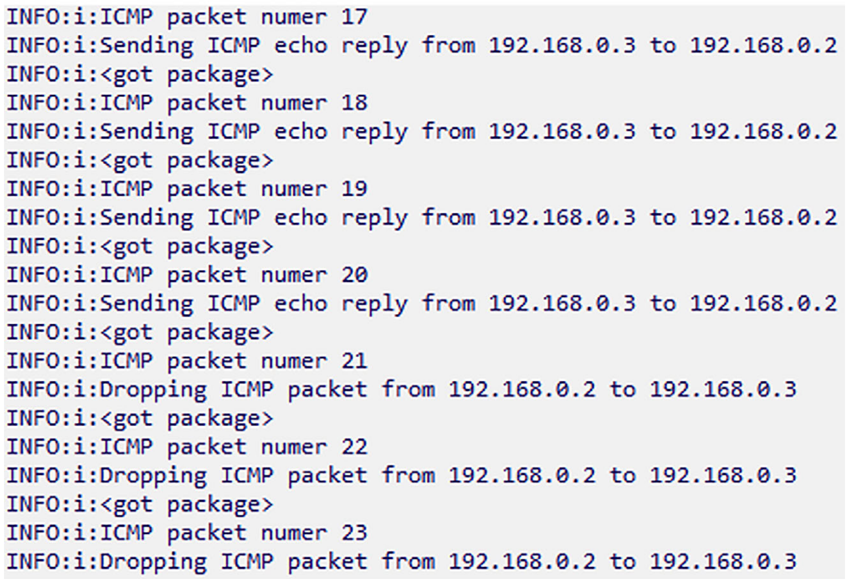
By implementing an instance of POX as an Open vSwitch controller, the capability to accurately analyze various types of packets, including ICMP, was achieved. Understanding anomalies and filtering ICMP traffic, which is used for routing control, host availability testing, and error communication, involves defining rules for data flow in the network. The investigation [22] (test scenario #1) focuses on analyzing packets and determining their types. The _handle_PacketIn function manages the event of receiving a packet from a switch on the OpenFlow controller. When a packet is received, the program logs information about the event. If the packet is of the IP type and contains the ICMP protocol, the program checks the number of received ICMP packets. If this number exceeds 20, it logs that the ICMP packet has been discarded; otherwise, it sends an “Echo Reply” ICMP response to the source IP address. This code segment provides control over the number and type of ICMP packets accepted by the controller, which is crucial for network security. The program’s results are illustrated in Figure 4, showing that packets beyond the 20th iteration are discarded (“Dropping ICMP packet from...”). POX effectively analyzes ICMP packets and can detect and counter related attacks. This controller responds efficiently to potential ICMP-related threats, making POX an essential tool for maintaining network security.

Scenario #2 – MQTT traffic management

The second research scenario involves deterministic operation of sensors in the context of sampling and transporting measured parameters to the target system. It assumes that each sensor provides information at fixed intervals and operates similarly to other sensors. If the number of packets deviates from the expected count, an anomaly is reported. Additionally, in cases of packet redundancy, measurements exceeding the predetermined limit are discarded. The key assumptions for this scenario are as follows:

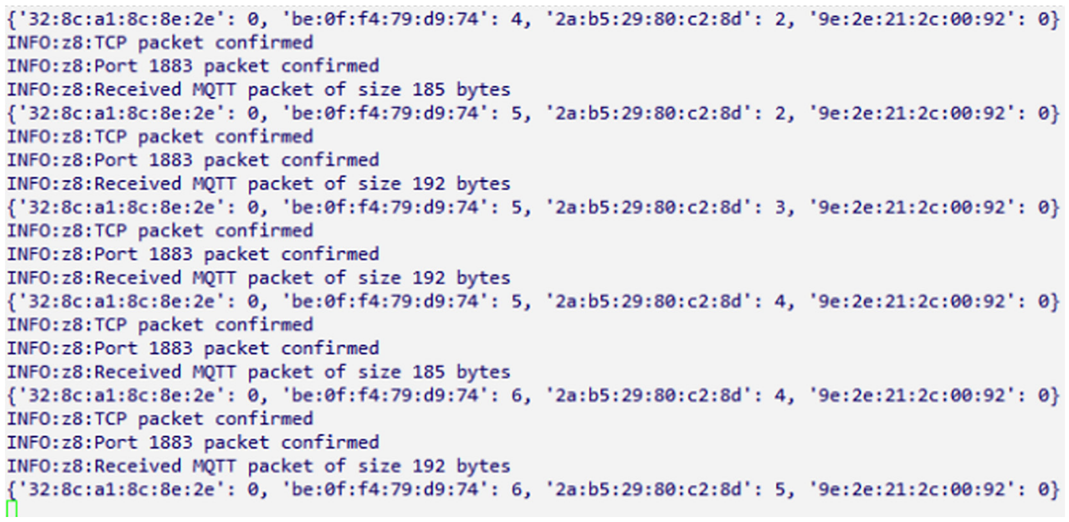
- assumed transmission of 50 to 60 packets in 60 seconds,
- if the number of packets below the accepted limits – console event notification,
- if the number of packets above the accepted limits – event notification and redundant rejection.

Figures 5–7 illustrate the basic operation of the scenario elements. Figure 5 displays the process of counting individual packets without using the anomaly detection mechanism. An attempt was made to count packets for 60 seconds with a sending interval of 1 second to evaluate the correctness of the scripts’ operation. Figure 7 shows



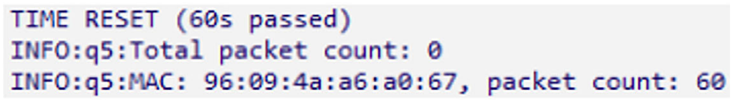
```
INFO:i:ICMP packet numer 17
INFO:i:Sending ICMP echo reply from 192.168.0.3 to 192.168.0.2
INFO:i:<got package>
INFO:i:ICMP packet numer 18
INFO:i:Sending ICMP echo reply from 192.168.0.3 to 192.168.0.2
INFO:i:<got package>
INFO:i:ICMP packet numer 19
INFO:i:Sending ICMP echo reply from 192.168.0.3 to 192.168.0.2
INFO:i:<got package>
INFO:i:ICMP packet numer 20
INFO:i:Sending ICMP echo reply from 192.168.0.3 to 192.168.0.2
INFO:i:<got package>
INFO:i:ICMP packet numer 21
INFO:i:Dropping ICMP packet from 192.168.0.2 to 192.168.0.3
INFO:i:<got package>
INFO:i:ICMP packet numer 22
INFO:i:Dropping ICMP packet from 192.168.0.2 to 192.168.0.3
INFO:i:<got package>
INFO:i:ICMP packet numer 23
INFO:i:Dropping ICMP packet from 192.168.0.2 to 192.168.0.3
```

