Intrusion Detection in Critical SD-IoT Ecosystem

Hammam Algamdi*, Gagangeet Singh Aujla*, Anish Jindal*, Amitabh Trehan*
*Department of Computer Science, Durham University, Durham, United Kingdom
E-mail: {hammam.algamdi, gagangeet.s.aujla, anish.jindal, amitabh.trehan}@durham.ac.uk

***Abstract*—The Internet of Things (IoT) connects physical objects with intelligent decision-making support to exchange information and enable various critical applications. The IoT enables billions of devices to connect to the Internet, thereby collecting and exchanging real-time data for intelligent services. The complexity of IoT management makes it difficult to deploy and manage services dynamically. Thus, in recent times, Software Defined Network (SDN) has been widely adopted in IoT service management to provide dynamic and adaptive capabilities to the traditional IoT ecosystem. This has resulted in the evolution of a new paradigm known as Software-defined IoT (SD-IoT). Although there are several benefits of SD-IoT, it also opens new frontiers for attackers to introduce attacks and intrusions. Specifically, it becomes challenging working in a critical IoT environment where any delay or disruption caused by an intruder can be life-threatening or can cause significant destruction. However, given the flexibility of SDN, it is easier to deploy different intrusion detection systems that can detect attacks or anomalies promptly. Thus, in this paper, we have deployed a hybrid architecture that allows monitoring, analysis, and detection of attacks and anomalies in the SD-IoT ecosystem. In this work, we have considered three scenarios, a) denial of services, b) distributed denial of service, and c) packet fragmentation. The work is validated using simulated experiments performed using SNORT deployed on the Mininet platform for three scenarios.**

***Index Terms*—Internet of Things, Intrusion Detection System, Software-defined Networking, SNORT.**

I. Introduction

The Internet of Things (IoT) refers to the ability of objects, or 'things', to exchange information without requiring direct human interaction [1]. These may be sensors, actuators, computers or intelligent objects capable of monitoring and interacting with their internal and external environments. Various technologies interconnect and enable new applications in an IoT environment by connecting physical objects with intelligent decision-making support. Unlike traditional IT networks with well-established architecture, research communities or industries (e.g., Cisco) suggest several IoT architectures. However, a general IoT network architecture consists of three layers: a sensor and object layer, a transmission network layer, and an application layer. The IoT connects billions of devices to the Internet, collecting and exchanging real-time data to provide intelligent services. These connected devices can control and access the IoT services remotely, utilising the underlying network infrastructure. However, the requirements of the IoT cannot be scaled and efficiently handled by existing traditional network technologies [2]. The complexity of Internet management makes it difficult to deploy new services dynamically. Adding new features and re-configuring a network is exhaustive if done traditionally [3]. However, some of the complex issues associated with IoT management can be addressed through enabling technologies such as Software Defined Networks (SDN). Having a programmable network has been in development for some time. A serious programmable network model began to emerge in 2007 [4].

Over the years, network operations and management have become more complex with the increased number of connected IoT devices. In the years that followed, many RFCs and IEEE standards were developed, and network vendors - who had to satisfy many clients with one product - continued to add features to their routers, even when their customers did not use all of them [5]. The SDN, by contrast, separates the control plane from the data plane and enables the programmability of the network. SDN controllers act as network operating systems (NOS) and handle the logic and computation required for network operations - hence they are referred to as network brains. Data plane devices (routers/switches) perform simple matching operations to determine how to forward packets based on flow tables [6]. Figure 1 illustrates the difference between traditional network architecture and SDN technology.

