Using supervised machine learning in an SDN edge-controlled IoT network for anomaly detection and mitigation

Andrea Antony aanton24@uwo.ca
Sudipto Baral sbaral3@uwo.ca
Gabriella Gerges ggerges2@uwo.ca
Xi He xhe379@uwo.ca
Roman Koval rkoval2@uwo.ca

I. Introduction

The Internet of Things (IoT) is a concept which represents the Internet extending beyond traditional computing devices like desktop computers and mobile phones. IoT networks include a multitude of devices, often referred to as "smart" devices, equipped with sensors, actuators, and communication hardware, enabling them to collect, exchange, and act on data, often with minimal human intervention [1]. This concept is not new however, it has grown increasingly popular due to the exponential development of modern hardware in the last decade [2]. Its originated purpose was to observe, identify, and understand the world with the limitations of error prone human-entered data [3].

The IoT infrastructure is recognized for its impact on various aspects of daily life and the impact it has on the creation of smart cities, smart grids, intelligent transportation, smart healthcare and many more. IoT devices in healthcare, particularly wearable wireless devices, are increasingly used to monitor vital signs in hospitalized adults continuously. Compared to standard intermittent vital signs measurements, these devices offer the potential for timely detection of clinical deterioration. However, as of a study conducted up to September 2019, these devices were still in the clinical validation and feasibility testing phases. No high-quality, extensive, well-controlled studies were available that showed such devices' significant clinical benefit or cost-effectiveness. Further research is needed to establish their effectiveness in clinical practice or in-home monitoring [4]. The global market size of IoT is even expected to grow significantly as the reliance on these networks grows [5].

These devices must have a way to protect against attacks due to the relative importance of the IoT network they may be a part of. For example, an IoT network within a hospital would have dire consequences if, let us say, a Distributed Denial of Service attack was to take place on the network, with the attackers' goal of changing the availability of the network to hospital staff or patients. Developing a cost-effective and inexpensive method for IoT devices to protect against such attacks is the problem we will address in our research project.

II. Problem Specification

Despite the overwhelming benefits associated with the Internet of Things (IoT), significant scalability, security, and standardization challenges persist. IoT device hardware is commonly equipped with computational resources optimized for low power consumption and cost-effectiveness rather than high performance. They typically house micro-controllers or low-power CPUs that provide enough computational power for their singular tasks while consuming minimal energy. This is a necessity for scalability in extensive IoT networks. However, this limited computational capacity can pose challenges as the network grows, potentially leading to issues in handling increased data volumes and device management [6].

Security challenges in IoT are complicated, originating from the diverse and expansive nature of the network. The limited computational resources of IoT devices often mean they cannot implement advanced security features, making them vulnerable to cyber-attacks [7]. This vulnerability is further amplified by the vast number of devices and the need for continuous data exchange. This increases the potential attack surface for malicious activities.

Another significant challenge is the need for universal standards in IoT, which can lead to decreased interoperability between devices. The absence of standardized protocols and communication interfaces can result in compatibility issues, hindering seamless communication and data exchange between different devices within the network [8]. This lack of standardization can negatively affect the development and deployment of scalable and secure IoT solutions, as manufacturers and developers may follow divergent design principles and communication protocols.

Software-defined networking (SDN) offers a robust solution to the security challenges of the Internet of Things (IoT). By separating the control and data planes, SDN enables centralized management, which is crucial for efficiently monitoring and securing the widespread and often vulnerable IoT devices. The programmability of SDN is critical, allowing for quick adaptation and the implementation of advanced security protocols network-wide, a significant advantage given the dynamic nature of cyber threats. Additionally, SDN's use of open APIs encourages the development of specialized security applications, which can leverage central control for coordinated defence strategies, thus significantly bolstering IoT network security. This architecture enhances security and introduces flexibility and scalability in protecting an expansive network of interconnected devices. Machine learning is being

widely deployed in software-defined networking (SDN) to efficiently monitor network dynamics, analyze network data, and predict network usage [9].

This paper will use a supervised machine learning algorithm within an SDN edge-controlled IoT network to offer an anomaly detection and mitigation. This solution will act as an attempt to address security concerns of IoT devices while keeping in mind the resource limitataions.

III. Literature Review

Shafi et al. [10] creates a framework that implements an SDN to monitor IoT traffic and applies forwarding rules to IoT devices to mitigate the attack if an intrusion is detected. Because the computation required for anomaly detection is too expensive to run on an IoT device, all detection occurs at the SDN controller, allowing the system to scale independently from the IoT devices. The paper evaluates multiple supervised machine learning techniques trained on the UNSW-NB15 network attack dataset; an E3ML (entropy-based triple machine learning) technique is chosen. This paper is the primary motivation for our proposed solution; however, our supervised models will be simpler than this paper has proposed for intrusion detection.

Sarac M. et al. at Singidunum University [11] propose a block-chain interface that separates the communication between an IoT device and its network to remove the single control authority. Their method allows for actions performed by each device in the network to be tracked and mapped; it adds anonymity and versatility to the IoT infrastructure while also improving the reliability of data sent to remote services. Although their block-chain interface benefits the IoT security infrastructure, it also has a limitation of storage and scalability while having expensive processing times, energy consumption, and overall required resources.

