CyberPulse++: A Machine Learning Based Security Framework for Detecting Link Flooding Attacks in Software Defined Networks

RAIHAN UR RASOOL[1], KHANDAKAR AHMED[1], ZAHID ANWAR[2,3], HUA WANG[1], USMAN ASHRAF[4] AND WAJID RAFIQUE[5]

[1]Victoria University, Melbourne, Australia
[2]National University of Sciences and Technology, Islamabad, Pakistan
[3]North Dakota State University, USA
[4]School of Business, Torrens University, Sydney, Australia
[5]Montreal Blockchain Laboratory, Department of Computer Science and Operational Research,University of Montreal, Canada

Corresponding author: Raihan ur Rasool (e-mail: raihan.rasool@live.vu.edu.au)

ABSTRACT A new class of link flooding attacks (LFA) can cut off internet connections of target links by employing legitimate flows to congest these without being detected. LFA is especially powerful in disrupting traffic in software-defined networks if the control channel is targeted. Most of the existing solutions work by conducting a deep packet-level inspection of the physical network links. Therefore these techniques incur a significant performance overhead, are reactive, and result in damage to the network before a delayed defense is mounted. Machine Learning of captured network statistics is emerging as a promising, lightweight, and proactive solution to defend against LFA. In this paper, we propose a machine learning-based security framework, CyberPulse++, that utilizes a pre-trained machine-learning repository to test captured network statistics in real-time to detect abnormal path performance on network links. It effectively tackles several challenges faced by network security solutions such as the practicality of large-scale network-level monitoring and collection of network status information. The framework can use a wide variety of algorithms for training the machine-learning repository and allows the analyst a birds-eye view by generating interactive graphs to investigate an attack in its ramp-up stage. An extensive evaluation demonstrates that the framework offers limited bandwidth and computational overhead in proactively detecting and defending against LFA in real-time.

INDEX TERMS Control Channel Attacks, Link Flooding Attacks, Machine Learning, Network Security, Traffic Classification, SDN.

I. INTRODUCTION

A fundamental characteristic of Software Defined Networking (SDN) is that it provides separation between the control plane and data plane. This separation offers better management and flexible programmability of the network elements. OpenFlow (OF)- a popular network architecture of SDN develops network services as applications where the application plane interacts with the controller using a northbound API. While SDN allows a considerable advantage of centralized network management over traditional networks, this same feature can be exploited to cause various types of attacks such as Denial of Service (DoS), flow table memory exploitation attacks, open programmability related issues, and data plane attacks. These lead to service disruptions thereby increasing management costs and incurring delays for the stakeholders. Providing seamless network services have become a critical challenge in the modern-day. Hence, mitigating threats to network performance and operation has become vital to meet network availability requirements [1].

A. LINK FLOODING ATTACKS

The motivation for this research comes from the analysis of LFAs which have emerged as one of the most devastating threats to the internet backbone in the modern era. In contrast to traditional DoS attacks targeting individual servers, LFAs target critical links that connect to cloud servers [2]. LFAs are lethal in nature because they clog core links by paralyzing and sometimes disconnecting the target network connected to the link. LFA is a growing threat to the network infrastructure and has attracted a lot of attention from the networking community[3], [4], [5], [6]. Mitigating LFA is a challenging task because the attack traffic remains indistinguishable from the legitimate flows and is almost unobservable if the attacker employs a low bit rate. Therefore, typical flow filtering techniques to screen-out malicious traffic become inoperative. Multiple bots are also used to send rate-limiting traffic to the decoy servers, causing a persistent attack. Traditional intrusion detection (IDS) and intrusion prevention (IPS) countermeasures may not work in this situation because the attack traffic never reaches the target server. Hence, legitimate traffic to the target server is interrupted.

In Fig. 1, the phenomena of LFA is presented, where an adversary is sending traffic to the bots which in turn send traffic to the links surrounding the target server. Ultimately, customers are unable to connect to it. At the start, the probers of the adversary group utilize traceroute packets to create a link map of the network. Then the adversary calculates the attack-cost strategy and ascertains the number of bots that can occupy the links. Finally, the adversary sends TCP like traffic to the decoy servers to occupy the links. This sequence of operations results in the full utilization of the link capacity thereby disrupting the legitimate traffic.

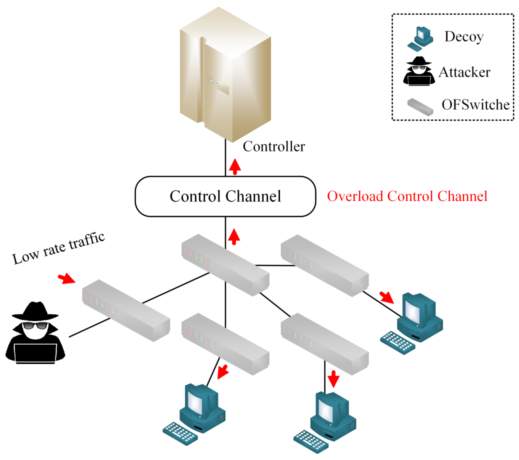


FIGURE 1: LFA Scenario in SDN.