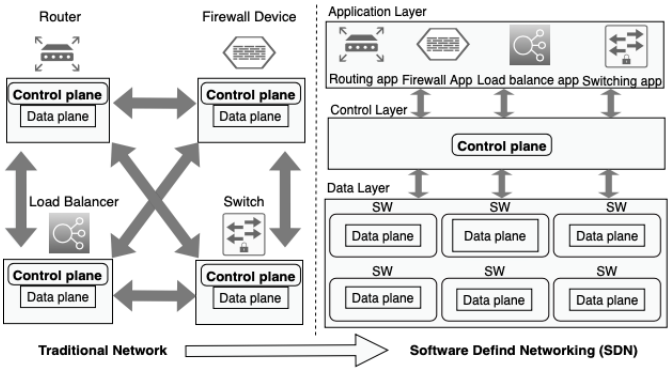

Fig. 1: Comparison between traditional networks and SDN

Figure 1 shows the multiple layers of SDN architecture. The data layer consists of a forwarding plane and an operational plane. The forwarding plane handles packets based on the instructions (e.g., forwarding, dropping, and changing packets) from the control plane. The operational plane determines a network device's state, such as whether it is active and which ports are available. The control layer consists of a control plane and a management plane. The control plane assigns instructions to one or more network devices on how packets should be forwarded. The management plane is responsible for configuring, monitoring, and maintaining network devices, such as making decisions about the state of the network

device. It focuses on the operational plane of the device and less on the forwarding plane. An application or service that defines network behaviour resides on the application layer. The applications (including routing processes), which directly support the forwarding plane (e.g., functions within the control plane), are not considered a part of this layer.

The control layer can communicate with the application and data layer via north and south interfaces. A southbound interface connects the control layer with the data layer in network devices, while a northbound interface connects the control layer with the application layer. An SDN controller (such as Opendaylight, Floodlight, etc.) consists of control-plane applications and services that use OpenFlow in the southbound direction and offer an interface for applications in the northbound direction. The OpenFlow protocol defines how a logically centralized controller controls OpenFlow switches. OpenFlow switches maintain one or more flow tables used to look up packets. Different actions need to be taken regarding packet lookup and forwarding.

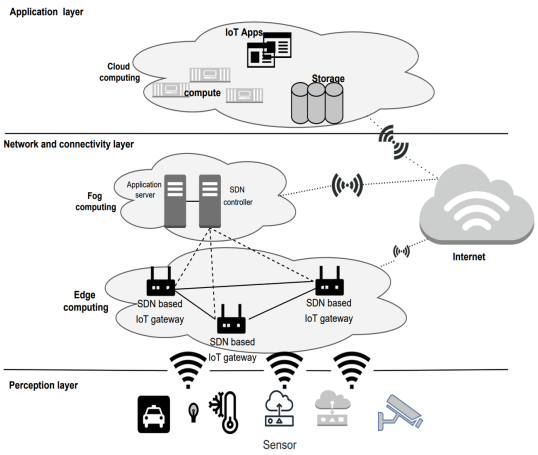

Fig. 2: Software-defined IoT

SDN can improve the distribution and control of traffic flow in the network for load balancing and minimizing network latency. The SDN global view of the network can be leveraged to improve load balancing, fine-grained traffic forwarding, and bandwidth utilization. By virtualization, hardware resources can be optimized with low network load, and network infrastructure can be shared between different service providers and services [3]. Owners of IoT devices desire the ability to control and secure them from anywhere at any time. Since the SDN has access to all network components, this can be easily accomplished [2]. SDN controllers can send requests from multiple users through the desired path based on their flow rules. The SDN's ability to efficiently map user requests will improve resource utilization, an essential requirement for the IoT [7]. The SDN's centralized control makes network monitoring and consistency verification easier. Consequently, some types of attacks (e.g. Denial of Service (DoS)) can be mitigated, which is an advantage for IoT security [3]. Thus, motivated by the above benefits, Figure 2 depicts the reference architecture for the SD-IoT.

A. Research Problem and Motivation

While the SDN offers many advantages in IoT environment, some issues (such as detecting intrusions and anomalies in data traffic) still remain unaddressed. In the IoT ecosystem, the endpoints (sensors and actuators) generate big data for various applications (sensing temperature, light, motion, etc.). The timely delivery of data packets is vital since it impacts the data-driven decision-making process within critical IoT environments. In IoT systems, monitoring, analysing, and controlling all the traffic generated is not easy. It is difficult to detect and block traffic in the event of intrusions or anomalies due to IoT devices' distributed and diverse nature. However, timely detection of intrusions (and/or anomalies) can help to prevent a significant degradation in the performance or failure of IoT services.