Figure 4. ICMP is ignored when 20 packets are exceeded



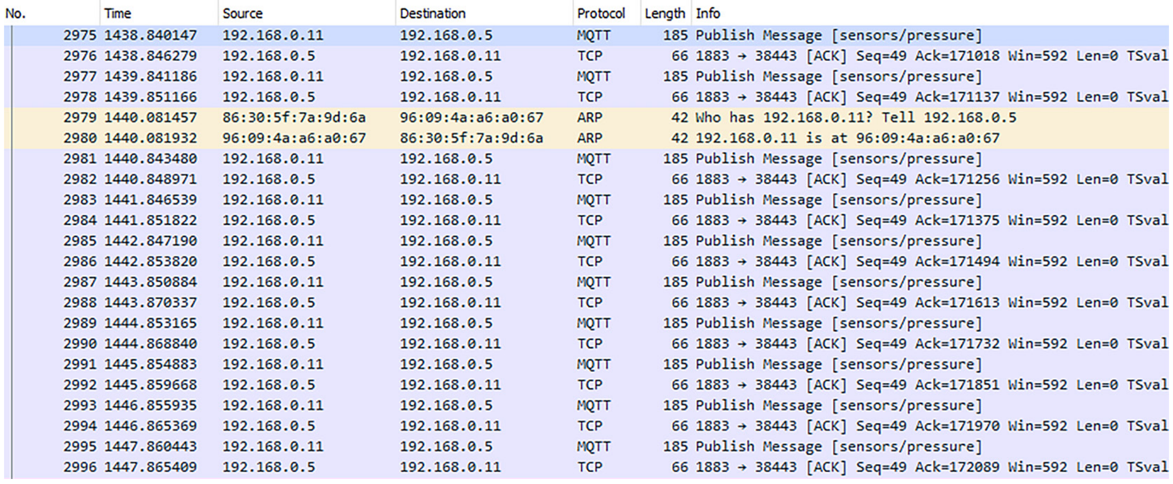
```
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 4, '2a:b5:29:80:c2:8d': 2, '9e:2e:21:2c:00:92': 0}
INFO:z8:TCP packet confirmed
INFO:z8:Port 1883 packet confirmed
INFO:z8:Received MQTT packet of size 185 bytes
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 5, '2a:b5:29:80:c2:8d': 2, '9e:2e:21:2c:00:92': 0}
INFO:z8:TCP packet confirmed
INFO:z8:Port 1883 packet confirmed
INFO:z8:Received MQTT packet of size 192 bytes
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 5, '2a:b5:29:80:c2:8d': 3, '9e:2e:21:2c:00:92': 0}
INFO:z8:TCP packet confirmed
INFO:z8:Port 1883 packet confirmed
INFO:z8:Received MQTT packet of size 192 bytes
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 5, '2a:b5:29:80:c2:8d': 4, '9e:2e:21:2c:00:92': 0}
INFO:z8:TCP packet confirmed
INFO:z8:Port 1883 packet confirmed
INFO:z8:Received MQTT packet of size 185 bytes
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 6, '2a:b5:29:80:c2:8d': 4, '9e:2e:21:2c:00:92': 0}
INFO:z8:TCP packet confirmed
INFO:z8:Port 1883 packet confirmed
INFO:z8:Received MQTT packet of size 192 bytes
{'32:8c:a1:8c:8e:2e': 0, 'be:0f:f4:79:d9:74': 6, '2a:b5:29:80:c2:8d': 5, '9e:2e:21:2c:00:92': 0}
```

Figure 5. An example of a process that counts packet occurrences from different sources



```
TIME RESET (60s passed)
INFO:q5:Total packet count: 0
INFO:q5:MAC: 96:09:4a:a6:a0:67, packet count: 60
```

Figure 6. POX console for a single sensor that sends data every 1 s. After 60 s, the traffic was summarized and no anomalies were reported

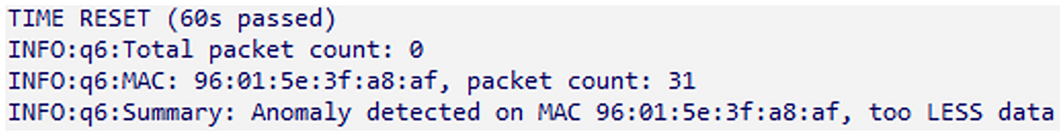


```
No.        Time            Source              Destination         Protocol  Length Info
     2975 1438.840147     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2976 1438.846279     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171018 Win=592 Len=0 TSval
     2977 1439.841186     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2978 1439.851166     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171137 Win=592 Len=0 TSval
     2979 1440.081457     86:30:5f:7a:9d:6a   96:09:4a:a6:a0:67   ARP         42 Who has 192.168.0.11? Tell 192.168.0.5
     2980 1440.081932     96:09:4a:a6:a0:67   86:30:5f:7a:9d:6a   ARP         42 192.168.0.11 is at 96:09:4a:a6:a0:67
     2981 1440.843480     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2982 1440.848971     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171256 Win=592 Len=0 TSval
     2983 1441.846539     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2984 1441.851822     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171375 Win=592 Len=0 TSval
     2985 1442.847190     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2986 1442.853820     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171494 Win=592 Len=0 TSval
     2987 1443.850884     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2988 1443.870337     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171613 Win=592 Len=0 TSval
     2989 1444.853165     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2990 1444.868840     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171732 Win=592 Len=0 TSval
     2991 1445.854883     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2992 1445.859668     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171851 Win=592 Len=0 TSval
     2993 1446.855935     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2994 1446.865369     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=171970 Win=592 Len=0 TSval
     2995 1447.860443     192.168.0.11        192.168.0.5         MQTT       185 Publish Message [sensors/pressure]
     2996 1447.865409     192.168.0.5         192.168.0.11        TCP         66 1883 → 38443 [ACK] Seq=49 Ack=172089 Win=592 Len=0 TSval
```

Figure 7. Transport from 1 sensor data in Wireshark. The ARP packets that were not considered in the count are shown

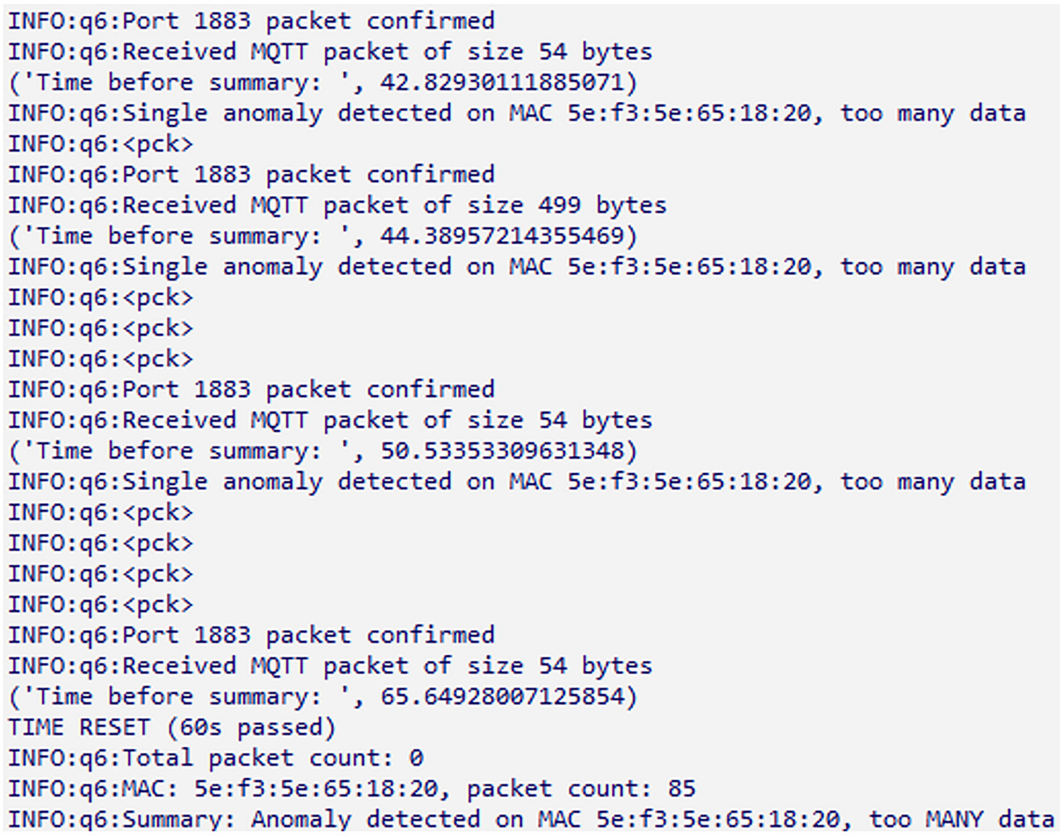
the process of counting packets from a single source in Wireshark. In the next step, a data transport model was used where the sensor sends data every 2 seconds (Fig. 8). After 60 seconds, an anomaly was detected with the message "Anomaly detected on MAC … too LESS data." Another test considered the case where the sensor sends data continuously, and the program reported an anomaly in both cases, as shown in Figure 9 with the message "Single anomaly detected on MAC."

