Flood Control: TCP-SYN Flood Detection for Software-Defined Networks using OpenFlow Port Statistics

Tapadhir Das, Osama Abu Hamdan, Shamik Sengupta, and Engin Arslan
Department of Computer Science and Engineering, University of Nevada, Reno, USA
Email: tapadhird,oabuhamdan@nevada.unr.edu, ssengupta,earslan@unr.edu

***Abstract*—As software-defined network (SDN) adoption increases, it becomes increasingly important to develop effective solutions to defend them against cyber attacks. A prominent cyberattack that can compromise SDNs is TCP-SYN floods, which can exhaust network resources by initiating too many fraudulent TCP connections. Previous efforts to detect SYN Flood attacks mainly rely on statistical methods to process mirrored traffic or flow statistics. Thus, they either incur high overhead (in the case of port mirroring) or lead to low accuracy (in the case of using flow statistics). In this paper, we propose a machine learning (ML)-enabled TCP-SYN flood detection framework using Openflow port statistics. We demonstrate that ML models such as Random Forest classifiers can differentiate normal traffic from SYN flood traffic with up to 98% accuracy. We also introduce a novel threat localization technique that can pinpoint where the attack traffic originates from in the network.**

***Index Terms*—Software-defined networking, anomaly detection, TCP-SYN floods, DDoS detection**

I. Introduction

Since the original ARPANET, networks have become permanently embedded within the fabric of computer communications. Traditional computer networks were developed to disperse intelligence across the physical devices in the topology, like routers and switches. These architectures consisted of both a control plane, which administered configuration of the nodes and path programming, and a data plane that assisted with packet forwarding. As innovation progressed, networking applications became more diverse, dynamic, interconnected, and geographically distributed like the Internet of Things (IoT), vehicular networks, and smart grids. Increasing diversity along with increasing complexity hindered the management of networks due to complicated synchronization problems and complex implementation issues [1].

To address these problems, software-defined networks (SDN) were introduced. SDNs decouple the data and control planes, so a logically centralized controller can oversee routing decisions for all network devices in a network. The SDN controller can facilitate network policy upgrades, alterations, and monitoring due to having the ability to inject flow tables in network switches that control network operations and policies [2]. This consolidated architecture also enables the ability to perform network data analysis and mining to improve efficiency or provide security as large networks can be monitored.

This research is supported by the National Science Foundation (NSF) Award #2019164.

978-1-6654-9952-1/22/$31.00 ©2022 IEEE

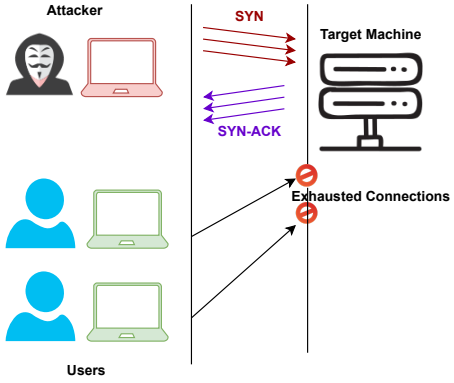

Fig. 1: TCP-SYN flood attack

Cyber attacks such as Distributed Denial of Service (DDoS) threaten the healthy operation of SDN networks. The number of global DDoS attacks is expected to reach 15.4 million by 2023 [3]. The main goal of a DDoS attack is to consume and exhaust the resources, like bandwidth and memory, of a target machine, thereby preventing it from serving legitimate and legal requests. An eminent DDoS attack against SDNs is the TCP-SYN flood, which takes advantage of the Transmission Control Protocol (TCP) handshake mechanism, by not returning the final ACK packet to the target node. Attackers keep sending TCP SYN packets to the target, thereby consuming the target device's resources [4] as illustrated in Figure 1. This type of attack can be catastrophic for organizations and their SDN infrastructures as they can risk the loss of revenue, reputation, and intellectual property. The problem can be exacerbated if TCP-SYN flood attacks occur on SDNs associated with national critical infrastructures like the power grid, transportation systems, and energy sectors.

Network administrators can monitor and manually identify the existence of TCP-SYN floods on a network. However, such manual solutions are not feasible as attacks could cause significant damage before they are detected. Thus, automated solutions are key to the uninterrupted operation of critical SDN infrastructures. Anomaly detection methods can be placed in the logically centralized SDN controller to identify malicious traffic using network metrics that network devices report to the controller periodically [5].