For IP packets to be transmitted across a network, they must comply with specific standards. Packets that are received several times (in a repeated manner), reach multiple hosts (at the same or different times) or exceed the network's maximum transmission unit (MTU) may not be normal and thus require different treatment. A DoS attack floods a server with traffic, making a resource unavailable. The flooding attack, a kind of DoS attack, can overload a host by sending the same packet repeatedly, causing its resources to be depleted and making the services unavailable. Flooding attacks overwhelm the target host's resources by sending many requests or messages to prevent it from processing legitimate requests or providing routine services. Consequently, the target host can become unresponsive, slow, or even crash. A Distributed DoS (DDoS) attack is a type of DoS attack that uses multiple computers or machines to flood a targeted resource [9]. Moreover, specific hosts can be unreachable when a DDoS attack targets multiple hosts within the network, such as with a sweep scan. Defeating DoS or DDoS attacks is critical to keep networked systems and services available.

Both attacks overload a server or web application intending to interrupt services. IP headers contain packet information, including fragment IDs (the same as IP IDs), fragment offsets, fragment lengths, and more fragment flags. Packets that exceed a network's MTU must be fragmented for transmission over the network. The MTU of Ethernet, for example, is 1500 bytes. This means that packets transmitted over Ethernet cannot exceed 1500 bytes in size. In order to send malicious packets through the network, attackers may exploit weaknesses in IP header structures by flooding packets, broadcasting ping network broadcasts, or fragmenting packets. In order to avoid detection by IDS, attackers may use various techniques, including sending smaller packets to a single host or multiple hosts, fragmenting data and reassembling it, or sending data bigger than the network's MTU. It can lead to system failures and crashes, posing persistent challenges to network security.

B. Contributions

Given these points, our contributions are listed below.

- We deploy a hybrid architecture for IDS in an SD-IoT system that monitors, analyzes, and detect intrusions in a realistic virtual IoT network.
- We design three attack scenarios, namely, a) DoS, b) DDoS, and c) packet fragmentation, for analyzing the performance of hybrid architecture.
- We simulate the three attack scenarios on the SDN network and evaluate the performance of IDS, thereby validating its usage in the SD-IoT paradigm.

II. Proposed Hybrid Architecture for Intrusion Detection in SD-IoT Environment

This section proposes a hybrid architecture for intrusion detection managed through a control plane in the SD-IoT environment. Fig. 3 shows the proposed hybrid architecture segregated into different components discussed in the subsequent sections.

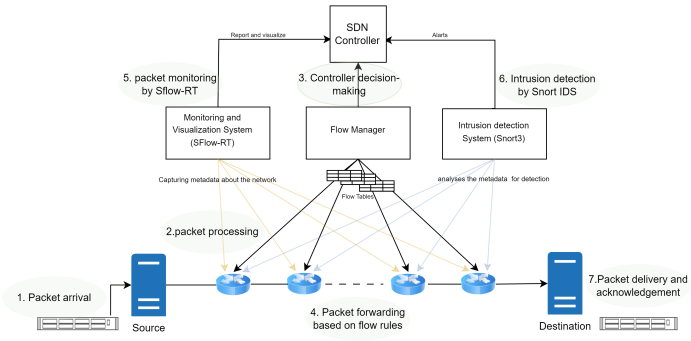

Fig. 3: Proposed Architecture for Intrusion Detection

A. Flow Manager

The Flow Manager component manages the flow table in the switches and routers within the network. This table is the foundation of OpenFlow switches. The switch might have one or more flow tables, and a packet must pass through at least one. Each entry in the table contains five main components - Match, Action, Counter, Priority, and Timeout. The flow manager checks each incoming packet to ensure matching flow entries are available in the flow table so the packets can smoothly travel across the network. It forwards the packet if there is a suitable match in the flow table based on the corresponding entry in the action field. However, if no suitable match is available, it requests the SDN controller to create a new flow rule and install it in the switch's flow table. In this way, the flow manager ensures efficient traffic across the network.