In the codes referenced by [23–25] for network control, packets are handled based on their type, with only MQTT packets being counted. When an anomaly occurs, such as the number of packets from a particular sensor exceeding 60 or dropping below 50 in 60 seconds, the program generates an anomaly report and stops receiving excess data. The code includes mechanisms for monitoring and reporting anomalies every 60 seconds and flow control for specific MQTT packets,



```
TIME RESET (60s passed)
INFO:q6:Total packet count: 0
INFO:q6:MAC: 96:01:5e:3f:a8:af, packet count: 31
INFO:q6:Summary: Anomaly detected on MAC 96:01:5e:3f:a8:af, too LESS data
```

Figure 8. Example for a sensor that sends data at an interval of 2 seconds



```
INFO:q6:Port 1883 packet confirmed
INFO:q6:Received MQTT packet of size 54 bytes
('Time before summary: ', 42.82930111885071)
INFO:q6:Single anomaly detected on MAC 5e:f3:5e:65:18:20, too many data
INFO:q6:<pck>
INFO:q6:Port 1883 packet confirmed
INFO:q6:Received MQTT packet of size 499 bytes
('Time before summary: ', 44.38957214355469)
INFO:q6:Single anomaly detected on MAC 5e:f3:5e:65:18:20, too many data
INFO:q6:<pck>
INFO:q6:<pck>
INFO:q6:<pck>
INFO:q6:Port 1883 packet confirmed
INFO:q6:Received MQTT packet of size 54 bytes
('Time before summary: ', 50.53353309631348)
INFO:q6:Single anomaly detected on MAC 5e:f3:5e:65:18:20, too many data
INFO:q6:<pck>
INFO:q6:<pck>
INFO:q6:<pck>
INFO:q6:<pck>
INFO:q6:Port 1883 packet confirmed
INFO:q6:Received MQTT packet of size 54 bytes
('Time before summary: ', 65.64928007125854)
TIME RESET (60s passed)
INFO:q6:Total packet count: 0
INFO:q6:MAC: 5e:f3:5e:65:18:20, packet count: 85
INFO:q6:Summary: Anomaly detected on MAC 5e:f3:5e:65:18:20, too MANY data
```

Figure 9. The program reports exceeding the target number of packets for 60 s

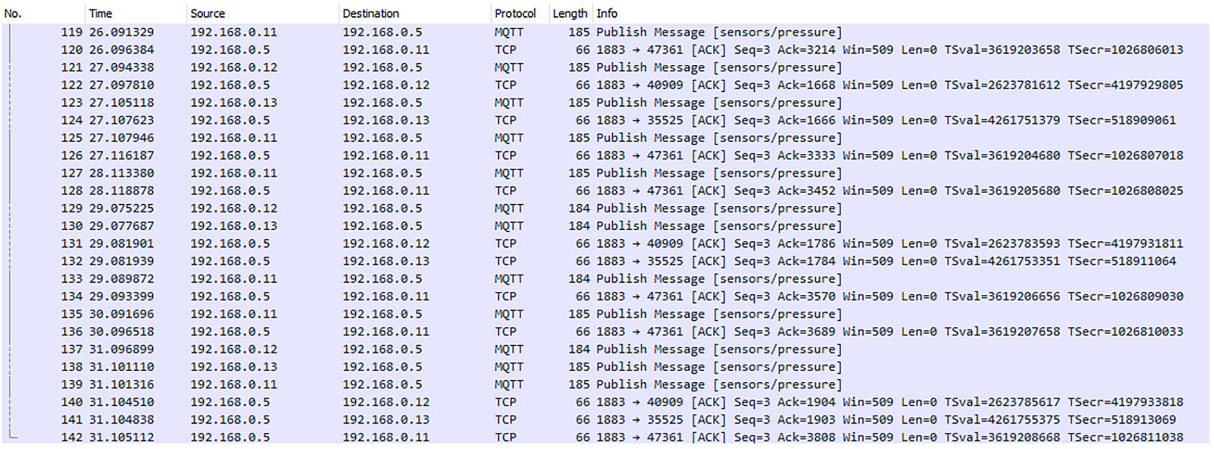
counting them with time and quantity limits. The state of network traffic for multiple sensors with different operating intervals is shown in Figure 10. In contrast, the combined use case for all scenario elements is shown in Figure 11, where two anomalies occurred, indicating that the amount of data delivered was too low ("Anomaly detected on MAC … too LESS data").

To sum up, in scenario #2, Python was used within the POX framework to monitor the number of packets from sensors. The system was configured to expect 50 to 60 packets to be transmitted within 60 seconds from each sensor. If these limits were exceeded or not met, the script logged the anomaly in the console and discarded excess measurements. This approach enabled the detection and response to any incidents in the network that deviated from the expected sensor operation standards.

Scenario #3 – traffic filtering based on the MAC address match of the packet content and header

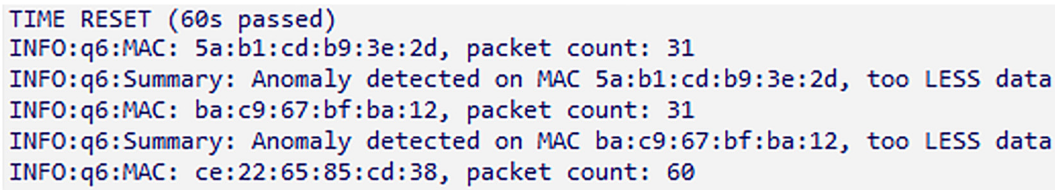
Scenario #3 involves implementing communication using the MQTT protocol (see Scenario #2), with an additional information component: the sender's MAC address embedded in the packet. This address is represented as an element within a JSON structure [26]. The purpose of including the MAC address in the packet body is to control the packet's source during processing. Within the POX environment in Python, it is verified whether the MAC address in the JSON packet content matches the sender's address in the MQTT packet header.

This procedure aims to prevent unauthorized transmission of MQTT packets by ensuring consistency between the sender's MAC address in the JSON content and the packet header [27]. This



No.	Time	Source	Destination	Protocol	Length	Info
119	26.091329	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
120	26.096384	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3214 Win=509 Len=0 TSval=3619203658 TSecr=1026806013
121	27.094338	192.168.0.12	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
122	27.097810	192.168.0.5	192.168.0.12	TCP	66	1883 → 40909 [ACK] Seq=3 Ack=1668 Win=509 Len=0 TSval=2623781612 TSecr=4197929805
123	27.105118	192.168.0.13	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
124	27.107623	192.168.0.5	192.168.0.13	TCP	66	1883 → 35525 [ACK] Seq=3 Ack=1666 Win=509 Len=0 TSval=4261751379 TSecr=518909061
125	27.107946	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
126	27.116187	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3333 Win=509 Len=0 TSval=3619204680 TSecr=1026807018
127	28.113380	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
128	28.118878	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3452 Win=509 Len=0 TSval=3619205680 TSecr=1026808025
129	29.075225	192.168.0.12	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
130	29.077687	192.168.0.13	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
131	29.081901	192.168.0.5	192.168.0.12	TCP	66	1883 → 40909 [ACK] Seq=3 Ack=1786 Win=509 Len=0 TSval=2623783593 TSecr=4197931811
132	29.081939	192.168.0.5	192.168.0.13	TCP	66	1883 → 35525 [ACK] Seq=3 Ack=1784 Win=509 Len=0 TSval=4261753351 TSecr=518911064
133	29.089872	192.168.0.11	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
134	29.093399	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3570 Win=509 Len=0 TSval=3619206656 TSecr=1026809030
135	30.091696	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
136	30.096518	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3689 Win=509 Len=0 TSval=3619207658 TSecr=1026810033
137	31.096899	192.168.0.12	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
138	31.101110	192.168.0.13	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
139	31.101316	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
140	31.104510	192.168.0.5	192.168.0.12	TCP	66	1883 → 40909 [ACK] Seq=3 Ack=1904 Win=509 Len=0 TSval=2623785617 TSecr=4197933818
141	31.104838	192.168.0.5	192.168.0.13	TCP	66	1883 → 35525 [ACK] Seq=3 Ack=1903 Win=509 Len=0 TSval=4261755375 TSecr=518913069
142	31.105112	192.168.0.5	192.168.0.11	TCP	66	1883 → 47361 [ACK] Seq=3 Ack=3808 Win=509 Len=0 TSval=3619208668 TSecr=1026811038