K. Giotis et al. [12] investigate using the OpenFlow protocol to enhance RTBH (Remotely Triggered Black Hole) routing for mitigating DDoS attacks on legacy networks. They leverage OpenFlow's network programmability for dynamic RTBH configuration, enabling per-flow traffic handling and ensuring the victim's regular operation while moving the mitigation process toward the network's edge. They introduce a sketch-based anomaly detection mechanism that identifies the victim and remotely triggers offensive traffic mitigation. This paper showcases an approach where SDN can be leveraged for efficient DDoS attack detection in legacy networks.

Ashraf et al. [13] propose an SDN-based IoT Anomaly detection system using ML algorithms, specifically targeting botnet attacks from compromised hosts. They extract 13 vital network flow features, later streamlined to 5 for optimization, which are used to train three machine learning models—Support Vector Machines (SVM), k-Nearest Neighbours (kNN), and Multilayer Perceptron (MLP). Through evaluations using the UNSW and ISCX benchmark datasets, they achieve remarkable detection rates, with the kNN model showing almost 99% accuracy for the ISCX dataset and the MLP classifier demonstrating consistent accuracy, showing competitive results compared to similar ML techniques based Intrusion detection systems. In contrast, our solution will look at different classifiers while utilizing a different dataset.

Bhayo et al. [14] demonstrate a new machine learning-based framework called SAD-F; the framework uses various machine learning algorithms such as decision trees, random forests, and support vector machines, and the traffic is captured by a logging mechanism added to the SDN-WISE(Software Defined Networking solution for Wireless Sensor Networks) controller, then the framework uses algorithms mentioned above to classify network packets, they achieved a peak accuracy of 98.1% for Decision Tree; however, they only account for DDoS attacks. Our approach aims to support multiple attack variations.

Wang et al. [15] propose to use SDN sEcure COntrol and Data plane (SECOD) algorithm to resist DDoS attacks on the real SDN-based IoT testbed, SECOD monitoring the network and comparing the real-time counter with a predefined threshold every fixed period, once it detects DDoS, they will insert predefined rules to OpenFlow switches to block the access in the data plane to against the DDoS attack, this approach does not involve machine learning and have a bad limitation when comes to other attacks not included in the predefined method, our approach aims to generalize the attack traffic feature detection.

Mustafa and Quasem [16] employed a supervised learning algorithm to detect LR-DDoS attacks in the MQTT protocol, a key IoT communication protocol. While their lightweight supervised models demonstrated high accuracy, they did not assess their performance with real attack datasets. Moreover, their solution is protocol-specific and lacks mitigation techniques. In contrast, our solution aims to be a full attack detection and mitigation system.

Shafi et al.'s SDN-based framework for monitoring IoT traffic and Sarac M. et al.'s blockchain interface for IoT communication security represent significant strides in IoT security, each addressing specific challenges like scalability and resource consumption. K. Giotis et al.'s utilization of OpenFlow protocol for RTBH routing and Ashraf et al.'s ML-based anomaly detection system reflects the growing integration of advanced networking protocols and machine learning in cybersecurity. Bhayo et al.'s SAD-F framework, highly accurate but limited to DDoS attacks, contrasts with Wang et al.'s SECOD algorithm, which is effective against DDoS yet lacks the adaptability to diverse threats. Despite its accuracy, Mustafa and Quasem's focus on LR-DDoS attacks in the MQTT protocol is constrained by its protocol-specific nature and the absence of comprehensive mitigation strategies. Inspired by these varied methodologies, our research aims to develop a more encompassing approach. By focusing on a comprehensive attack detection and mitigation system that integrates machine learning for broader and more dynamic threat response, our proposed solution offers a more adaptive and robust defence mechanism for the complex and evolving challenges in IoT environments while remaining simple.

IV. Background Knowledge

A. Software-Defined Networking

Software-defined networking (SDN) represents a progressive architectural model in the realm of network technology. It fundamentally restructures network design by defining the control plane, responsible for decision-making, from the data plane, tasked with data forwarding. This enables centralized software management, thereby enhancing the overall network behavior [17]. SDN offers benefits in terms of simplifying the administration of both existing and forthcoming network infrastructures, facilitating the integration of advanced technologies [18].

This innovative approach addresses the inefficiencies of traditional routing methods by segmenting the network into two distinct planes, fostering a more dynamic and efficient configuration. A pivotal feature of SDN is its capacity to render network devices programmable, transferring the locus of network management and control to software-based systems [19].

A programmable software entity, called the controller, is central to SDN's functionality, orchestrating packet forwarding. Conversely, the forwarding elements are relegated to the role of conduits for data transmission, operating under the directives of the controller [20]. Furthermore, SDN incorporates open application programming interfaces (APIs), promoting seamless interaction between the controller and various applications and developing innovative network management applications.

B. Environment Tools

1) Python: Python is a high-level interpreted programming language known for its simplicity [21]. It has an extensive standard library and a large community of developers. Python enables rapid prototyping and application development due to its vast ecosystem of libraries and frameworks. The system will be implemented in Python using libraries provided by the Mininet project and the Ryu framework.

2) Mininet: Mininet is a network emulator which creates virtual networks using process-based virtualization and network namespaces. It provides a virtual test bed for software-defined networks (SDN).

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network. In Mininet, hosts are emulated as bash processes running in a network namespace, so any code that would normally run on a Linux server will run in a Mininet "Host" [22]. The Mininet "Host" will have its own private network interface and can only see its own processes. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. Mininet enables rapid prototyping of SDNs with a complex topology without the need for a physical network [23].