Due to the significance and prevalence of TCP-SYN floods on an SDN infrastructure, it has received considerable attention from researchers, especially using machine learning(ML)-based anomaly detection. However, existing solutions to detect TCP-SYN attacks have few issues. First, they utilize established SDN anomaly and intrusion detection datasets [6] [4], which are collected from fixed network topologies, that might be different than most SDN topologies in terms of users, nodes, devices, geographical spread; all of which can vastly affect network metrics. Second, they mostly rely on flow statistics, which again can limit the transferability of solutions to different networks since the flow statistics depend on network topology and traffic characteristics. For example, the original DARPA dataset provides artificially high feature values compared to real traffic data, as it was simulated in a military network environment. The same problem exists in other datasets that are derived from DARPA such as NSL-KDD. Similarly, the open-sourced datasets like the UNSW-NB15 dataset contain multiple flow features like IP addresses, protocols, TCP/IP headers, payload information, recorded start time, SYN Flag, and ACK Flag counts. These may end up as redundant features that play no impact in being able to detect potential TCP-SYN flood attacks. Depending on the approach being used to detect TCP-SYN floods, these features can also increase the cardinality of the dataset, leading to increased training and inference times. Finally, a common problem faced by many open-sourced datasets is that of class imbalance. In general, many of these SDN datasets just do not have enough data samples that can model certain anomaly types effectively. This leads to poor performance metrics when trying to detect these classes. Hence, training a TCP-SYN flood attack detector using these datasets will hinder the effectiveness of the models in production networks, putting organizations at risk against cyber threats [7]. Addressing the above limitations is crucial to ensuring that organizations can generate, train, and deploy their detectors to identify TCP-SYN flood attacks, and not rely on open-sourced datasets.

In this paper, we propose a TCP-SYN flood detection framework using an SDN simulation environment, that addresses the above-mentioned limitations. Organizations can gear the simulation environment and record metrics towards their topology, instead of relying on using the recorded metrics derived from static topologies from the SDN datasets. This can provide better performance and security. We, also, investigate different network statistics to conduct attack detection. Flow statistics may contain many redundant features and can increase cardinality, leading to higher training, and inference times. Another option is to utilize port-level statistics instead of flow statistics where we capture statistics from every single SDN port in the infrastructure. Yet, this can also be an issue as certain port statistics are reported as a cumulative value, affecting the accuracy of the models.

Our proposed approach focuses on delta/differential port statistics. Differential port statistics refer to the change in magnitude of observed statistics in switch ports in a time interval. Differential port statistics can provide a more fine-grained analysis of network flows from the port level. Normal port statistics may get aggregated over time and can have high magnitudes for the feature values. This can lead to lower ranges over the magnitude scale, between the feature magnitudes during normal and attack circumstances. This can induce instances of attack to be misclassified as normal. Using differential port statistics, this range can be higher over the magnitude scale and can lead to more accurate performance and faster identification. This can allow us to rapidly capture and identify potential TCP-SYN flood attacks from the network traffic before they cause harm. To avoid the class imbalance problem, we gather an equal number of logs for both normal and malicious traffic.

Our main contributions proposed in this work are:

- Setting up an SDN simulation environment that each organization can gear towards their topology.
- Capturing various differential port statistics under both normal and attack conditions to model both circumstances, while minimizing class imbalance by ensuring an equal representation.
- Training an ML model on our captured data to generalize both conditions.
- Testing the performance of our approach towards real-time detection of TCP-SYN floods.

The rest of the paper is structured as follows: Section II provides the literature study and related work. Our methodology is provided in Section III. Section IV demonstrates our simulation, results, and analysis. Finally, conclusions are drawn in Section V.

II. Related Work

Detection of infrastructure level SDN threats like DDoS attacks continues to be a point of interest in the research community. There have been multiple approaches that have been attempted to combat the threat of DDoS. The authors in [8] and [9] proposed source-based mechanisms, where the mechanisms were placed near the source of the attack. In [8], the work aimed to detect DDoS floods by monitoring the ingress and egress of a source network and comparing it with predefined normal flow models. The researchers in [9] proposed MULTOPS which is a heuristic and a data structure that network devices at the source subnet can use to detect and filter DDoS flooding attacks. The disadvantage of source-based methods is that it can be difficult to differentiate between normal and DDoS flows due to insufficient volumes of traffic at the source [10]. Other studies proposed destination-based solutions, where mechanisms are placed at destination hosts or victims [11] [12]. The work in [11] proposed a packet marking mechanism called RIM to detect the path of attack traffic which can help distinguish it from legitimate traffic after detection. Authors in [12] proposed a hop-counter filtering mechanism where information about a source IP address and its hops from the destination are recorded in a table when the destination is working normally. If the destination gets attacked, the victims look at incoming packets and determine the source from the previously logged source and corresponding

hops. The main shortcoming with destination-based methods is that they cannot accurately detect and respond to the attack before it reaches the victim [10].

Some works investigated the usage of network-based methods to detect DDoS threats, where detection and response methods are deployed at network devices like routers and switches [13] [14]. In [13], the researchers proposed a route-based packet filtering method as an expansion to ingress filtering at the core of the network. The work in [14] proposed Watchers that detected misbehaving routers that launched DDoS attacks by absorbing, discarding, and misrouting packets. Network-based techniques suffer from high storage and processing overhead at the core devices, which can cause deficient performance [10]. Detection of TCP-SYN flood attacks has also received attention as a prominent type of DDoS attack that affects networks [15] [16]. In [15], the authors presented SAFETY that provided early detection of TCP-SYN floods by harnessing the programming and wide visibility approach of SDN with entropy method to determine the randomness of flow data. The experiments in this work were conducted on only singular destination victims, which is not always the case in TCP-SYN attacks. [16] proposed AEGIS that detected and mitigated SYN floods against the SDN controller by regularly checking if there is a performance lag in the controller during floods. This method can detect SYN floods after the controller performance drops, by when significant damage could have already been done to the controller.