Figure 10. Movement of several sensors in Wireshark



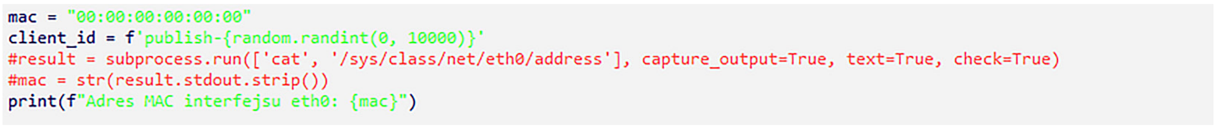
```
TIME RESET (60s passed)
INFO:q6:MAC: 5a:b1:cd:b9:3e:2d, packet count: 31
INFO:q6:Summary: Anomaly detected on MAC 5a:b1:cd:b9:3e:2d, too LESS data
INFO:q6:MAC: ba:c9:67:bf:ba:12, packet count: 31
INFO:q6:Summary: Anomaly detected on MAC ba:c9:67:bf:ba:12, too LESS data
INFO:q6:MAC: ce:22:65:85:cd:38, packet count: 60
```

Figure 11. Detection of anomalies indicative of too few packets for 2 out of 3 connected sensors transmitted within 60 seconds

additional control layer secures the integrity and authenticity of transmitted data within an MQTT-based communication system. If a MAC address mismatch is detected, an incident is reported in the POX console, and the packet is not forwarded. The system, detailed in [28], parses MQTT packets to check if the sender's MAC address in the header matches the one in the packet body. This data integrity control mechanism stops further packet processing if the addresses do not match. Error handling ensures that the code checks for the presence of the "mac" key to avoid blocking control packets that do not transport the intended data, such as ARP packets [27].

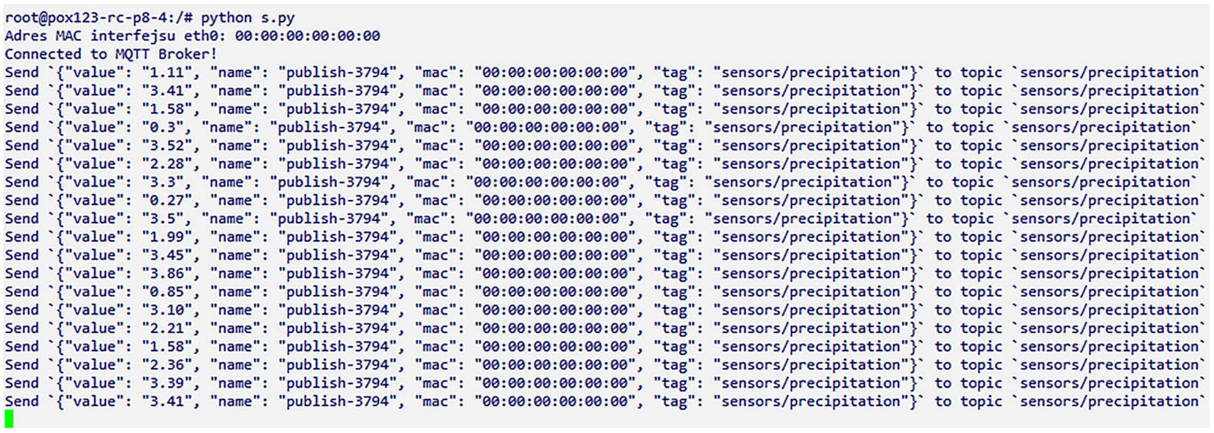
Figure 12 illustrates a modified method for determining the physical address of one sensor. After applying the change, this device sends incorrectly prepared data packets, with the address inside the packet differing from the one in the packet header. To confirm the program's functionality, sensors were activated, including one with a mismatched physical address (Figures 13 and 14). Figure 15 shows the SDN controller console receiving MQTT packets and confirming whether the sender's MAC address is present in the packet body. The program reports results for both matching and mismatching cases.

To summarize, in scenario #3, MQTT-based communication was implemented with an additional information component in the form of the sender's MAC address, embedded in the packet content as a JSON element. This allows for source control during packet processing. To prevent unauthorized transmission of MQTT packets, the consistency of the sender's MAC address is verified at two points in the packet structure: the JSON content and the MQTT packet header. When processing MQTT packets, a program on the POX controller (in Python) retrieves the source MAC address from the packet header, parses the JSON structure, deserializes the packet body, retrieves



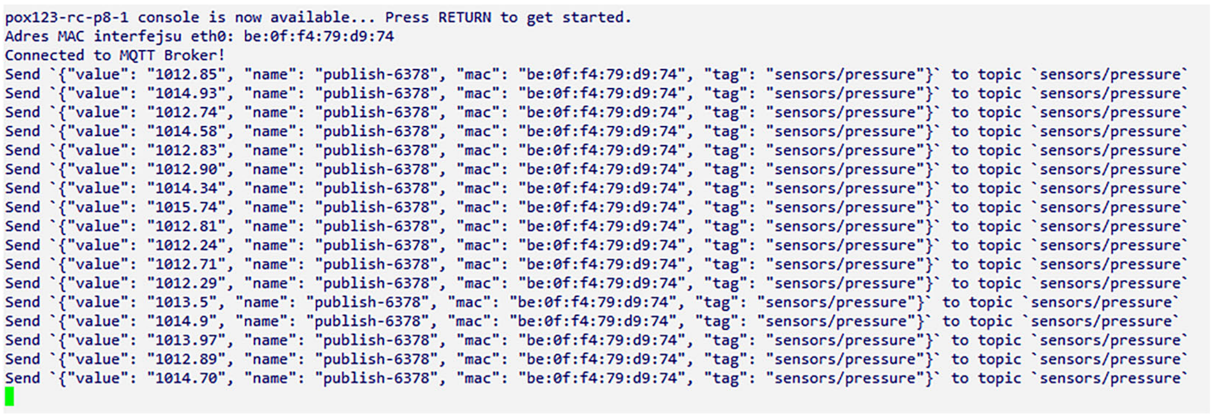
```python
mac = "00:00:00:00:00:00"
client_id = f'publish-{random.randint(0, 10000)}'
#result = subprocess.run(['cat', '/sys/class/net/eth0/address'], capture_output=True, text=True, check=True)
#mac = str(result.stdout.strip())
print(f"Adres MAC interfejsu eth0: {mac}")
```