Mininet has three components; Isolated Hosts, which is a group of user-level processes moved into a network namespace that provide ownership of interfaces, ports and routing tables, Emulated links that enforces data rate of each link to shape traffic and emulated switches used to switch packets across the network. We are using Mininet framework for simulating SDN networks by running a pre-built Mininet virtual machine (VM) [24].

While there are many network simulation tools, Mininet was chosen due to its simplicity and focus on SDN simulation through the OpenFlow protocol. The Mininet VM comes bundled with all the necessary tools and Python libraries to quickly prototype SDN-enabled applications. Instead of configuring a virtual SDN environment, time was spent developing the intrusion detection system using the default parameters and configurations provided by Mininet. Mininet also provides its own Python API that was used to programmatically design the SDN topology and run the network.

3) Ryu: Ryu is a component-based software defined networking framework. Ryu provides software components with well-defined API that make it easy for developers to create new network management and control applications. The framework facilitates development by providing the basic functions for controlling the data plane and the functions that are common to SDN applications [25]. It is designed to increase the agility of the network by making it easy to manage and adapt how traffic is handled. The SDN Controller is the brain of the SDN environment and communicating information down to the switches and routers with southbound APIs, and up to the applications and business logic with northbound APIs [26]. We use Python and the Ryu framework for enhancing a SDN controller with intrusion detection and mitigation capabilities.

The Ryu framework was chosen for implementing the SDN controller because of its compatibility with the Mininet Python environment and its support for the OpenFlow protocol. Both, the ethernet switch created as part of the Mininet topology and the SDN controller communicated using the OpenFlow protocol through the default TCP port of 6653. Additional configurations were not needed to enable the communication between the Mininet SDN network and the Ryu SDN controller.

4) VirtualBox: VirtualBox is a powerful x86 virtualization product we used for running the Mininet virtual machine. VirtualBox is designed to run virtual machines on your physical machine without the need to reinstall the host operating system [27]. VirtualBox is also the recommended way to run the Mininet VM per the Mininet documentation.

5) Hping3: The hping3 tool is used to simulate a network attack [28]. This network tool allows the creation of custom TCP packets to test connectivity and perform network attacks on hosts.

V. Methodology

A. Intrusion Detection System

Our proposed approach employs a three-model framework for detecting DDoS attacks within Software-Defined Networking (SDN). In the subsequent sections, the architecture of the intrusion detection system, the dataset we used, and the performance of our model are discussed.

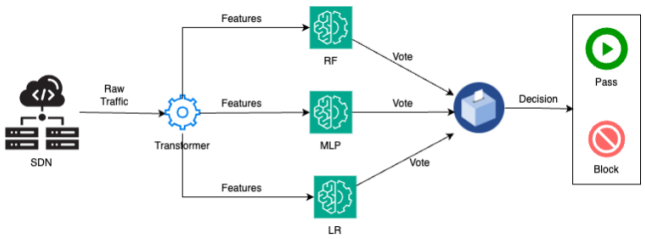

Fig. 1. Architecture of Intrusion Detection System

a) Architecture: The foundation of our system involves the input of traffic data into three distinct classifiers: Random Forest. Logistic Regression, and Multi-Layer Perceptron (MLP). Each classifier is designed to independently assess incoming traffic and determine whether it exhibits normal behavior or represents a potential DDoS attack. Upon the completion of the individual assessments, a consensus is reached through a voting mechanism that considers the outputs of all three classifiers. The majority decision is then selected to categorize the incoming traffic as normal or indicative of an attack. This collective decision-making process enhances the robustness and reliability of our DDoS detection system. The three chosen classifiers offer a diverse set of analytical perspectives, leveraging the strengths of each model to provide a comprehensive evaluation of network traffic. Random forest is known for its efficiency in handling large datasets, contributing to its clustering capabilities. Logistic Regression, a widely used statistical method, brings its interpretability and simplicity to the ensemble. Finally, the Multi-Layer Perceptron (MLP), as a form of artificial neural network, captures intricate patterns within the data. An architectural diagram of the system is given in Figure 1.

Our comprehensive strategy not only capitalizes on the unique strengths of each classifier but also harnesses the diversity in their decision-making processes. The collective decision, determined through a majority vote, stands as a resilient and trustworthy marker for pinpointing potential DDoS attacks. Given our emphasis on DDoS detection, we have carefully selected classifiers that prioritize lightweight performance without compromising accuracy.

b) Dataset Overview: In our quest to identify a suitable dataset for training our model on DDoS attack detection in SDN, we explored several datasets available online. Among the datasets considered were the KDD Cup 99 Dataset, UNSW NB15 Dataset, and CIC IoT 2023 Dataset.

Upon thorough analysis and conducting preliminary proof-of-concept (POC) work, we chose the KDD Cup 99 dataset as our primary training dataset. This decision was informed by several factors, including the dataset's comprehensive coverage of network traffic scenarios and its historical significance in the realm of cybersecurity research. The KDD Cup 99 dataset is renowned for its applicability in evaluating intrusion detection systems, making it a fitting choice for our research. Also, the dataset is already labeled which saved us pre-processing time. Additionally, the dataset's structure aligns well with our objectives, allowing us to train our model effectively and assess its performance accurately. Features used for training machine learning models are given in Table I. While training we took 20% data for testing and 80% of total data for training our classifiers.