B. Monitoring and Visualization System

In this architecture, we use Sflow-RT[1] as a monitoring and visualization system managed by the SDN controller. Sflow-RT captures metadata about network traffic and generates reports and alerts based on this metadata. The southbound API connects it to the flow manager and SDN controller. It analyses traffic in real-time and displays traffic statistics and flow data from applications through its API. Sflow-RT provides the following functionality in the hybrid architecture.

- It allows monitoring of switches, routers, servers, and virtual machines. It analyzes packets from the network to extract information about network flows by sampling packets from the network.
- It provides real-time information on network traffic and flow and a graphical representation of the network topology.
- It helps to visualize information about flow statistics (including packet and byte counts), flow duration, and network protocols.

C. Intrusion Detection System using SNORT

Snort IDS is a powerful open-source real-time packet processing and traffic analysis tool. It makes use of several rules that allows the detection of any anomalous or malicious incidents. It can be used in multiple roles, like, a) packet sniffer, b) packet logger (for traffic debugging), and c) full-fledged IDS. The underlying architecture of Snort comprises several components. These components are shown in Figure 4 and described below.

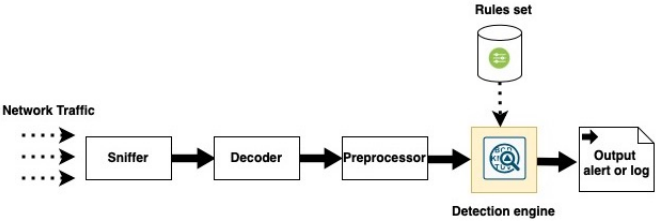

Fig. 4: Snort3 components

- **Sniffer:** The packet sniffer mirrors the packets traversing through a specific network interfacing these to the Snort tool for detailed analysis.
- **Decoder:** This component decodes the mirrored packets for further analysis to identify malicious aspects or for network debugging.
- **Preprocessors:** A collection of engines matching specific types of packets, such as those with larger MTU size or certain applications.
- **Detection Engine:** This component matches packets against the match conditions configured on the Snort tool.
- **Rules:** The rules represent the specific match conditions with actions configured in the Snort tool that is passed to the detection engine. The rules can be designed and deployed as and when required based on the requirement of the underlying network.
- **Output:** The outcome or action taken by the Snort tool based on the rule is collected as output.

III. Results and Validation

A. The Experimental Setup

The experiments were conducted on an Ubuntu virtual machine running on VirtualBox. The network topology shown in Figure 5 was deployed on Mininet, a network emulator. The deployed scenario uses Open Virtual Switch (OVS)

[1]https://sflow-rt.com/

and Floodlight controller, an SDN controller that uses the OpenFlow protocol to orchestrate traffic flows. Sflow-RT, a monitoring tool, provides real-time visibility to SDN. It offers a comprehensive perspective of network active routes and usage.

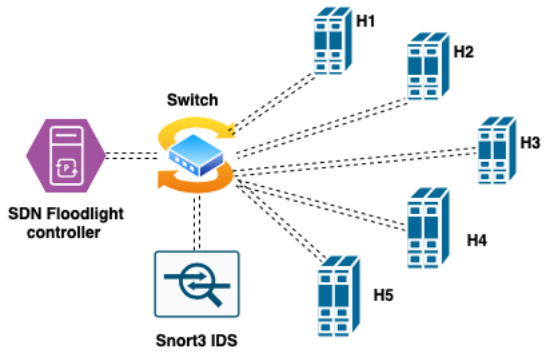

Fig. 5: The experiment network topology

B. Experimental Scenarios

In this work, we have considered three scenarios for experimentation. These scenarios are explained below.