Figure 12. Changing the sent MAC address from the client level



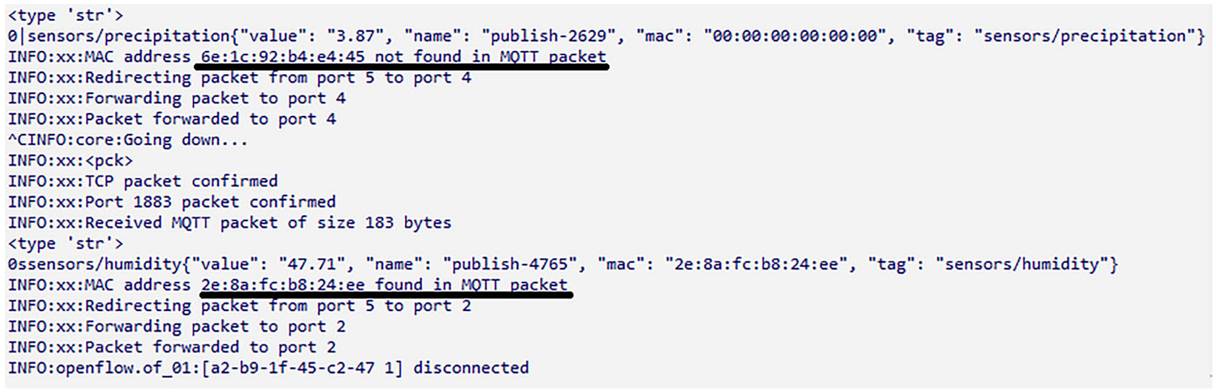
```
root@pox123-rc-p8-4:/# python s.py
Adres MAC interfejsu eth0: 00:00:00:00:00:00
Connected to MQTT Broker!
Send `{"value": "1.11", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.41", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "1.58", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "0.3", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.52", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "2.28", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.3", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "0.27", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.5", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "1.99", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.45", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.86", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "0.85", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.10", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "2.21", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "1.58", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "2.36", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.39", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
Send `{"value": "3.41", "name": "publish-3794", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}` to topic `sensors/precipitation`
```

Figure 13. Sensor console with a crafted physical address in the packet body



```
pox123-rc-p8-1 console is now available... Press RETURN to get started.
Adres MAC interfejsu eth0: be:0f:f4:79:d9:74
Connected to MQTT Broker!
Send `{"value": "1012.85", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1014.93", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.74", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1014.58", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.83", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.90", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1014.34", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1015.74", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.81", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.24", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.71", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.29", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1013.5", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1014.9", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1013.97", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1012.89", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
Send `{"value": "1014.70", "name": "publish-6378", "mac": "be:0f:f4:79:d9:74", "tag": "sensors/pressure"}` to topic `sensors/pressure`
```

Figure 14. Sensor console with the correct physical address in the packet body



```
<type 'str'>
0|sensors/precipitation{"value": "3.87", "name": "publish-2629", "mac": "00:00:00:00:00:00", "tag": "sensors/precipitation"}
INFO:xx:MAC address 6e:1c:92:b4:e4:45 not found in MQTT packet
INFO:xx:Redirecting packet from port 5 to port 4
INFO:xx:Forwarding packet to port 4
INFO:xx:Packet forwarded to port 4
^CINFO:core:Going down...
INFO:xx:<pck>
INFO:xx:TCP packet confirmed
INFO:xx:Port 1883 packet confirmed
INFO:xx:Received MQTT packet of size 183 bytes
<type 'str'>
0ssensors/humidity{"value": "47.71", "name": "publish-4765", "mac": "2e:8a:fc:b8:24:ee", "tag": "sensors/humidity"}
INFO:xx:MAC address 2e:8a:fc:b8:24:ee found in MQTT packet
INFO:xx:Redirecting packet from port 5 to port 2
INFO:xx:Forwarding packet to port 2
INFO:xx:Packet forwarded to port 2
INFO:openflow.of_01:[a2-b9-1f-45-c2-47 1] disconnected
```

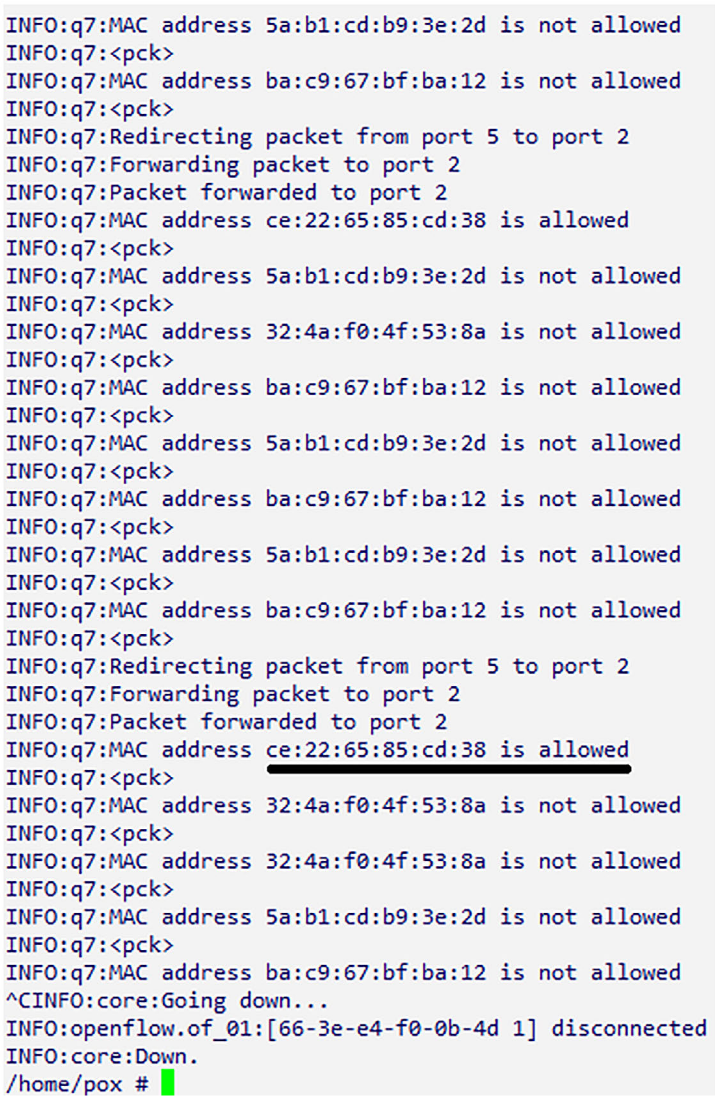
Figure 15. Confirmation and rejection of the MAC address from the content in POX

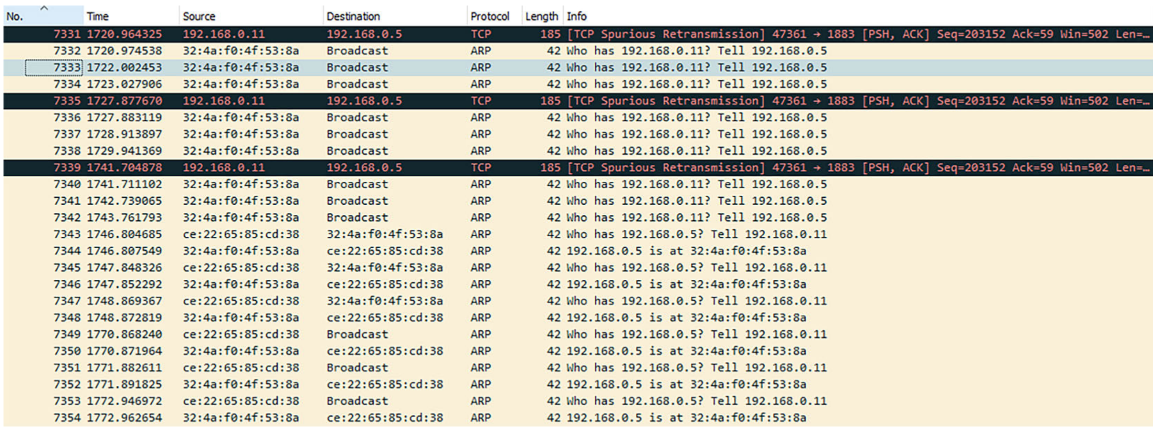
the MAC address from the body, and compares the two MAC addresses. If they do not match, the program reports an incident in the POX console, preventing the packet from being forwarded.