TABLE I
DESCRIPTION OF SELECTED FEATURES IN KDD CUP 99 DATASET

Feature Name	Description	Type
count	Number of connections to the same host as the current connection in the past two seconds	Continuous
serror_rate	% of connections that have "SYN" errors	Continuous
rerror_rate	% of connections that have "REJ" errors	Continuous
same_srv_rate	% of connections to the same service	Continuous
diff_srv_rate	% of connections to different services	Continuous
srv_count	Number of connections to the same service as the current connection in the past two seconds	Continuous
srv_serror_rate	% of connections that have "SYN" errors for same-service connections	Continuous
srv_rerror_rate	% of connections that have "REJ" errors for same-service connections	Continuous
srv_diff_host_rate	% of connections to different hosts for same-service connections	Continuous
duration	Length (number of seconds) of the connection	Continuous
protocol_type	Type of the protocol, e.g., TCP, UDP, etc.	Discrete
service	Network service on the destination, e.g., http, telnet, etc.	Discrete
src_bytes	Number of data bytes from source to destination	Continuous
dst_bytes	Number of data bytes from destination to source	Continuous
flag	Normal or error status of the connection	Discrete
land	1 if the connection is from/to the same host/port; 0 otherwise	Discrete
wrong_fragment	Number of "wrong" fragments	Continuous
urgent	Number of urgent packets	Continuous

c) Logistic Regression: Logistic Regression is a widely employed statistical method in machine learning for binary classification tasks, making it a valuable tool for our DDoS attack detection model. Unlike linear regression, logistic regression is suitable for predicting binary outcomes, where the dependent variable falls into one of two classes. In our context, it aids in distinguishing between normal and potentially malicious network traffic. Logistic Regression operates by modeling the log-odds of the probability of the event occurring, and its outcome is transformed using the logistic function to ensure it lies within the [0, 1] range, representing probabilities. The logistic regression model can be expressed mathematically as given Equation (1)

$$Y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}} \tag{1}$$

	precision	recall	f1-score	support
0	0.96	0.99	0.98	19466
1	1.00	0.99	0.99	78282
accuracy			0.99	97748
macro avg	0.98	0.99	0.99	97748
weighted avg	0.99	0.99	0.99	97748

Fig. 2. Accuracy metrics of Logistic Regression on KDD99 Dataset

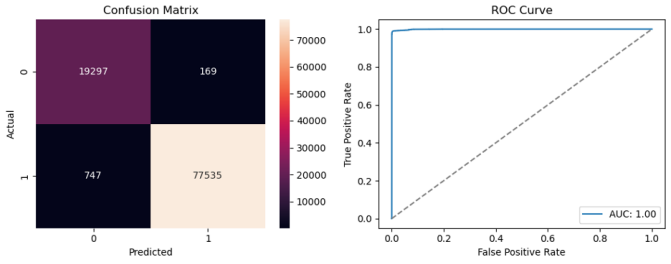

Fig. 3. Confusion metrics of Logistic Regression on KDD99 Dataset

where Y is the dependent variable representing the probability of an event occurring, e is Euler's number (approximately 2.71828), $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are regression coefficients, and $X_0, X_1, X_2, \ldots, X_n$ represent features. The performance metrics of our logistic regression are summarized in Figure 2 and Figure 3.

d) Multi-Layer Perceptron (MLP): MLP is a type of artificial neural network that excels in capturing complex patterns within data, making it well-suited for the intricate nature of network traffic analysis. Unlike logistic regression, MLP is capable of handling non-linear relationships, enabling a more nuanced understanding of the features contributing to the classification of network traffic as either normal or indicative of a DDoS attack. The architecture of an MLP involves multiple layers, including an input layer, one or more hidden layers, and an output layer. The computation of the output in a neural network is carried out through a series of weighted connections and activation functions. The output $\hat{Y}$ is typically obtained using the following Equation (2)

$$\hat{Y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2}$$

Where $\hat{Y}$ is the predicted output, f is the activation function, w_i represents the weights associated with input features x_i, and b is the bias term. In our case, we trained the model using two hidden layers each having five neurons. The performance measure of the model is given in Figure 4 and Figure 5.

The model exhibits high accuracy, a high AUC score, and a high precision score for both normal traffic and DDoS attacks. However, in comparison to other models, it demonstrates a relatively lower recall score for normal attacks. This suggests that some instances of normal attacks may not be identified correctly.

	precision	recall	f1-score	support
0	0.99	0.96	0.98	19466
1	0.99	1.00	0.99	78282
accuracy			0.99	97748
macro avg	0.99	0.98	0.98	97748
weighted avg	0.99	0.99	0.99	97748

Fig. 4. Accuracy metrics of MLP on KDD99 Dataset

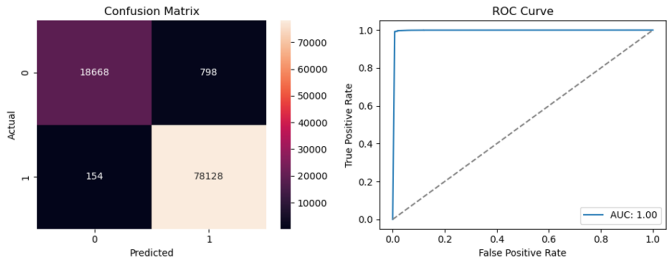

Fig. 5. Confusion metrics of MLP on KDD99 Dataset

e) Random Forest: Random Forest is our third classifier in the DDoS attack detection framework. This ensemble learning algorithm combines the predictions of multiple decision trees, offering improved generalization and mitigating the risk of overfitting. Applying random forest in our dataset we get the following result summarized in Figure 6 and Figure 7.