There are several variations of LFAs differing in behavior that are in existence today. Noteworthy among these are the Crossfire attack [2], Coremelt [7], and Spamhaus attacks [8] detailed in section 2. This makes it challenging for traditional firewall and IDS/IPS solutions to detect and mitigate LFAs effectively.

B. LFAS TARGETING SOFTWARE DEFINED NETWORKS

It is emphasized that SDN is a popular network paradigm that has attracted attention from academia and industry and is a building block for most datacenter networks today. Hence, securing SDN will ensure the security of many global network infrastructures. Despite its importance, most of the existing solutions [6], [9], [3], [10] have been unable to adequately resolve the problem of LFA on traditional networks and SDN. Some works have used SDN testbeds only for proof of their proposed techniques, and there is a clear lack of sufficient work to address SDN specific issues in the context of LFAs. We argue that providing defense against LFA on the control channel is one of the prime concerns to secure today's Internet. There is a strong need for a holistic approach to this problem. In our previous work we proposed using a deep content inspection system such as [11] as a basis of identifying LFA traffic. We then conducted a preliminary study on the effectiveness of employing a deep learning based classifier to classify traffic as either *legitimate* or *flooding* [12] for the LFA problem in SDN. The said approach dubbed as CyberPulse accurately detected LFA with a low false positive rate when trained using a publicly available dataset from the UCI machine learning repository.

C. CONTRIBUTIONS

In this paper a novel framework, CyberPulse++, is proposed that extends the ideas from our previous work CyberPulse [12] to cover the gaps in existing solutions to LFA in SDN. CyberPulse++ draws on several components to secure SDN, including, real-time network surveillance, ML-based traffic analysis, and an efficient and accurate LFA defense with low overhead. Where CyberPulse was based on deep learning alone, CyberPulse++ employs an algorithm-agnostic approach which leverages multiple ML models for traffic classification and LFA mitigation. Recent research [13], [14] formulates the robustness of scale-free networks to cascading failures as a multi objective optimization problem. This is based on the fact that networks which are robust against one type of attack may not be robust against another type of attacks. In practical situations, different types of attacks may happen simultaneously. CyberPulse++ furthers these research efforts in the SDN domain where an attacker may attempt different variations of LFA on the victim network. In this regards multiple algorithmic selection would increase the attack coverage. Our results demonstrate that certain models are more suited that others depending on the network configuration at hand and CyberPulse++ allows for this flexibility in the selection.

For the ML classifier component to work accurately, we carefully select and train algorithms with flood attack data sets. Ten different ML algorithms were selected and observed during the training phase. Adaptive Boosting (Ad boost) and Stochastic Gradient Descent (SGD) provided improved accuracy. CyberPulse++ successfully detects LFA and drops malicious flows, by acquiring the flow rules installed in the switches. It also provides a dashboard of multiple real-time network status graphs including throughput, saturated bandwidth, end-to-end delay and flooding rate thereby performing the surveillance in a very interactive manner.

To the best of our knowledge, this is the first work that proposes a practical defense for LFA that has high detection accuracy and works at line speeds. The evaluation of the CyberPulse++ framework quantifies this. This paper makes the following contributions:

- We discuss the design and implementation of CyberPulse++, an interactive real-time security solution for control channel LFAs in SDN which employs ML techniques for training and real-time classification of network traffic.
- We highlight the algorithm-independent capability of CyberPulse++ in that it works independent from the use of training algorithms and datasets and is flexible enough to accept models trained in any programming language to perform the classification of the network traffic.
- We demonstrate using a fully functional prototype of CyberPulse++, that it is indeed possible to dynamically change the network configurations including topology, traffic, and the underlying machine learning algorithms.
- Finally, we provide a comprehensive performance evaluation of CyberPulse++ encompassing ten different ML algorithms across a spectrum of varying network and traffic settings. CyberPulse++ provides substantial mitigation against LFAs.
- Our evaluation results demonstrate the selection of specific classification method, is problem based and some are more suitable than others thereby providing better results.

The rest of the paper is organized as follows. Section II describes the current literature in the context of SDN and LFA, and Section III outlines the overview and architecture of CyberPulse++. Section IV discusses the ML component of the framework, furthermore, Section V provides the experimental setup. Section VI discusses the results and Section VII concludes the paper and provides some future directions.