Scenario #4 - filtering traffic not intended for the implemented infrastructure

Data filtration is crucial for effective traffic management in networks supporting telemetry data transmission. For MQTT data transfer, specific legitimate use cases for filters include eliminating interference and securing networks against unauthorized access. Filters help maintain the stability and integrity of transmitted telemetry data by identifying and excluding incorrect or unwanted packets. They also play a vital role in preventing access from unauthorized devices by filtering packets from senders without the appropriate permissions, such as devices with undefined MAC addresses. Additionally, filters optimize network bandwidth by excluding irrelevant packets that do not meet specific criteria.

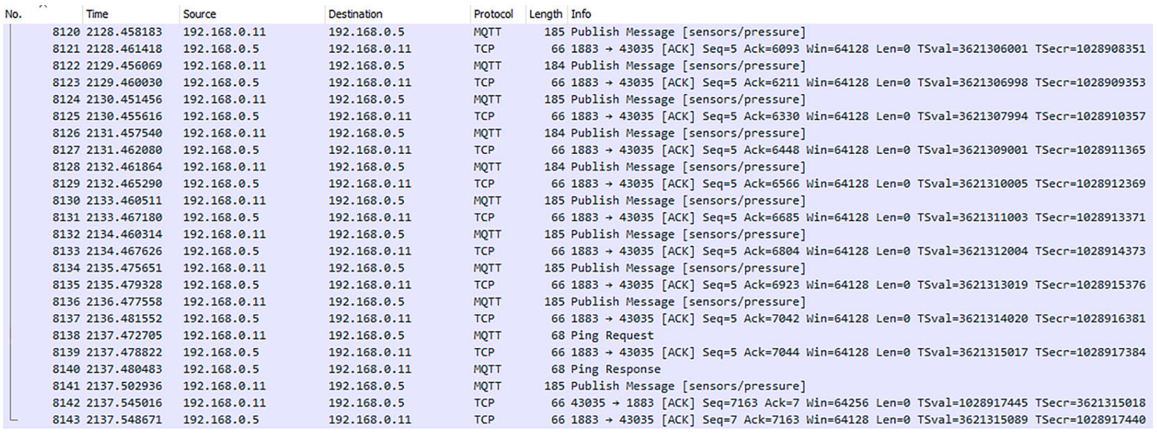
In the assumed scenario, a script was prepared to report anomalies when an unknown packet type or source is detected. A list of allowed physical addresses was defined to permit communication within the network. Data from devices not on this list are reported as rejected. The second part of the scenario involves a mechanism for rejecting non-TCP packets at the transport layer, such as UDP packets, since MQTT uses TCP. In this case, ARP traffic was not blocked. Reference [29] shows a fragment of the PacketIn event handling in the POX controller. The _handle_PacketIn method logs event information and then redirects the packet to the handle_pck method. This second method checks whether the source MAC address is in the allowed list; if so, the packet is forwarded, otherwise, information about an illegal MAC address is logged. Figure 16 illustrates the algorithm's operation. A detailed packet transport analysis using Wireshark (Fig. 17) indicates that all outgoing traffic, except from the indicated MAC address, was blocked. However, communication was one-way, so the transport of MQTT packets was not fully correct. In the next script run, correct bidirectional communication of both MQTT and TCP packets was observed (Fig. 18). Additionally, Wireshark only detected packets from physical addresses on the allowed list [30]. In the next test phase, the controller received various packet types, filtering out those unrelated to the MQTT protocol. UDP packets, incompatible with MQTT, were identified. The code in [31] checks packet types, while the testing code periodically sends UDP packets to the


```
INFO:q7:MAC address 5a:b1:cd:b9:3e:2d is not allowed
INFO:q7:<pck>
INFO:q7:MAC address ba:c9:67:bf:ba:12 is not allowed
INFO:q7:<pck>
INFO:q7:Redirecting packet from port 5 to port 2
INFO:q7:Forwarding packet to port 2
INFO:q7:Packet forwarded to port 2
INFO:q7:MAC address ce:22:65:85:cd:38 is allowed
INFO:q7:<pck>
INFO:q7:MAC address 5a:b1:cd:b9:3e:2d is not allowed
INFO:q7:<pck>
INFO:q7:MAC address 32:4a:f0:4f:53:8a is not allowed
INFO:q7:<pck>
INFO:q7:MAC address ba:c9:67:bf:ba:12 is not allowed
INFO:q7:<pck>
INFO:q7:MAC address 5a:b1:cd:b9:3e:2d is not allowed
INFO:q7:<pck>
INFO:q7:MAC address ba:c9:67:bf:ba:12 is not allowed
INFO:q7:<pck>
INFO:q7:MAC address 5a:b1:cd:b9:3e:2d is not allowed
INFO:q7:<pck>
INFO:q7:MAC address ba:c9:67:bf:ba:12 is not allowed
INFO:q7:<pck>
INFO:q7:Redirecting packet from port 5 to port 2
INFO:q7:Forwarding packet to port 2
INFO:q7:Packet forwarded to port 2
INFO:q7:MAC address ce:22:65:85:cd:38 is allowed
INFO:q7:<pck>
INFO:q7:MAC address 32:4a:f0:4f:53:8a is not allowed
INFO:q7:<pck>
INFO:q7:MAC address 32:4a:f0:4f:53:8a is not allowed
INFO:q7:<pck>
INFO:q7:MAC address 5a:b1:cd:b9:3e:2d is not allowed
INFO:q7:<pck>
INFO:q7:MAC address ba:c9:67:bf:ba:12 is not allowed
^CINFO:core:Going down...
INFO:openflow.of_01:[66-3e-e4-f0-0b-4d 1] disconnected
INFO:core:Down.
/home/pox #
```
Figure 16. The process of traffic separation dependent on the physical address of the packet