To facilitate faster detection of TCP-SYN flood attacks, the usage of ML methods has also been investigated [17] [4]. In [17], the researchers propose a TCP-SYN flood mitigation technique using a K-nearest neighbor algorithm for SDN. The authors in [4] propose an anomaly detection model using isolation forest and k-means clustering. The limitation of these works is that they have conducted experiments on open-sourced and customarily used datasets. That limits the topology upon which these methods can be useful. Organizations will not be able to cater these trained models to their SDN infrastructures, as it can lead to sub-optimal performance and put the organization at risk. Also, most of these datasets utilize flow-level statistics. Many of these data features are redundant, leading to high cardinality, training, and inference times.

III. Methodology

Our proposed approach primarily consists of using an SDN simulation environment to build custom topologies that each organization can cater to their use case. Network flows are simulated to showcase the appropriate functionality of the generated network. During these flow simulations, differential port statistics are collected, which form the basis of modeling normal behavior in the simulated SDN. Following, a TCP-SYN flood attack is launched that targets a particular host in the network. The same network metrics are collected under this scenario. The accumulated network metrics form the dataset to train an ML model in an offline manner. The trained ML model can then be placed in the SDN controller, equipped to detect TCP-SYN floods in real-time. Figure 2 illustrates our proposed framework.

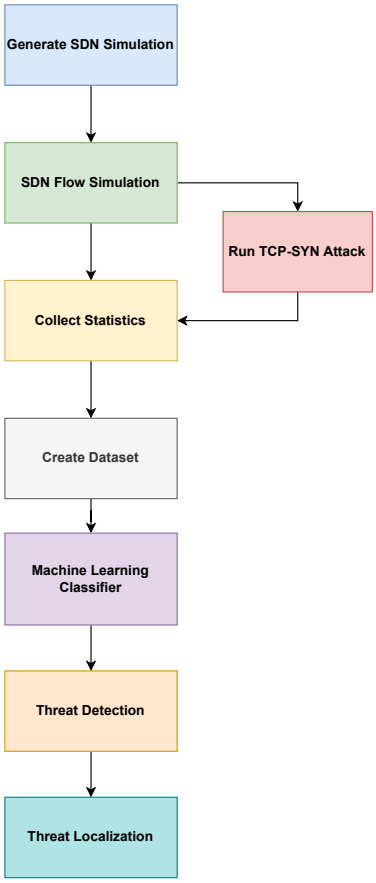

Fig. 2: Proposed TCP-SYN detection framework

A. SDN Simulation Generation

To set up the simulation environment we used Open Network Operating System (ONOS) SDN controller (API version 2.5.0) alongside Mininet. ONOS uses Open Service Gateway Initiative (OSGi) service component at runtime for the creation and activation of components and for auto-wiring components together, which makes it easy to create and deploy new user-defined components without altering the core components. Mininet creates a realistic virtual network, and runs a real kernel, switch, and application code, on a single machine which creates a realistic testbed environment. We also created our ONOS application (component) to collect the needed statistics. We were able to gather differential and cumulative port, flow entry, and flow table statistics for each connected Open vSwitch in the Mininet topology. We created a custom Mininet topology using Mininet API (version 2.3.0) with Open Flow (OF) 14 protocol deployed to the switches.

B. SDN Flow Simulation

IPerf is a tool for testing, measuring, and simulating network functionalities. It is used to create TCP and UDP data streams simulating network flows in virtual and real networks with pre-chosen source and destination IPs and with a dummy

TABLE I: Differential port statistics collected for every port on every switch

Differential Port Statistic	Description
Received Packets	Number of packets received by the port
Received Bytes	Number of bytes received by the port
Sent Packets	Number of packets sent by the port
Sent Bytes	Number of bytes sent
Port alive Duration	The time port has been alive in seconds
Packets Rx Dropped	Number of packets dropped by the receiver
Packets Tx Dropped	Number of packets dropped by the sender
Packets Rx Errors	Number of transmit errors
Packets Tx Errors	Number of receive errors

payload. By using the Mininet API with IPerf, we created a Python script to simulate realistic network flows. Every 5 seconds, we created a flow between a randomly chosen source-destination host pair with a bandwidth of 10 Mbps that lasted for 5 seconds. Hence, we identify the total number of normal flows by N. This represents the number of simulated flows under normal working conditions of the simulated SDN and not under any attack.

C. Collect Statistics

As mentioned in Section III-A, we created a custom application to collect and log the available statistics that get polled on a configured interval (5 seconds) from OF switches. We use differential port statistics which were collected by the message exchanging of OFPPortStatsRequest and OFPPortStatsReply and computed on the controller side by taking the difference between the last two collected data instances. We created a key-value map of this data by gathering it from the data storage service, using the "Device Service" API provided by ONOS. After this, we logged the map of the collected statistics to a Javascript Object Notation (.json) file with a name N_i*.json*. Table I shows the collected statistics and their descriptions per port on every switch in the simulated SDN. These differential port statistics are chosen as they are customarily available in most SDN setups. Also, in the case of any kind of DDoS threat like TCP-SYN, metrics that get influenced by byte/packet transmission along with network throughput are key in detecting if a DDoS attack is actively exhausting network resources.