- **DoS attack:** The first scenario involved a DoS attack launched from Host 1 to Host 2, as shown in the topology in Figure 5. Snort3 was used to capture the traffic for further examination.
- **DDoS attack:** This scenario involved the setup of an SDN scenario comprising five hosts (Figure 5). Host 5 was designated as the sniffer where Snort was employed to monitor the traffic from Host 4.
- **Packet Fragmentation attack:** As illustrated in figure 5), the attack was launched from Host 1 to Host 2 while Snort3 was capturing network traffic for analysis.

C. Results and Discussion

This experiment simulated three scenarios to check the Snort3 operations and performance.

1) DoS Scenario: The first scenario involved a DoS attack launched from Host 1 to Host 2, as shown in the topology in Figure 5. Snort3 was utilized to capture the traffic for further examination. During the simulation, the Sflow Chart (Figure 6) revealed a sudden surge in traffic between Host 1 (represented in blue, with IP: 10.0.0.1) and Host 2 (represented in red, with IP: 10.0.0.2) during the attack. This resulted in unusual data transfer behaviour. During the development of our experiment, we employed customized rules to identify specific alerts related to our testing scenarios. For instance, in this case, one rule we utilized was designed to detect a typical DoS attack, an Internet Control Message Protocol (ICMP) flood attack, also known as a Ping flood attack. The rule would trigger an alert with the message ”PROTOCOL-ICMP Attack from Host-1 to Host-2.”. To demonstrate the effectiveness of this rule, we carried out a test in which a large volume of ICMP protocol data packets was initiated from Host 1 and directed towards Host 2. Snort promptly responded with alerts generated on its console, as shown in Figure 7.

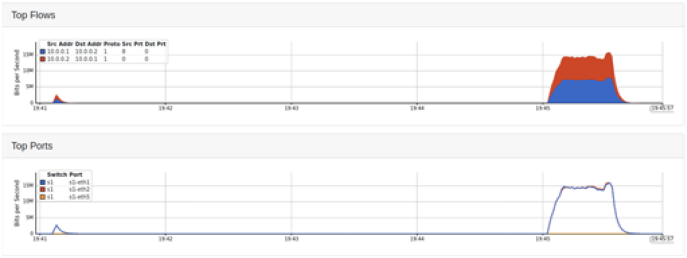

Fig. 6: Sflow tool showing increased traffic on Host1 and Host2 ports during attack

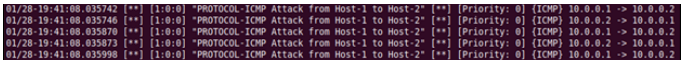

Fig. 7: Alert generated by snort3 after detecting a DoS attack

The attack lasted approximately 30 seconds, and its packet statistics are displayed in Table I. The log file presents Snort received 20,800 packets. Out of these, it analyzed 16,276 packets, dropped 4,524 packets, and marked 4,524 as outstanding. These results indicate that Snort effectively handles and analyses most packets during a DoS attack, demonstrating its robust performance under this type of threat.

TABLE I: Packet Statistics

Scenarios	DoS	DDoS	Packet Fragmentation
Received	20800	206888	78731
Analyzed	16276	140848	56051
Dropped	4524	66040	22680
Outstanding	4524	66040	22680
Allow	16276	140848	56051

2) DDoS scenario: This scenario involved the setup of an SDN ecosystem comprising five hosts (Figure 5). Host 5 was designated as the sniffer and employed Snort to monitor the traffic originating from host four. The network consisted of three malicious hosts (h1, h2, h3), a victim host (h4), and an IDS represented by a Snort-equipped host (h5). The five hosts were interconnected through a switch connected to the Floodlight SDN Controller. A DDoS attack was launched against the victim host (h4) by the three malicious hosts (h1, h2, and h3), as evidenced by the Sflow monitoring tool, which can be seen in Figure 8. The flow of ping packets flood sent from h1 (represented by a green line on port s1-eth1), h2 (represented by an orange line on port s1-eth2), and h3 (represented by a red line on port s1-eth3) to h4 (represented by a blue line on port s1-eth4). However, the IDS machine (h5) could detect the attack and trigger an alert per our customized rule to detect an ICMP flood DDoS attack. The alert was generated and displayed the ”unusual ping detected” message in the Snort console, as depicted in Figure 9. After stopping Snort, the log statistics in Table I reveals that the detection engine has done a remarkable job identifying and managing incoming packets. Out of the 206888 packets received, a significant portion of 140848 packets was successfully analyzed. The rest 66040 packets were dropped based on the analysis performed by the engine. This analysis shows that the attack was a DDoS attack initiated by multiple sources or hosts. In this