The model has excellent accuracy, precision, and recall scores on both normal traffic and DDoS attacks, it has balanced performance compared to other models, and this model can distinguish normal traffic and DDoS attacks precisely and ef-

	precision	recall	f1-score	support
0	0.98	0.99	0.98	19466
1	1.00	0.99	1.00	78282
accuracy			0.99	97748
macro avg	0.99	0.99	0.99	97748
weighted avg	0.99	0.99	0.99	97748

Fig. 6. Accuracy metrics of Random Forest on KDD99 Dataset

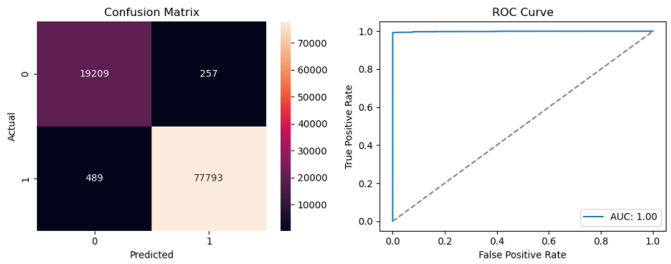

Fig. 7. Confusion metrics of Random Forest on KDD99 Dataset

ficiently. Compared to the aforementioned models, this model achieved the best performance.

B. IoT SDN Implementation

All development of the IoT SDN will occur inside a Mininet VM using Python version 3.8. The Mininet Python API is used to create the SDN topology, add ethernet switches, and run network hosts. The SDN controller is built with the Ryu Python framework and deployed using the *ryu-manager* CLI tool.

a) IoT Communication: Due to the low computing power of IoT devices and how sensitive the data they transfer is, a lightweight and reliable communication protocol is required. Most commonly, the MQTT protocol [29] which runs on top of a TCP connection is used.

MQTT is a messaging protocol designed for IoT devices. It proves a lightweight and efficient publish/subscribe transport and reliable message delivery with 3 defined quality-of-service (QoS) levels: at most once, at least once, and exactly once. MQTT also allows encrypting data using TLS.

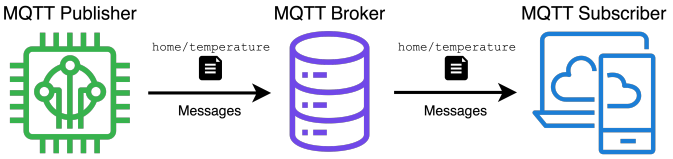

Fig. 8. MQTT architecture diagram

You can see a diagram of a standard MQTT architecture in Figure 8. The MQTT network consists of MQTT publishers, such as IoT devices, transmitting messages to a topic to an MQTT broker. Then, MQTT subscribers will subscribe to that topic and receive the transmitted messages from the broker.

The Eclipse Mosquitto [30] broker application and tools will be used to simulate an MQTT network in this SDN environment. The MQTT broker will run a Mosquitto broker server with a custom configuration allowing all incoming connections. The MQTT publishers and subscribers will use the *mosquitto_pub* and *mosquitto_sub* CLI tools respectively.

b) SDN Implementation: The architecture of the SDN is depicted in Figure 9. The network will consist of four MQTT-enabled hosts and an SDN controller, all connected through a central ethernet hub. The first MQTT host is the central MQTT broker (shown in purple in Figure 9), relaying messages between the MQTT publishers and subscribers. The second host will be an MQTT subscriber (shown in blue) that reads messages from the publisher and outputs them to a file, this host will emulate a server application that processes IoT sensor data. The remaining two hosts (in green) will act as IoT devices (MQTT publishers), they will publish a message to the MQTT broker every second, this message contains the timestamp of when it was generated along with a random sensor reading.

All normal traffic in the network will consist of the MQTT messages passing between these hosts. One of the MQTT

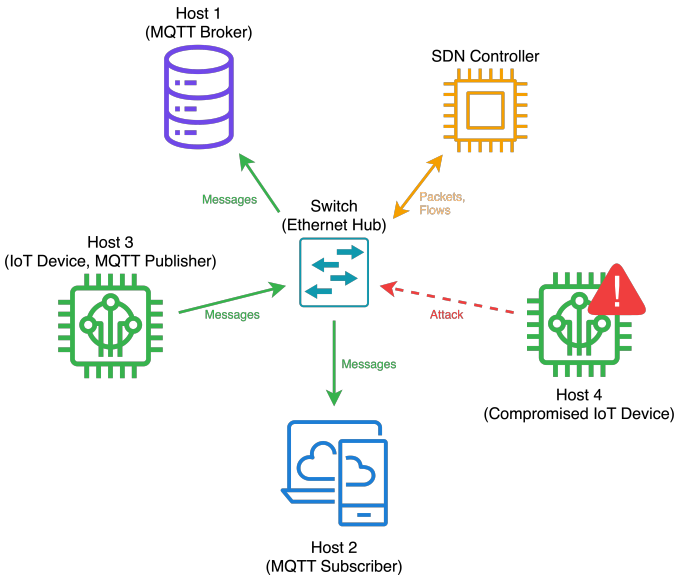

Fig. 9. SDN architecture diagram

publishers, host 4 in Figure 9, will act as a compromised IoT device and will be used to perform an attack on the MQTT broker.