Additionally, we also collect other network metrics as shown in Table II. These metrics provide the real-time state of the OF ports within the switches of the SDN. Also, they are contemporary measures that can be collected from any SDN setup, including any SDN simulation environment like the one utilized in our proposed framework. We denote the dataset of collected differential port statistics, under normal conditions as X_N. The total number of samples in X_N can be symbolized as E. Here, x_{n_i} represents a single data sample and $x_{n_i} \in X_N, i = \{0, 1.....E\}$. All the samples in X_N are labeled as *normal*.

D. TCP-SYN Attack

To capture differential port statistics during a threat, we launch a TCP-SYN flood attack on the network. This attack is launched with a particular host machine as the intended victim.

TABLE II: Additional statistics collected for anomaly detection

Statistic	Description
Connection Point	Network connection point expressed as a pair of the network element identifier and port number
Total Load/Rate	Obtain the current observed total load/rate (in bytes/s) on a link
Total Load/Latest	Obtain the latest total load bytes counter viewed on that link
Unknown Load/Rate	Obtain the current observed unknown-sized load/rate (in bytes/s) on a link
Unknown Load/Latest	Obtain the latest unknown-sized load bytes counter viewed on that link
Time seen	When the above-mentioned values were last seen
is_valid	Indicates whether this load was built on valid values

The compromised machines that launch the attack, from here on referred to as attackers, can be any of the remaining host machines in the simulated SDN topology. Let the total number of attack flows by A. This represents the number of simulated flows under attack conditions of the simulated SDN. To create minimal variance and class imbalance, we ensure that $A = N$.

Let the dataset of collected differential port statistics, under attack conditions, be denoted as X_A. The total number of samples in X_A can be symbolized as F. Here, x_{a_i} represents a single data sample and $x_{a_i} \in X_A, i = \{0, 1, ..F\}$. All the samples in X_A is labeled as *attack*.

E. Dataset Creation

After collection of the differential port statistics, we must combine both datasets N and A into one dataset X, where $X = \{X_N, X_A\}$. This combination ensures that all data is aggregated into one source, from which the ML algorithm can begin learning.

F. ML Classifier

The ML classifier is utilized to perform predictions on network flows, to see if the SDN infrastructure is under attack. For this ML procedure, X gets separated into the X_{train} and X_{test} datasets. We denote the total number of samples in X_{train} as Y, while the total number of samples in X_{test} is represented by Z and $Y > Z$. Let the ML classifier be represented by μ. The classifier is trained on X_{train}, to obtain a model ν, that is saved:

$$\nu = \mu(X_{train}) \tag{1}$$

Once the ML classifier is trained, the values in X_{test} are parsed through to obtain the accurate prediction of the network flow γ:

$$\gamma = \nu(X_{test}) \tag{2}$$

The main objective behind running the testing dataset X_{test} through the trained ML classifier ν is to obtain performance metrics like precision, recall, and F-Measure scores only. The primary objective of an anomaly detector for an SDN

environment is to rapidly parse through real-time network traffic. This traffic does not have any labels associated with the data samples. Hence, it is entirely contingent upon the trained anomaly detector ν to decipher between normal and attack traffic in an online setting.

G. Threat Detection

This is a primary component of the proposed TCP-SYN flood detection framework. As we do not collect any flow-level metrics for classification, we do not have access to any host-level information like IP addresses. So, to accurately detect TCP-SYN floods and where they are originating from on the network, we depend on the Threat Detection and Threat Localization modules. These modules will also be tested on real-time traffic that have no predicted and ground-truth labels associated with it.

For Threat Detection, let the list of real-time flows be U, and the total number of flows be denoted with V. Here, u_i represents a single flow and $u_i \in U, i = \{0, 1, ..V\}$. Within every single flow u_i, multiple switches can be denoted as u_{i_s}. Within each switch u_{i_s}, some ports predict, using ν, if that flow is normal or under attack. We assume the total number of switches per flow is denoted as P_i. Let the total number of ports per-flow u_i be W_i, and list of ports for each flow u_i get denoted by $u_{i_{s_p}}$ such that $u_{i_{s_p}} \in u_i, p = \{0, 1, ...W_i\}$.