case, three hosts initiated the attack. Such distributed attacks are more challenging to detect and prevent than attacks from a single source. However, the detection engine was able to receive, analyze and manage the incoming packets efficiently, demonstrating its capability to handle complex security threats and maintain the integrity of the network.

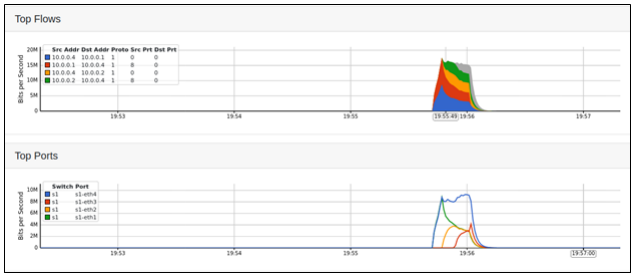

Fig. 8: Sflow showing increased traffic due to DDoS attack

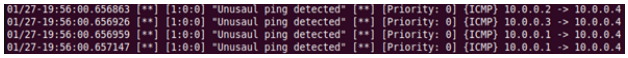

Fig. 9: Alert generated by snort3 after detecting DDoS

3) Packet Fragmentation scenario: In the third case scenario, an attempt to compromise network security was made through a packet fragmentation attack. The experiment on the topology is illustrated in Figure 5, and the attack was launched from Host 1 to Host 2 while Snort3 was capturing network traffic for analysis. The sflow chart in Figure 10 displays the sudden increase of data flow during the attack between (IP: 10.0.0.1, represented by a blue colour on the chart) and host 2 (IP: 10.0.0.2, represented by red colour on the chart) during the attack. Snort was able to detect the attack and respond promptly. This is evident from the alerts generated by the system and displayed in the Snort console, as seen in Figure 11. The attack was carried out by sending many oversized ICMP data packets from Host 1 to Host 2, which resulted in the fragmentation of the packets and a corresponding increase in the data flow between the two hosts. The ability of Snort to detect and respond to the attack effectively highlights its effectiveness in safeguarding the network against security threats in the SD-IoT environment. The packet fragmentation attack can cause significant damage to the network and its connected devices by exploiting vulnerabilities in the IP protocol. This attack aims to divide large packets into smaller ones to bypass security measures and gain unauthorized access to the targeted system. In this scenario, Snort detected the attack by analyzing the network traffic and generating alerts, providing valuable information to the network administrator for further investigation and mitigation efforts. The logs in Table I demonstrate the difference in received packets (78731) compared to analyzed packets (56051) by Snort. The fact that 22680 packets were not analyzed suggests that Snort was overwhelmed by the traffic or that the packets were deemed a risk and were dropped as a precautionary measure. The analysis revealed many sessions generated by the attacker (Host 1) towards the victim (Host 2) due to the fragmentation caused by the large MTU size of the ICMP packets. The results show that 5210 sessions were created during the fragmentation attack, resulting in 81794032 reassembled and 83921728 fragmented bytes. This generated significant traffic, potentially causing the network and the victim host to be unavailable. However, Snort detected the fragmentation attack, but the enormous amount of traffic could still threaten the network's stability, which requires further investigation.