The central OpenFlow-enabled switch will connect all four hosts and the SDN controller. During initialization, the SDN controller will add a flow-entry into the switch instructing it to forward all frames to the controller. When the controller receives a frame from the switch, it will process the TCP segment inside the frame and instruct the switch to flood the frame out of all of its ports. This makes the switch behave as a simple ethernet hub.

c) Monitoring the SDN: The SDN controller will analyze all IPv4 TCP segments it receives from the ethernet switch to detect intrusions in the network. First, the TCP segments need to be grouped into TCP connections. A single TCP connection is uniquely identified by the source IPv4 address, source port, destination IPv4 address, and destination port. Incoming TCP segments will be mapped to a TCP connection based on their source and destination address and port. To determine when a TCP connection begins and ends, TCP segment flags will be analyzed and the TCP connection will flow through 7 different states.

A diagram of the seven TCP connection states is shown in Figure 10. These states contain the expected TCP segment flags throughout the entire TCP connection lifetime. This includes the sender initiating the connection with a SYN flag, and later either side ending the connection by sending a FIN flag.

In the diagram, the sender on the left is an IoT device, and the receiver is an MQTT broker on the right. The first state is the sender initiating the connection by transmitting a TCP segment with the SYN flag set. Then, in the second state, the receiver responds to the connection request with a SYN/ACK segment. The sender confirms by responding with an ACK

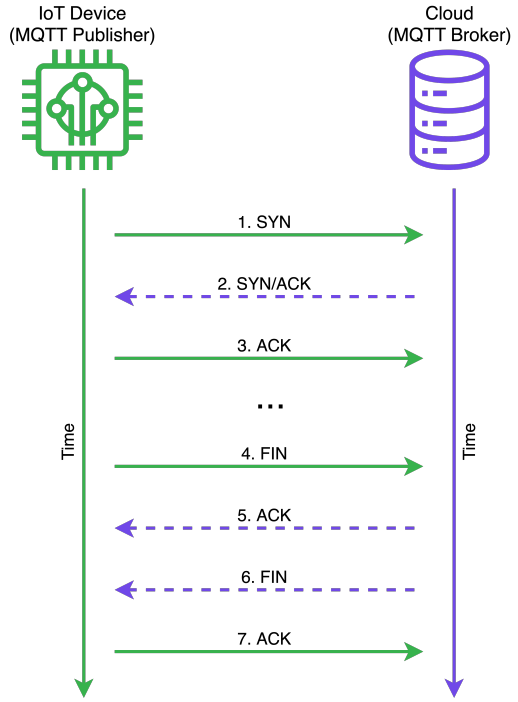

Fig. 10. SDN architecture diagram

segment of its own, putting the connection in state three. After this, normal communication occurs between the sender and receiver, and the state remains at three. To gracefully end the connection one side sends a FIN segment, which puts the TCP connection in state four. In the diagram above it is the sender that initiates closing the connection. Then the receiver acknowledges with an ACK flag (state five) and later sends a FIN flag of its own (state six). The sender then sends a final ACK flag, closing the TCP connection and moving it to state seven.

The SDN controller will monitor whether TCP connections flow through these exact states. Each TCP connection will have an invalid segment counter that increments whenever one of its TCP segments contains an invalid flag combination or attempts to transition to invalid states. Also, a TCP connection can be abruptly closed by either side sending a segment with an RST flag set. If this is encountered, the TCP connection will be in an error state. The controller will also calculate the total size of data transmitted for all TCP segments in a connection.

d) Performing Intrusion Detection and Mitigation: To perform intrusion detection, closed TCP connections are added to a double-ended queue to process them in a two-second window. Before any analysis, TCP connections older than two seconds are removed from the window, this ensures old connections do not influence the classifier results while bounding the window memory usage. The controller will then calculate individual and aggregate metrics for all TCP connections in the window. These metrics match the inputs of the classifiers as described in Table I. Each one of the classifiers will be used to predict whether the host initiating the TCP connection is performing an attack. If the majority of the classifiers identify a possible attack, the compromised host's IPv4 address is blacklisted. Any frames received from a blacklisted host are discarded and not forwarded by the switch.

e) Simulating an Attack: For this IoT SDN, hping3 is used to send a SYN flood attack from host four to host one. This attack consists of sending 20 TCP segments with the SYN flag set containing 120 bytes of data. These packets are sent at a rate of 10 packets a second.

f) Software Structure: All code implemented in the system is structured into separate classes. The main class is the *Controller* class that implements our SDN controller. This class extends the general Ryu application class, *RyuApp*, with additional intrusion detection and mitigation functions. The controller communicates with the Ethernet switch using Ryu by sending and receiving OpenFlow messages. First, the controller creates an event handler to configure switch features. This event handler is executed by Ryu when the switch is initialized and instructs the switch to route all packets to the SDN controller. When an SDN controller receives a packet from the switch, it will first analyze it and then send an OpenFlow message to the switch instructing it to flood the packet out of all of its ports.

The analysis first occurs inside the SDN controller by grouping TCP segments into TCP connections as described in the monitoring the SDN section. Closed TCP connections are added to a second class, *TcpConnWindow*. This class is used to store a 2-second window of TCP connections and compute statistics about individual connections in the window. At the SDN controller, a separate thread is created to continuously monitor the TCP connections in the window. Every two seconds, the controller will retrieve statistics from the *TcpConnWindow* class and perform intrusion detection using the third class of the system, the *Detector* class.