First, we must detect the number of ports that are classifying a flow as under attack. That can be determined by running each port and its corresponding differential port statistics through the pre-trained classifier to obtain their prediction ω_p:

$$\omega_p = \begin{cases} 0 & \nu(u_{i_{s_p}}) == Normal, \\ 1 & \nu(u_{i_{s_p}}) == Attack, \end{cases} p = \{0, 1, ...W_i\} \quad (3)$$

The predictions of the flow then get summed up, denoted as η_p:

$$\eta_p = \sum_{0}^{W_i} \omega_p \quad (4)$$

We introduce a hyperparameter called *port threshold*, denoted as Φ. Φ can be defined as a threshold ratio where any value over this ratio infers that the SDN is under a potential attack. Φ allows us to possess more control over the threat detection approach by acting as a control variable. This value is determined by the system administrator(s) who oversee the SDN infrastructure and security, and the magnitude of Φ is determined by the important parameters in the SDN topology like the number of hosts, switches, links, and ports. We see from previous equations that, for every single flow, a decision is made within every single port from all switches that the flow passes through. The port classifies, based on observed differential port statistics, if that flow is undergoing an attack using the trained ML model ν. The SDN is categorized as undergoing a potential TCP-SYN flood attack if the ratio of ports that classify the flow as an attack to the total number of ports that the flow passed through is greater or equal to Φ. Hence, a flow is classified as under attack if:

$$\omega_p = \begin{cases} Attack & \eta_p < \Phi, \\ Normal & \eta_p \geq \Phi, \end{cases} p = \{0, 1, ...W_i\} \quad (5)$$

Using Equation 5, we can classify if a network is undergoing a TCP-SYN flood attack currently.

H. Threat Localization

For threat localization, we introduced another hyperparameter called the *switch threshold* denoted as Θ. Θ allows us to possess more control over the threat localization approach by acting as a control variable and is responsible for recognizing the switches where at least $\Theta + 1$ ports are classifying the flow as under attack. This value is also selected by the system administrator(s) in charge of the SDN infrastructure and security, and the magnitude of Θ is determined by the important parameters in the SDN topology like the number of hosts, switches, links, and ports. Every SDN has its unique topology. But the basic building blocks will primarily be the same. There will be various hosts that communicate with each other using network switches. The main goal of Θ is to accumulate the total list and frequency of flagged switches using the trained ML model ν. The primary goal will be to localize the switch that is flagged most often, as the malicious hosts performing the TCP-SYN flood will be connected to that switch. Finding the flagged switches from a network flow can be accomplished by:

$$\zeta_s = \begin{cases} Attack & \sum_{0}^{p} \eta_p > \Theta, \\ Normal & \sum_{0}^{p} \eta_p \leq \Theta, \end{cases} \quad (6)$$

where ζ_s represents if a switch has flagged the flow and where $p = \{0, 1..W_i\}, s = \{0, 1..P_i\}$. Equation 6 allows us to detect all the switches that flag a flow as under ongoing attack. The main goal here is to find all the flagged switches to pinpoint which switches are passing attack flow through its ports. This can be used to localize the hosts in the network that are malicious. To perform the final localization of the switches to find out which hosts are malicious, we must compute the frequency with which the switches get flagged. The switch that is flagged the most ψ, will be directly connected to the malicious hosts that are performing the TCP-SYN flood attack. This can be detected by:

$$\psi = \arg\max_{(} \sum_{0}^{V} \zeta_s), s = \{0, 1, ..P_i\} \quad (7)$$

Using the above methodology, we can perform anomaly detection to detect ongoing TCP-SYN flood attacks in the SDN. Additionally, we are also able to localize the attackers by flagging the closest switch that the attackers are connected to. The proposed method employs the usage of differential port statistics, which are customarily available in most normal networking setups. For our experimental purpose, we are emphasizing on SDNs due to the wide relevance and usage of

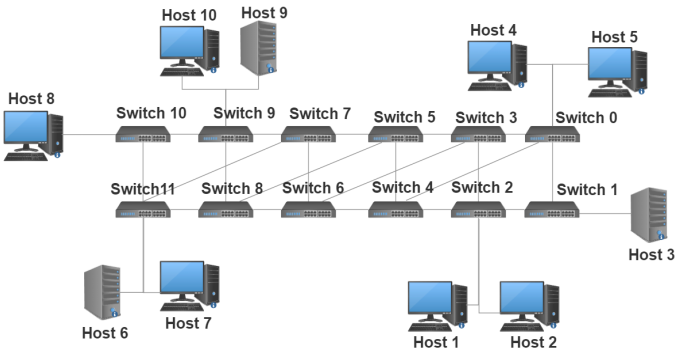

Fig. 3: Simulated SDN Topology

SDN architectures across industries, organizations, and critical infrastructures over normal networking architectures. That is why our experimentation and results are conducted using an SDN and metrics collection is regulated using OpenFlow statistics. However, this research and the proposed method can be expanded to other modern computer network architectures like SD-WAN, SASE, or other traditional networks.

IV. Experimentation and Results

A. Setup

The simulated topology for our experimentation can be seen in Figure 3 which consists of 10 hosts and 12 switches. For the attack scenario, we launched a TCP-SYN flood from Hosts 1 and 2, while the intended target/victim machine was Host 8. The goal of the proposed framework is to be able to detect the ongoing attack and ultimately, localize which of the hosts are malicious by finding the switch closest to the attackers. The attack deployment was conducted using the scapy library while ML analysis, threat detection, and localization were conducted using TensorFlow, sklearn, and python.