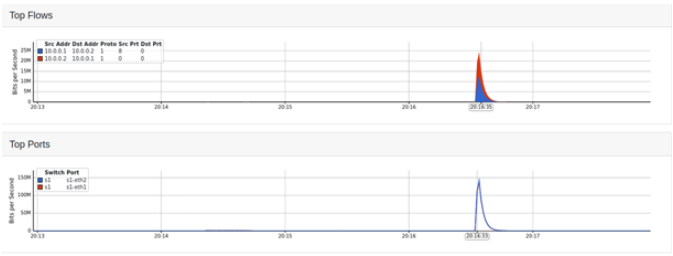

Fig. 10: Sflow tool showing the increased traffic because of packet fragmentation attack

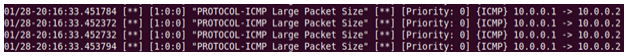

Fig. 11: Alert generated by snort3 after detecting an attack

D. Resource Usage Analysis

We monitored Snort's resource consumption during our tests while performing intrusion detection tasks under different attack scenarios. We utilized the Linux "top" command to observe CPU and memory usage before and during the three attack scenarios: DoS attack, DDoS attack, and Packet Fragmentation attack. Table II summarises the CPU and memory usage results during the three attacks.

TABLE II: Resource usage before and during attack

Scenario	Description	Before attack	During attack
DoS	CPU Usage	1.1%	13.01%
	Memory Usage	2.2%	2.2%
DDoS	CPU Usage	1.1%	9.7%
	Memory Usage	2.2%	2.2%
Packet Fragmentation	CPU Usage	1.1%	13.02%
	Memory Usage	2.2%	8.7%

Before the attacks, Snort's CPU usage was at 1.1%, and memory usage was at 2.2%. During the attack scenarios, Snort's CPU and memory usage changed as follows:

- **DoS Attack:** Snort's CPU usage increased to 13.2% during the DoS attack, while memory usage remained constant at 2.2%. This indicates that Snort effectively handles DoS attacks with a moderate increase in CPU usage but without a significant impact on memory consumption.
- **DDoS Attack:** In the DDoS attack scenario, Snort's CPU usage reached 9.7%, and memory usage stayed at 2.2%. The results show that Snort can manage large-scale, distributed attacks with a slight increase in CPU usage while maintaining stable memory consumption.
- **Packet Fragmentation Attack:** During the Packet Fragmentation attack, Snort's CPU usage rose to 13.2%,

and memory usage increased to 8.7%. This demonstrates that Snort can process and analyze fragmented packets effectively, albeit with a higher memory usage increase than the other two attack scenarios.

Snort consumes resources efficiently when performing intrusion detection tasks under various attack scenarios. Snort maintains relatively moderate resource consumption levels during attacks, demonstrating its ability to handle a variety of network threats without overburdening the system.

E. Scalability Analysis

Our study tested Snort's ability to analyze different numbers of packets. We sent varying packet volumes, as illustrated in Fig. 12. The graph includes five data points, representing packet counts of 5k, 10k, 25k, 50k, and 100k, with corresponding analysis percentages of 97.86%, 95.03%, 98.68%, 97.91%, and 98.55%. The x-axis denotes the number of packets received by Snort, while the y-axis displays the percentage of packets analyzed. As can be seen from the graph, Snort consistently maintains high performance during the analysis of different packet volumes. Despite the significant increase in packets received, the analysis percentages remain relatively stable. This finding suggests that Snort can efficiently handle various packet volumes with minimal impact on its analysis capabilities.

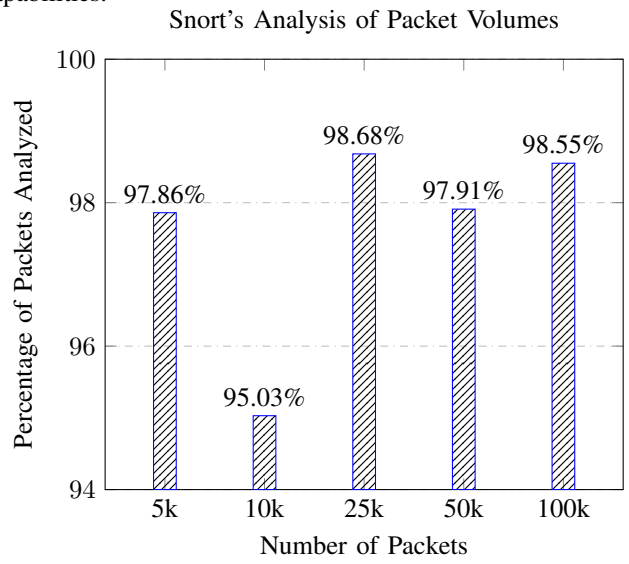

Fig. 12: Performance for Different Packet Volumes.