II. RELATED WORK

Being an emerging threat targeting the Internet infrastructure, LFA has gained a lot of attention in the literature during the past several years [5], [6], [15], [16]. LFA was initially proposed by Kang et al. [2], [17] in what the authors term as the Crossfire attack which can disconnect target links without being detected. The Crossfire LFA sends low-rate traffic to the target link by employing several bots to congest legitimate traffic to the target server. As the target server does not receive any direct flood, it causes traditional flow filtering and IPS/IDS techniques to fail in detecting this attack.

We broadly categorize LFA defense techniques into three categories, 1) links inspection-based, 2) traffic engineering-based, and 3) SDN-based approaches.

A. LINKS INSPECTION-BASED

The key idea behind link inspection-based techniques [6], [4], [18] is to continuously measure the network statistics on a link to detect any malicious activity. The link obfuscation technique proposed by Wang et al. [10] presents fake links to the adversaries which makes it harder to create a true link map of the network to attack. Linkscope [6] inspects network links on a hop-by-hop and end-to-end basis and reports any malicious activity observed on the link. SPIFFY [4] adapts to the increase in network bandwidth temporarily and measures flow statistics before and after the expansion. Legitimate flows tend to adapt to bandwidth expansion. Alternatively, malicious flows will result in consuming all their bandwidth and hence will be easily detected. SDN HoneyNet [9] measures graph statistics and identifies the lowest betweenness centrality links that can become a bottleneck. It then deploys an SDN HoneyNet to fake the adversary.

B. TRAFFIC ENGINEERING-BASED

Traffic engineering based techniques rely on multiple attackers and defender interactions in the network to expose LFA sources. Takayuki [9] employs an increase in traceroute packets volume based measurement to detect LFA. Authors in [3], [15] presented multiple attacker-defender interactions to balance the network traffic and expose the attacker's identity. A virtual network-based dynamic resource allocation strategy has been proposed by [19], however, the limitation with this technique is that it relies on the security of the virtual layer, and becomes vulnerable in case the layer is exposed. Xiaobao et al. [16] proposed a cost incentive approach for the source and destination Autonomous System (AS) domain to cooperate and alleviate LFA.

Although traffic engineering techniques provide a viable solution for the defense against LFAs however, they incur extra overhead while detecting and defending this attack. These techniques rely on traffic manipulation, which increases traffic on certain links in the network, which thereby becomes a bottleneck during attack detection.

C. SDN-BASED APPROACHES

SDN-based techniques rely on controller-based measurements to inspect the flows in the network, and then utilize SDN principles to increase network connectivity or flow rules installation on the switches to drop the malicious flows. Peng et al. [20] proposed a flow table inspection technique that examines the flow tables when the resource utilization ratio of a flow is not normal and employs a bloom filtering technique to detect adversaries. An SDN-based traffic maneuvering technique was proposed by Abdullah et al. [21] which obfuscates the links making it difficult for the adversary to create a true link map, hence avoiding the attack. LFADefender [5] uses SDN based measurement to detect and defend against LFA. Woodpecker [22] uses incremental SDN deployment to increase network connectivity in the case of LFA. Both LFADefender and Woodpecker only work in providing defense for traditional networks.

SDN-based approaches provide a centralized control on detecting and mitigating LFAs. However, these techniques merely provide a testbed to mitigate LFAs in traditional networks. To address this limitation, there is a need to provide security against LFAs on the control channel of SDN.

Recently, some research works focused on mitigating DoS on the control channel have been proposed. FloodDefender [23] leverages the controller's capability to perform network measurements and flow rule management characteristics to defend against LFA. It also employs a table miss-oriented technique and packet filtering mechanism to mitigate DoS attacks. Similarly, FloodGuard [24] distributes load to the neighboring switches to protect the link from being clogged. However, flow filtering and table miss-based techniques become invalid in the case of LFA which uses low-rate legitimate traffic to flood the network. AvantGuard [25] uses TCP handshake to provide defense against table miss attacks however it only works for TCP traffic. SLICOTS [26] provides defense against TCP flooding attacks. It forwards the TCP handshake packets to the controller to authenticate however, it increases the load on the controller during the authentication process. FloodShield [27] and deep learning-based defense [28] are recent techniques proposed to cater to DoS attacks. BWManager [29] provides defense against the controller and switch table overloading attacks by isolating the TCP and UDP attack traffic. RADAR [30] is a recent solution aimed at mitigating crossfire attacks on the data plane using commercial-off-the-shelf switches. RADAR uses the controller and data plane cooperation to provide defense against crossfire attacks, however, RADAR uses multiple controllers and data plane interactions which can increase network.

These techniques provide security against DoS on SDN, however, low-rate traffic makes these techniques ineffective in detecting LFAs. Hence, the abovementioned approaches suffer from a lack of detection accuracy while applied in the LFA context.