B. Results and Analysis

For our experiments, we utilized the following initial values: $N = 50, A = 50, U = 100, \Phi = 0.3, \Theta = 3$. Our generated datasets were split using an 80%-20% ratio between training and testing data. For performance metrics, we are using Accuracy (A), Precision (P), Recall (R), and F-Measure (F) scores. The ML classifier chosen for our experiments was a Random Forest (RF) classifier. This algorithm was chosen due to its relevance and wide usage when studying SDN anomaly detection in literature.

First, we look at the performance that is being achieved by various ML classifiers on our collected differential port statistics dataset. The classifiers tested were RF, Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM) due to their relevance and wide usage when studying SDN anomaly detection in literature. The model with the highest accuracy can be correlated with better performance during TCP-SYN flood attack detection. Table III illustrates the performance of the algorithms when running the generated datasets. We notice that the RF gives us the best performance followed by the SVM, while the MLP provides us with mediocre performance for our dataset.

TABLE III: Performance of RF, MLP, and SVM to the generated dataset

Classifier	Accuracy
Random Forest	99.342%
Multi Layer Perceptron	51.409%
Support Vector Machine	97.852%

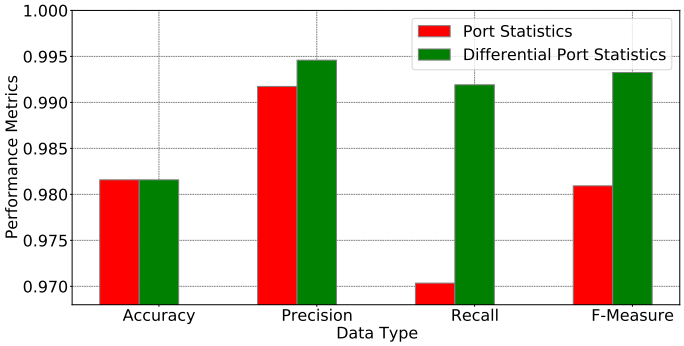

Fig. 4: ML performance on port statistics and differential port statistics

Next, we analyze the impact of differential port statistics on the performance of our RF anomaly detector. For this, we observe the performance achieved by our ML algorithm when facing normal port statistics versus differential port statistics. Figure 4 shows us the performance achieved on port statistics and differential port statistics. The ML classifier achieves good performance on both data types, but the algorithm performs better on differential port statistics as it attains better P, R, and F scores. This can be attributed to differential port statistics being collected and reported as magnitude differences in a time interval. This prevents the aggregation of metrics over time, which may be occurring in normal port statistics. This also allows more fine-grained analysis of network flows from the port level and provides better performance metrics. Superior performance is preferred as this anomaly detector will be placed to analyze live/real-time traffic which has no labels.

For a comprehensive analysis of the degree of influence our captured network metrics exhibit over the classification performance of our RF model, we also analyze the explainability of the model. This is conducted by analyzing the predictions of the model, on a random single testing sample, using the Local Interpretable Model-agnostic Explanations (LIME) framework [18] and the results are provided in Figure 5. This helped provide explainability and interpretability to our TCP-SYN detection framework. We observe that, for the sample, the model predicted it as undergoing a *TCP-SYN flood* attack. The main dataset features that were influential in the prediction were differential port statistics like *Received Packets*, *Sent Packets*, *Received Bytes*, and *Sent Bytes*. From this, we can see the impact these metrics have on the detection of TCP-SYN attacks detection. This also reinforces customary knowledge that network metrics revolving around byte/packet transmission, along with network throughput, are key in detecting if a DDoS attack is actively exhausting network resources.

Subsequently, we observe the performance of our proposed

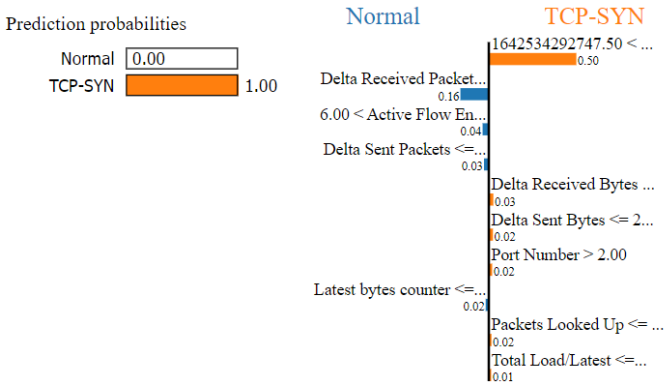

Fig. 5: Using LIME to find most influential features

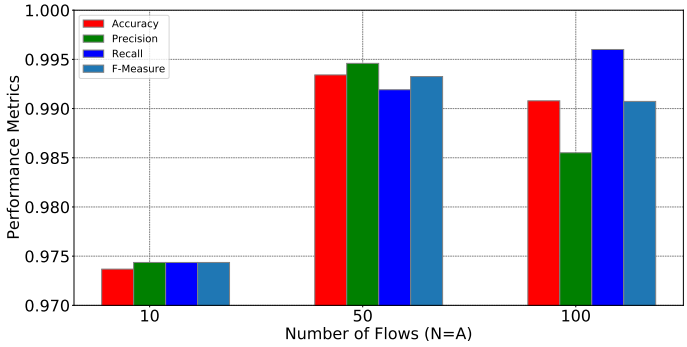

Fig. 6: ML performance on varied number of flows for N and A

method by varying the number of normal and attack flows used to generate the dataset. For this, we obtain performance metrics while we are varying the values of $N = A = \{10, 50, 100\}$. The results of this experiment are illustrated in Figure 6. We observe that $N = A = 10$ provided decent A, P, R, and F scores, but those scores rose when $N = A = 50$. The magnitudes continue to be steady when $N = A = 100$. At $N = A = 10$, the model was not optimally learned. It achieved optimal learning at $N = A = 50$ and continued to hold the same performance till $N = A = 100$.