This class contains three previously fitted classifiers, random forest, logistic regression, and multi-layer perceptron. The process used to fit the classifiers is described in the intrusion detection section system section. The *Detector* class will first transform the TCP connection statistics into a vector that is compatible with the classifiers. Then each classifier is used to predict whether the statistics indicate a possible attack. If more than one classifier returns a positive result, then the outcome of the detection process is positive, that the TCP connection whom the statistics are about is malicious, and the host originating the connection has been compromised.

Finally at the controller, if the detection process yields a positive result, the compromised host's IPv4 address is blacklisted. To blacklist a host, its IPv4 address is added to a set and any future packets whose source IPv4 address is included in the set are not forwarded by the switch.

VI. Experimental Results and Analysis

The software-defined network was successfully set up incorporating the ML model for intrusion detection.

After performing a SYN flood attack the controller is able to detect the intrusion and start the mitigation process. Figure 11

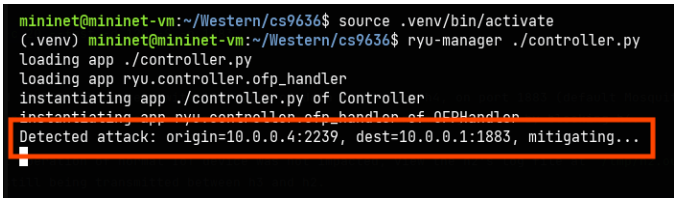

Fig. 11. SDN controller output after SYN flood attack

shows the output from the SDN controller after the attack occurs. In the output, the correct compromised host with an IPv4 address of 10.0.0.4 is identified. Also, the controller correctly outputs the attack target host which has the IPv4 address of 10.0.0.1.

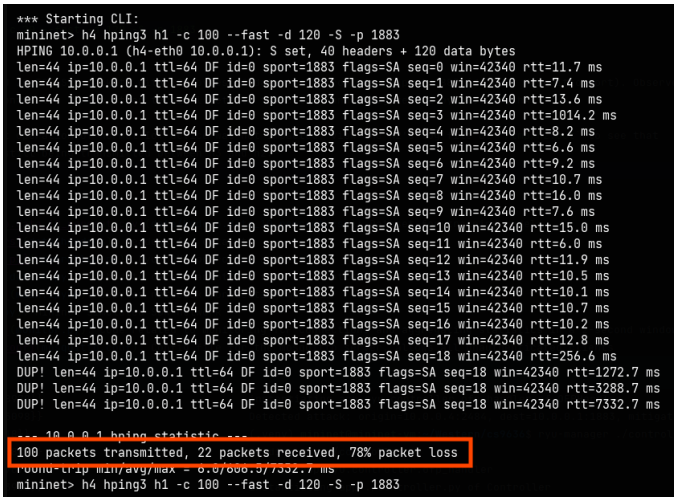

Fig. 12. Output from hping3 after first SYN flood attack

Figure 12 shows the output of the hping3 tool after performing the SYN flood attack. Only 22 packets from the attack make it to the target host. The remaining 78 packets are blocked. It takes a total of 2.2 seconds for the SDN controller to mitigate the SYN flood attack.

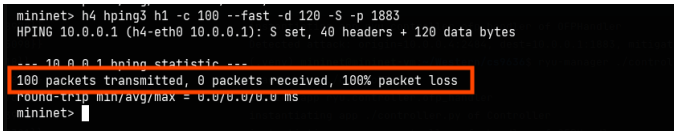

Fig. 13. Output from hping3 after first SYN flood attack

After running the SYN flood attack a second time, all the packets from the compromised host are discarded indicating that the intrusion detection and mitigation was successfully completed. Figure 13 shows the output of hping3 after performing the same SYN flood attack a second time, where all transmitted packets are dropped with a packet loss of 100%. To verify that normal operation of the IoT devices was not impacted by the attack or the mitigation process, the output of the MQTT subscriber was monitored to ensure that messages were still being transmitted between the IoT device and the subscriber through the MQTT broker. The IoT device was configured to include a timestamp in each message it generated. Once the messages arrived at the subscriber, the timestamps were compared to the time at which the attack occurred and, as expected, messages were still being transmitted to the subscriber during and after the SYN flood attack.

VII. Limitations and Future Work

Although the proposed project was successful, the system has a few limitations. Firstly, the network routing, intrusion detection and mitigation all occur on the SDN controller. This means that if the SDN is compromised or overwhelmed, the performance of the entire system would be impacted. Ideally, the intrusion detection should take place on a separate server, away from the SDN controller.

Secondly, having only one SDN controller means that there is a single point of failure. If the SDN controller is unresponsive, the network would not be protected and frames would not be routed. Balancing the load of the SDN controller across multiple servers would add redundancy to the system, improving the reliability and availability of the network.

Lastly, although the Ryu SDN framework is versatile and allows quick prototyping, it is not used for production environments. Adapting the intrusion detection and mitigation system to use a more sophisticated SDN framework such as OpenDaylight created by the Linux Foundation would yield a more robust system with better performance and features.

For the intrusion detection component, one of our main goals for the future is to expand the types of cyber attacks our model can detect. Right now, we're focusing on DDoS attacks, but we want to test our model with different datasets like UNSW NB15 and CIC IoT 2023 to make it better at spotting a variety of threats.

Currently, our model uses supervised learning, where it learns from labeled examples. However, in the future, we're considering trying unsupervised learning. This approach allows the model to learn on its own without specific instructions. We think this could help our system understand new types of threats we haven't seen before, making it even more effective at keeping our systems safe from cyber risks.