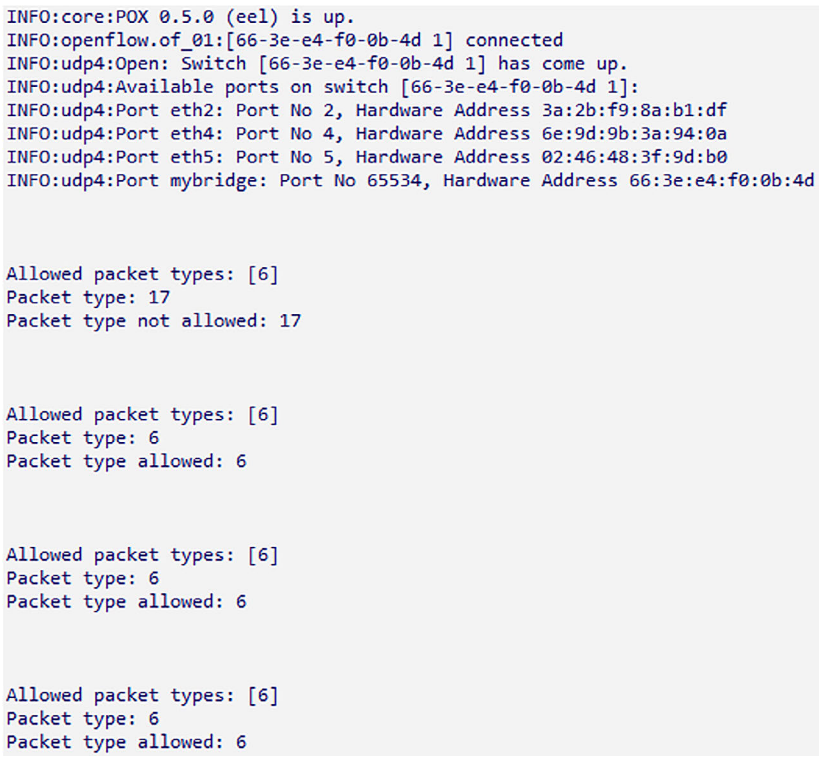
No.	Time	Source	Destination	Protocol	Length	Info
7331	1720.964325	192.168.0.11	192.168.0.5	TCP	185	[TCP Spurious Retransmission] 47361 → 1883 [PSH, ACK] Seq=203152 Ack=59 Win=502 Len=…
7332	1720.974538	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7333	1722.002453	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7334	1723.027906	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7335	1727.877670	192.168.0.11	192.168.0.5	TCP	185	[TCP Spurious Retransmission] 47361 → 1883 [PSH, ACK] Seq=203152 Ack=59 Win=502 Len=…
7336	1727.883119	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7337	1728.913897	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7338	1729.941369	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7339	1741.704878	192.168.0.11	192.168.0.5	TCP	185	[TCP Spurious Retransmission] 47361 → 1883 [PSH, ACK] Seq=203152 Ack=59 Win=502 Len=…
7340	1741.711102	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7341	1742.739065	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7342	1743.761793	32:4a:f0:4f:53:8a	Broadcast	ARP	42	Who has 192.168.0.11? Tell 192.168.0.5
7343	1746.804685	ce:22:65:85:cd:38	32:4a:f0:4f:53:8a	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7344	1746.807549	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a
7345	1747.848326	ce:22:65:85:cd:38	32:4a:f0:4f:53:8a	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7346	1747.852292	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a
7347	1748.869367	ce:22:65:85:cd:38	32:4a:f0:4f:53:8a	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7348	1748.872819	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a
7349	1770.868240	ce:22:65:85:cd:38	Broadcast	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7350	1770.871964	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a
7351	1771.882611	ce:22:65:85:cd:38	Broadcast	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7352	1771.891825	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a
7353	1772.946972	ce:22:65:85:cd:38	Broadcast	ARP	42	Who has 192.168.0.5? Tell 192.168.0.11
7354	1772.962654	32:4a:f0:4f:53:8a	ce:22:65:85:cd:38	ARP	42	192.168.0.5 is at 32:4a:f0:4f:53:8a

Figure 17. Blocking communication and devices recognition in the network



No.	Time	Source	Destination	Protocol	Length	Info
8120	2128.458183	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8121	2128.461418	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6093 Win=64128 Len=0 TSval=3621306001 TSecr=1028908351
8122	2129.456069	192.168.0.11	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
8123	2129.460030	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6211 Win=64128 Len=0 TSval=3621306998 TSecr=1028909353
8124	2130.451456	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8125	2130.455616	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6330 Win=64128 Len=0 TSval=3621307994 TSecr=1028910357
8126	2131.457540	192.168.0.11	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
8127	2131.462080	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6448 Win=64128 Len=0 TSval=3621309001 TSecr=1028911365
8128	2132.461864	192.168.0.11	192.168.0.5	MQTT	184	Publish Message [sensors/pressure]
8129	2132.465290	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6566 Win=64128 Len=0 TSval=3621310005 TSecr=1028912369
8130	2133.460511	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8131	2133.467180	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6685 Win=64128 Len=0 TSval=3621311003 TSecr=1028913371
8132	2134.460314	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8133	2134.467626	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6804 Win=64128 Len=0 TSval=3621312004 TSecr=1028914373
8134	2135.475651	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8135	2135.479328	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=6923 Win=64128 Len=0 TSval=3621313019 TSecr=1028915376
8136	2136.477558	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8137	2136.481552	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=7042 Win=64128 Len=0 TSval=3621314020 TSecr=1028916381
8138	2137.472705	192.168.0.11	192.168.0.5	MQTT	68	Ping Request
8139	2137.478822	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=5 Ack=7044 Win=64128 Len=0 TSval=3621315017 TSecr=1028917384
8140	2137.480483	192.168.0.5	192.168.0.11	MQTT	68	Ping Response
8141	2137.502936	192.168.0.11	192.168.0.5	MQTT	185	Publish Message [sensors/pressure]
8142	2137.545016	192.168.0.11	192.168.0.5	TCP	66	43035 → 1883 [ACK] Seq=7163 Ack=7 Win=64256 Len=0 TSval=1028917445 TSecr=3621315018
8143	2137.548671	192.168.0.5	192.168.0.11	TCP	66	1883 → 43035 [ACK] Seq=7 Ack=7163 Win=64128 Len=0 TSval=3621315089 TSecr=1028917440

Figure 18. Wireshark detects correct communication of devices from the list



```
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[66-3e-e4-f0-0b-4d 1] connected
INFO:udp4:Open: Switch [66-3e-e4-f0-0b-4d 1] has come up.
INFO:udp4:Available ports on switch [66-3e-e4-f0-0b-4d 1]:
INFO:udp4:Port eth2: Port No 2, Hardware Address 3a:2b:f9:8a:b1:df
INFO:udp4:Port eth4: Port No 4, Hardware Address 6e:9d:9b:3a:94:0a
INFO:udp4:Port eth5: Port No 5, Hardware Address 02:46:48:3f:9d:b0
INFO:udp4:Port mybridge: Port No 65534, Hardware Address 66:3e:e4:f0:0b:4d



Allowed packet types: [6]
Packet type: 17
Packet type not allowed: 17



Allowed packet types: [6]
Packet type: 6
Packet type allowed: 6



Allowed packet types: [6]
Packet type: 6
Packet type allowed: 6



Allowed packet types: [6]
Packet type: 6
Packet type allowed: 6
```

Figure 19. While listening, POX rejects UDP packets (17) and accepts TCP packets (6)

MQTT Broker device [32]. This script sends UDP packets to a specific port and IP address using a UDP socket to transmit 'UDP' messages every second to the MQTT proxy address (192.168.0.5) and default MQTT port (1883). Each send operation closes the socket and displays "Udp sent" on the console. The infinite while loop ensures the script sends UDP packets continuously, with a one-second interval between sends. Figure 19 shows the Python code activity in the POX controller, identifying and reporting packets on the console. UDP packets sent by [32] are reported as disallowed. As a result of applying the filters in scenario #4, the system successfully filtered out non-MQTT traffic. UDP packets were considered undesirable because MQTT relies on TCP. Additionally, packets from unregistered physical addresses were excluded by the SDN controller. This solution introduced an extra layer of security by eliminating unauthorized traffic sources in the context of telemetry data.

CONCLUSIONS

In this paper, four proposed scenarios showed that the use of SDN solutions, specifically the POX controller and Open vSwitch (OVS), can effectively resist various anomaly situations that threaten network infrastructure. Despite the simplicity of the network topologies, consisting of no more than five devices, the effectiveness of OVS was evident. The system's responses, presented in the figures, convincingly ensured stability and resilience against multiple anomalies that may occur during operation. This approach is practical for the MQTT protocol used in IoT and OT communications. The results from conducted scenarios underscore the efficacy of SDN solutions in detecting and mitigating anomalies within network infrastructures. Although the network topologies were simple, the principles demonstrated are scalable and flexible, suitable for more complex systems. The real-time anomaly detection capabilities of SDN-based systems are crucial for maintaining the security and reliability of critical infrastructures. This research shows that SDN-based systems can effectively monitor network traffic, identify irregularities, and take appropriate actions to mitigate potential threats.

The practical applications of this approach in IoT and OT environments highlight its relevance and potential for broader adoption. By integrating anomaly detection and adaptive response mechanisms, the proposed SDN structure enhances the overall security posture of the network, which is particularly important for critical infrastructure systems. The use of Docker containers for resource isolation and management ensures optimal utilization while maintaining high performance and reliability.