The above results were conducted using the testing dataset from our generated TCP-SYN flood detection dataset. However, the primary aim of this research is to propose a framework that can conduct anomaly detection on real-time data. The generated ML algorithm can be placed in the SDN controller, and it can observe live traffic to provide rapid inferences to protect the SDN infrastructure from TCP-SYN flood attacks. The remaining analysis is conducted using real traffic from the SDN infrastructure while it is undergoing the attack.

Next, we observe the performance of the proposed approach under varying values of $\Phi = \{0, 0.05, ...0.35\}$. The goal here is the observe the performance metrics when the value of Φ is diversified. The analysis is illustrated in Figure 7. We can see that P and F are low at the lower values of Φ. As the value of Φ increases, the values of P and F progressively increase till $\Phi = 0.3$. The value of R remains constant throughout. This

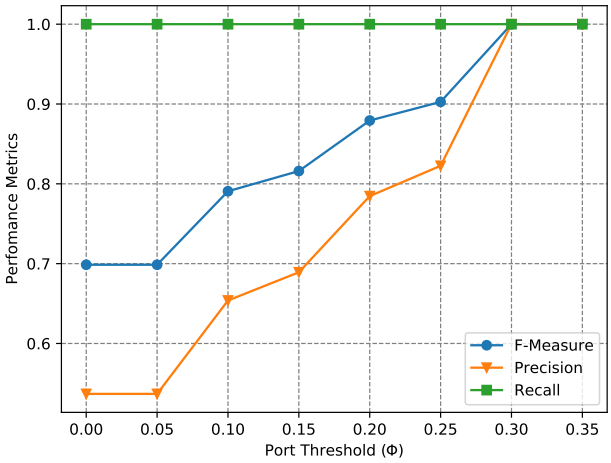

Fig. 7: ML performance on varied values for Φ

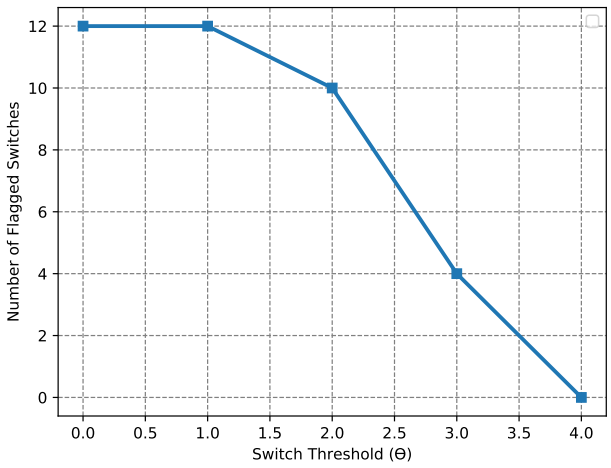

Fig. 8: Detecting Number of Flagged Switches by varying Θ

suggests that lower values of Φ restrict our ability to optimally detect TCP-SYN floods, as many normal flows get flagged as under attack, leading to high false positivity rates. This can limit the ability of the framework to optimally detect ongoing TCP-SYN floods. Higher values of Φ provide more balance to the framework and lead to better performance.

Similarly, we observe the performance of the proposed approach under varying values of $\Theta = \{0, 0.5, ...4\}$. The goal here is to observe the number of flagged switches when the value of Θ is diversified and to identify the value of Θ that minimizes the number of flagged switches. The analysis is illustrated in Figure 8. We observe that lower values of Θ yield a high number of flagged switches. That maximizes the number of switches through which a flow under attack passes and does not help identify and localize the malicious hosts, which is an optimization problem. As the value of Θ increases, it provides more control to the anomaly detector to appropriately detect an attack and localize the malicious hosts as it increases the threshold for how a switch becomes flagged. $\Theta = 3$ provides the best performance as it identifies the minimal number of flagged switches before that value becomes null, which is 4. This means that the malicious hosts must be connected to these 4 switches.