Also, the intrusion detection system only handles IPv4 TCP segments, limiting the type of attacks it can detect. Moving forward, the system needs to be adapted to handle various other attack types by analyzing different internet and transport protocols. For example, by processing UDP datagrams the system would be able to identify DNS tunneling attempts. Handling the ICMP protocol would enable the detection of ICMP-related attacks such as ICMP ping or echo-request flood attacks.

VIII. Conclusion/Summary

In conclusion, the system was able to accurately detect and swiftly mitigate a SYN flood attack on one of its hosts. By conducting the detection process at the SDN controller rather than at the IoT device, MQTT broker or subscriber, the attack

detection does not negatively impact the performance of hosts in the network. Additionally, there is no need to make any modifications to the applications running inside the network to take advantage of the intrusion detection and mitigation system.

REFERENCES

[1] K. Ashton, "That 'internet of things' thing," *RFID Journal*, 2009.

[2] E. Fleisch, "What is the internet of things? an economic perspective," *Economics, Management, and Financial Markets*, 2010.

[3] L. Research, "The rise of the internet of things," 2013.

[4] J. P. Leenen, C. Leerentveld, J. D. van Dijk, H. L. van Westreenen, L. Schoonhoven, and G. A. Patijn, "Current evidence for continuous vital signs monitoring by wearable wireless devices in hospitalized adults: systematic review," *Journal of medical Internet research*, vol. 22, no. 6, p. e18636, 2020.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[6] M. Levy, "Understanding the real energy consumption of embedded microcontrollers." https://www.digikey.ca/en/articles/understanding-the-real-energy-consumption-of-embedded-microcontrollers, 2012.

[7] G. A. Nikitha, G. J. W. Kathrine, C. R. Duthie, V. Ebenezer, and S. Silas, "Hybrid cryptographic algorithm to secure internet of things," in *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1556–1562, IEEE, 2023.

[8] A. Rayes and S. Salam, *The Things in IoT: Sensors and Actuators*. Springer, 2019.

[9] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *2016 IEEE 24th International conference on network protocols (ICNP)*, pp. 1–5, IEEE, 2016.

[10] Q. Shafi, A. Basit, S. Qaisar, A. Koay, and I. Welch, "Fog-assisted sdn controlled framework for enduring anomaly detection in an iot network," *IEEE Access*, vol. 6, pp. 73713–73723, 2018.

[11] M. Šarac, N. Pavlović, N. Bacanin, F. Al-Turjman, and S. Adamović, "Increasing privacy and security by integrating a blockchain secure interface into an iot device security gateway architecture," *Energy Reports*, vol. 7, pp. 8075–8082, 2021.

[12] K. Giotis, G. Androulidakis, and V. Maglaris, "Leveraging sdn for efficient anomaly detection and mitigation on legacy networks," in *2014 Third European Workshop on Software Defined Networks*, pp. 85–90, IEEE, 2014.

[13] J. Ashraf, N. Moustafa, A. D. Bukhshi, and A. Javed, "Intrusion detection system for sdn-enabled iot networks using machine learning techniques," in *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pp. 46–52, IEEE, 2021.

[14] J. Bhayo, S. A. Shah, S. Hameed, A. Ahmed, J. Nasir, and D. Draheim, "Towards a machine learning-based framework for ddos attack detection in software-defined iot (sd-iot) networks," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106432, 2023.

[15] S. Wang, K. Gomez, K. Sithamparanathan, M. R. Asghar, G. Russello, and P. Zanna, "Mitigating ddos attacks in sdn-based iot networks leveraging secure control and data plane algorithm," *Applied Sciences*, vol. 11, no. 3, 2021.

[16] M. Al-Fayoumi and Q. Abu Al-Haija, "Capturing low-rate ddos attack based on mqtt protocol in software defined-iot environment," *Array*, vol. 19, p. 100316, 2023.

[17] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[18] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and communication networks*, vol. 9, no. 18, pp. 5803–5833, 2016.

[19] J. Rischke and H. Salah, "Software-defined networks," in *Computing in Communication Networks*, pp. 107–118, Elsevier, 2020.

[20] S. Thakare and M. Pund, "Software defined network: Comprehensive study," in *Proceedings of Integrated Intelligence Enable Networks and Computing: IIENC 2020*, pp. 603–611, Springer, 2021.

[21] Python Software Foundation, "What is Python? Executive Summary — python.org." https://www.python.org/doc/essays/blurb/.

[22] Mininet Project Contributors, "Mininet Overview - Mininet — mininet.org." https://mininet.org/overview/.

[23] Mininet Project Contributors, "GitHub - mininet/mininet: Emulator for rapid prototyping of Software Defined Networks — github.com." https://github.com/mininet/mininet.

[24] Open Networking Foundation, "MININET - Open Networking Foundation — opennetworking.org." https://opennetworking.org/mininet/, 2023.

[25] Ryu SDN Framework Community, "Ryu SDN Framework — ryu-sdn.org." https://ryu-sdn.org/.

[26] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki, "Ryu sdn framework-open-source sdn platform software," *NTT Technical Review*, vol. 12, 08 2014.

[27] NAKIVO Team, "A Complete Guide to Using VirtualBox on Your Computer — nakivo.com." https://www.nakivo.com/blog/use-virtualbox-quick-overview/, 2023.

[28] S. Sanfilippo, "hping3 - kali linux." https://www.kali.org/tools/hping3/, 2022.

[29] "Mqtt - the standard for iot messaging." https://mqtt.org, 2022.

[30] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.