In another test conducted during our experiment, we evaluated Snort's ability to analyze packets under varying flood duration. We sent floods of packets over different time intervals and measured the percentage of packets Snort successfully analyzed compared to the number of packets received. The results, illustrated in a graph 13, demonstrate that Snort was highly effective in analyzing packets under all tested flood duration.

IV. Conclusion

In this paper, we deployed a hybrid architecture to detect malicious incidents in SD-IoT traffic based on three attack scenarios, DoS, DDoS, and packet fragmentation for different sizes of packet fragments. During this work, the primary objective was to assess the ability of the deployed hybrid architecture to detect these attacks and their effectiveness as an IDS solution in an SD-IoT environment. While the system effectively detects these attacks, attackers can still bypass its detection through various methods. It is necessary to perform additional testing on different networks and real-world scenarios to improve IDS systems' efficiency. The development of accurate attack detection rules begins this ongoing process.

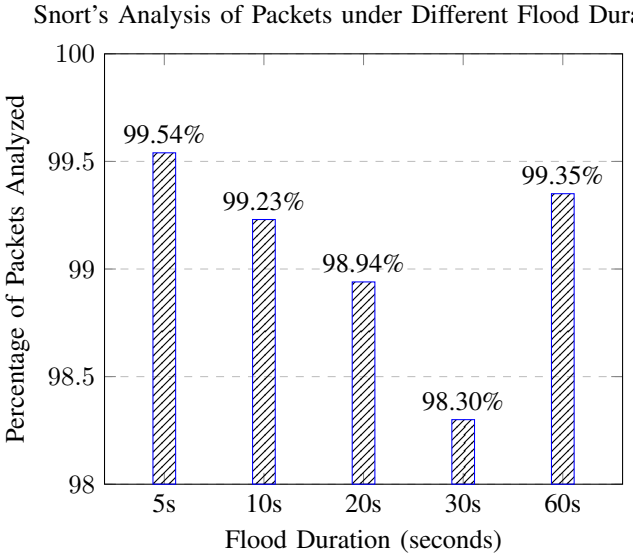

Fig. 13: Performance Under Varying Flood Duration's.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," IEEE communications surveys & tutorials, vol. 17, no. 4, pp. 2347-2376, 2015.

[2] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," IEEE Internet of Things Journal, vol. 4, no. 6, pp. 1994-2008, 2017.

[3] O. Salman, I. Elhajj, A. Chehab, and A. Kayssi, "IoT survey: An SDN and fog computing perspective," Computer Networks, vol. 143, pp. 221-246, 2018.

[4] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 87–98, 2014, doi: 10.1145/2602204.2602219.

[5] M. Casado, N. McKeown, and S. Shenker, "From ethane to SDN and beyond," SIGCOMM Comput. Commun. Rev., vol. 49, no. 5, pp. 92–95, 2019, doi: 10.1145/3371934.3371963.

[6] R. Chaudhary, G. S. Aujla, N Kumar, and P.K. Chouhan, "A comprehensive survey on software-defined networking for smart communities," International Journal of Communication Systems, e5296, 2022.

[7] A. El-Mougy, M. Ibnkahla, and L. Hegazy, "Software-defined wireless network architectures for the Internet-of-Things," in 2015 IEEE 40th local computer networks conference workshops (LCN Workshops), 2015: IEEE, pp. 804-811.

[8] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 732-794, 2015.

[9] Dong, S., et al. (2019). "A Survey on Distributed Denial of Service (DDoS) Attacks in SDN and Cloud Computing Environments." IEEE Access 7: 80813-80828.

[10] Khraisat, A., et al. (2019). "Survey of intrusion detection systems: techniques, datasets and challenges." Cybersecurity 2(1): 20.