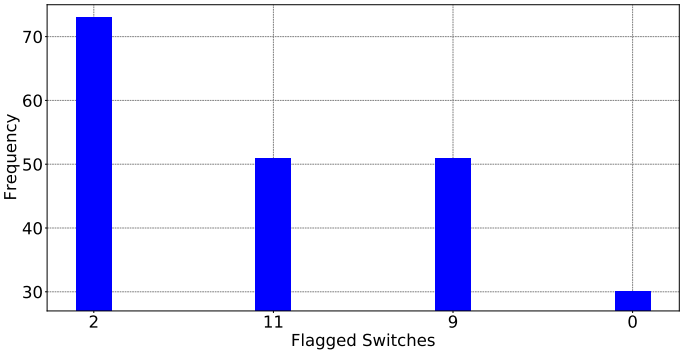

Fig. 9: Checking the most frequently flagged switches over U number of real-time flows

Lastly, we perform the final step of threat localization in which we compute the frequency with which certain switches are flagged over the total number of real-time flows being observed. This can be observed in Figure 9. Here, we observe that over U number of real-time flows, Switches 2, 11, 9, and 0 are flagged the most times with frequencies of 72, 51, 51, and 30, respectively. Switch 2 is flagged the highest number of times, which indicates that the malicious hosts must be closely connected to this switch. This is confirmed as we can observe, from the topology diagram in Figure 3, that the malicious hosts 1 and 2 are both directly connected to Switch 2. Using the proposed methodology, we can identify that the SDN infrastructure was undergoing a TCP-SYN flood attack using differential port statistics. On top of identification, we are also able to localize the malicious hosts 1 and 2 using our Threat Localization module.

V. Conclusion and Future Work

In this paper, we propose a TCP-SYN flood detection framework for SDNs. The framework encompasses the usage of an SDN simulation framework as the base architecture so that each organization can cater their approach to their infrastructure. Instead of traditional network flow statistics, we focus on collecting differential port statistics to conduct our analysis. The collection of these metrics to perform TCP-SYN flood detection has not been investigated previously. Also, usage of metrics can allow us to reduce dataset cardinality, training, and inference times. We use an ML algorithm to perform our detection on both testing and real-time data. Lastly, we also propose our Threat Detection and Localization technique that can pinpoint which of the network hosts are malicious. Experimentation and results show that using differential port statistics over normal port statistics provides better P, R, F, and A scores. The analysis also illustrates that a $\Phi = 0.3$ and $\Theta = 3$ provide the best performance for detecting and localizing malicious hosts. We aim to expand this framework to other attack scenarios besides DDoS attacks for future work. The focus will be placed on threats that are not easily perceivable. Potential options include other SDN threats like traffic diversion, traffic sniffing, IP spoofing, SQL injections, and flow table overflows. We also will add explainability to study those attack scenarios under our framework to provide more insights on which differential port statistics are most influential in detecting the various attacks that we propose to experiment on.

References

[1] T. Das, R. M. Shukla, and S. Sengupta, "The devil is in the details: Confident & explainable anomaly detector for software-defined networks," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2021, pp. 1–5.

[2] R. M. Shukla, S. Sengupta, and M. Chatterjee, "Software-defined network and cloud-edge collaboration for smart and connected vehicles," in *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, 2018, pp. 1–6.

[3] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[4] B. Özçam, H. H. Kilinc, and A. H. Zaim, "Detecting tcp flood ddos attack by anomaly detection based on machine learning algorithms," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2021, pp. 512–516.

[5] M. H. Khairi, S. H. Ariffin, N. A. Latiff, A. Abdullah, and M. Hassan, "A review of anomaly detection techniques and distributed denial of service (ddos) on software defined network (sdn)," *Engineering, Technology & Applied Science Research*, vol. 8, no. 2, pp. 2724–2730, 2018.

[6] D. K. Sharma, T. Dhankhar, G. Agrawal, S. K. Singh, D. Gupta, J. Nebhen, and I. Razzak, "Anomaly detection framework to prevent ddos attack in fog empowered iot networks," *Ad Hoc Networks*, vol. 121, p. 102603, 2021.

[7] L. Van Efferen and A. M. Ali-Eldin, "A multi-layer perceptron approach for flow-based anomaly detection," in *2017 international symposium on networks, computers and communications (ISNCC)*. IEEE, 2017, pp. 1–6.

[8] J. Mirkovic, G. Prier, and P. Reiher, "Attacking ddos at the source," in *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE, 2002, pp. 312–321.

[9] T. M. Gil and M. Poletto, "Multops: A data-structure for bandwidth attack detection." in *USENIX security symposium*, 2001, pp. 23–38.

[10] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[11] R. Chen, J.-M. Park, and R. Marchany, "Nisp1-05: Rim: Router interface marking for ip traceback," in *IEEE Globecom 2006*. IEEE, 2006, pp. 1–5.

[12] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *IEEE/ACM Transactions on networking*, vol. 15, no. 1, pp. 40–53, 2007.

[13] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," *ACM SIGCOMM computer communication review*, vol. 31, no. 4, pp. 15–26, 2001.

[14] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," *IEEE network*, vol. 12, no. 5, pp. 50–60, 1998.

[15] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1545–1559, 2018.

[16] N. Ravi, S. M. Shalinie, C. Lal, and M. Conti, "Aegis: Detection and mitigation of tcp syn flood on sdn controller," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 745–759, 2020.

[17] N. N. Tuan, P. H. Hung, N. D. Nghia, N. Van Tho, T. V. Phan, and N. H. Thanh, "A robust tcp-syn flood mitigation scheme using machine learning based on sdn," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2019, pp. 363–368.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.