In conclusion, this research provides a robust framework for using SDN solutions to enhance the security and resilience of network infrastructures. The practical applications in IoT and OT environments underscore its potential for wider adoption. Future work could explore scaling these solutions to more complex networks and integrating additional protocols and technologies to further enhance network security and performance.

Acknowledgment

This paper is partially supported by project: Critical Network SDN Security System (CriNet, DOB-SZAFIR 01/A/027/03/2021) co-financed by the National Center for Research and Development as part of the state defense and security program titled "Development of modern, breakthrough technologies for state security and defense".

REFERENCES

1. Sunday U.I., Akhibi S.D. Application of software-defined networking, European Journal of Computer Science and Information Technology, 2022; 10(2): 27–48.
2. Begović M., Čaušević S., Avdagić-Golub E. QoS management in software defined networks for IoT environment: an Overwiev, International Journal for Quality Research 2020; 15(1): 171–188.
3. Imran G.Z.; Alshahrani A., Fayaz A., Alghamdi A., Gwak J. A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges. Electronics 2021; 10(8): 880.
4. Yang L., Anderson T.A., Gopal R., Dantu R. Forwarding and control element separation (ForCES) framework, RFC 3746, https://datatracker.ietf.org/doc/rfc3746/, 2018.
5. Borgianni L., Adami D., Giordano S., Pagano M. Enhancing reliability in rural networks using a software-defined wide area network. Computers 2024; 13(5): 113.
6. Jefia A.O., Popoola S.I., Atayero A.A Software-Defined Networking: Current Trends, Challenges,

and Future Directions, in Proceedings of the International Conference on Industrial Engineering and Operations Management, Washington DC, USA, September 2018; 27–29.

7. Sokappadu B., Hardin A., Mungur A., Armoogum S. Software Defined Networks: Issues and Challenges, 2019 Conference on Next Generation Computing Applications (NextComp), Mauritius, 2019; 1–5.
8. Semong T., Maupong T., Zungeru A.M., Tabona O., Dimakatso S., Boipelo G., Phuthego M. A review on software defined networking as a solution to link failures, Scientific African, 2023; 21: e01865.
9. Rout S., Sahoo K.S., Patra S.S. Energy efficiency in software defined networking: a survey. SN Comput. Sci. 2021; 2: 308.
10. Bahashwan A.A., Anbar M., Manickam S., Al-Amiedy T.A., Aladaileh M.A., Hasbullah I.H. A systematic literature review on machine learning and deep learning approaches for detecting DDoS attacks in software-defined networking. Sensors 2023; 23: 4441.
11. Palka D., Matuszewski A. Software-defined network SDN in CriNet for cyber secure national critical infrastructure, Cybersecurity & Cybercrime, 2024.
12. Malik, S., Ahmad, S., Ullah, I., Park, D.H., Kim, D.H. An adaptive emergency first intelligent scheduling algorithm for e-client task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems. Sustainability 2019; 11: 2192.
13. Pereira, D.A., Ourique de Morais, W., Pignaton de Freitas, E. NoSQL real-time database performance comparison. Int. J. Parallel Emergent Distrib. Syst. 2017; 33: 144–156.
14. https://zsz.prz.edu.pl/en/research-stand-ioe/about
15. https://docs.openvswitch.org/en/latest/topics/ovs-extensions/
16. https://github.com/noxrepo/pox.git
17. https://pastebin.com/CptkTTBp A file that prepares an image with the POX controller (Dockerfile).
18. https://pastebin.com/78hrh0tL Building and running an image with the POX controller.
19. https://pastebin.com/He4jkUUp Minimal POX controller base class with incoming packet handling.
20. https://pastebin.com/Hk8Y7U6t Container construction based on the YAML configuration file.
21. https://pastebin.com/pYxttinb MQTT packet recognition and control of packet processing method.
22. https://pastebin.com/mwzh3gVE The code responsible for counting ICMP-type packets and performing the action (scenario #1).
23. https://pastebin.com/ZCRemHKV Handling packets based on their type (only MQTT packets are subject to counting).
24. https://pastebin.com/PN7gdD5V Writing out anomaly occurrences every 60 s of program operation.
25. https://pastebin.com/PN8qVg8q Code that controls the flow of a specific MQTT packet.
26. https://pastebin.com/T5Yfvcs8 Sample package content in JSON.
27. https://pastebin.com/xzSJSDvi The code checks whether the “mac” key exists in the package and introduces error handling.
28. https://pastebin.com/iEVgzNHs Checking whether the physical MAC address is the same both in the packet content and in its header.
29. https://pastebin.com/RLk1qtqj Create a list of allowed addresses and check whether the sender’s physical address is on the list of allowed addresses.
30. https://pastebin.com/7Pzx4jHH Adding the MAC address of an MQTT intermediary (Broker) to allow bi-directional traffic.
31. https://pastebin.com/3AvPR1YD Checking of the package type and whether it is on the list of allowed packages.
32. https://pastebin.com/YZfDbfGe Cyclic sending of UDP packets to the MQTT Broker device in Python.
33. Dul, M., Gugała Ł. and Łaba K. Protecting web applications from authentication attacks, Advances in Web Development Journal, 2023; 1(1). doi:10.5281/zenodo.10049992
34. Aldowah, H., Ul Rehman, S. and Umar, I., Security in internet of things: issues, challenges and solutions. In Recent Trends in Data Science and Soft Computing: Proceedings of the 3rd International Conference of Reliable Information and Communication Technology (IRICT 2018) 2019; 396–405. Springer International Publishing.
35. Chica, J.C.C., Imbachi, J.C. and Vega, J.F.B. Security in SDN: A comprehensive survey. Journal of Network and Computer Applications, 2020; 159: 102595.
36. Sezgin, A., and Boyacı, A. Enhancing intrusion detection in industrial internet of things through automated preprocessing. Advances in Science and Technology Research Journal, 2023; 17(2): 120–135. https://doi.org/10.12913/22998624/162004
37. Bolanowski, M., Paszkiewicz, A., Żabiński, T., Piecuch, G., Salach, M., and Tomecki, K. System architecture for diagnostics and supervision of industrial equipment and processes in an IoE device environment. Electronics, 2023; 12(24).
38. Almutairi, Y.S., Alhazmi, B., and Munshi, A.A. Network intrusion detection using machine learning techniques. Advances in Science and Technology Research Journal, 2022; 16(3): 193–206. https://doi.org/10.12913/22998624/149934
39. Al-Fayoumi, M. and Al-Haija, Q.A. Capturing low-rate DDoS attack based on MQTT protocol in software Defined-IoT environment. Array, 2023; 19: 100316.
40. Cheng, H., Liu, J., Xu, T., Ren, B., Mao, J. and Zhang,

W. Machine learning based low-rate DDoS attack detection for SDN enabled IoT networks. International Journal of Sensor Networks, 2020; 34(1): 56–69.

41. Liu, Y. and Al-Masri, E. Slow Subscribers: a novel IoT-MQTT based denial of service attack. Cluster Computing, 2023; 26(6): 3973–3984.
42. Galeano-Brajones, J., Carmona-Murillo, J., Valenzuela-Valdés, J.F. and Luna-Valero, F. Detection and mitigation of DoS and DDoS attacks in IoT-based stateful SDN: An experimental approach. Sensors, 2020; 20(3): 816.
43. Ahuja, N. and Mukhopadhyay, D. June. Identification of DDoS Attack on IoT Network Using SDN. In 2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN) 2023; 879–884.
44. Xiong, F., Li, A., Wang, H. and Tang, L. An SDN-MQTT based communication system for battlefield UAV swarms. IEEE Communications Magazine, 2019; 57(8): 41–47.
45. Chen, X., Wu, T., Sun, G. and Yu, H. Software-defined MANET swarm for mobile monitoring in hydropower plants. 2019; 7: 152243–152257.