ML-based solutions can learn from historical flooding behaviors and detect the characteristics of flooding flows. As ML-based techniques have strong learning ability and defense efficiency, these techniques can be applied for LFA detection and defense. Aapo et al. [31] presented an ML-based technique that combines normal traffic learning, blacklists integration, and elastic capacity invocation to provide effective load control, filtering, and service elasticity during an attack. The external blacklists are obtained from existing IDS. Fairuz et al. [32] proposed an anomaly-based approach to evaluate malware detection using ML classifiers. In [33], the authors propose a DoS attack detection system on the source side of the cloud. They consider multiple attack scenarios for evaluation including SSH, brute-force attacks, ICMP flooding attacks, DNS reflection, and TCP SYN attacks. A model is trained using the training dataset and an ML module is utilized to update the pre-training module [34]. They used multiple classifiers for evaluation including Logistic Regression, Support Vector Machines (SVM) linear kernel, SVM, RBF Kernel, SVM Poly Kernel, Decision Tree, Naïve Bayes, Random Forest, K-means and Gaussian EM. Flow-Based IDS for SDN was proposed in [35], which periodically gathers statistical information about flows from SDN OF switches, and performs the analysis and detects the intruders by utilizing the collected traffic features. In [28], the authors proposed a deep learning-based approach for DoS defense.

The challenge with these ML-based defense techniques is that they are restricted to the use of specific algorithms and that they are reactive. These techniques also suffer from low detection accuracy, adaptability problems, leakage report, and pose more overhead to the network. On the other hand, CyberPulse++ is not dependent on the use of a specific algorithm but provides a generic framework which can incorporate any algorithm for classification purpose. Moreover, it provides real-time defense and proactive LFA mitigation.

More recently researchers are focusing on the detection and mitigation of LFA in the network of the Internet of Things (IoT). Chen et. al [36] use a model based on Convolutional Neural Networks to learn the changing patterns in time and space of LFA attacks targeting sensor devices in IoT networks. Long short-term memory (LSTM) is used for storing samples allowing constant review of LFA patterns and removal of obsolete patterns to improve decision accuracy. The research is studied in a simulated environment. The authors in [37] frame the LFA problem in IoT networks as a Bayesian game-theoretic model. The volume of attack traffic is the attacking behavior and rerouting and scrubbing are defending actions. An SDN-based global network manager is assumed and used to solve the game using a divide and conquer method. The authors also provide local optimal strategies for the attacker and defender on a link-by-link basis. The practicality of the approach on a real network has not been explored.

In summary, existing LFA mitigation techniques employ link inspection, traffic engineering, or SDN-based approaches. There are very few available research works that utilize ML to detect and mitigate LFAs, and most of them only perform offline detection. Our recent study, CyberPulse [12] investigated the effectiveness of LFA defense using ML techniques and provides efficient results. However, CyberPulse is limited in that it does not provide real-time defense against LFAs. As network surveillance techniques become more critical, there is also a strong need to devise a solution that provides real-time network status information to the network administrators. Therefore, we propose a security framework, CyberPulse++, which employs an ML-based LFA detection and mitigation mechanism in SDN, that works at line speed and also provides network status information to the administrators. The CyberPulse++ prototype solution uses ten separately trained classification algorithm trained on Burst Header Packet (BHP) dataset [38] to accurately detect LFA.

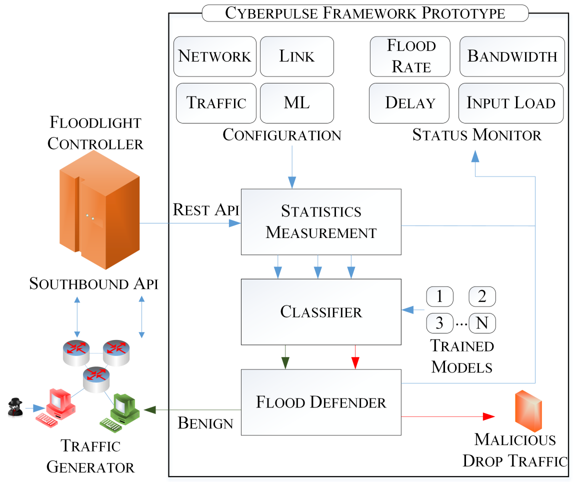


FIGURE 2: A high-level overview of CyberPulse modules.

III. OVERVIEW AND ARCHITECTURE

In this section, an overview of the CyberPulse++ framework including its architecture, components, and dependencies is provided and illustrated in Fig. 2. CyberPulse++ is developed as an application deployed at the application layer of the Floodlight open-source controller to defend against control channel LFA. As seen in the figure, the *controller* is connected with data plane devices using southbound API. A *malicious host* is shown for purposes of illustration which is generating flood traffic on the network. A *statistics measurement module* is responsible for extracting network statistics from the Floodlight controller using the REST API. The *classifier* contains pre-trained classification algorithms that acquire traffic features from the statistics measurement module. It then classifies the network traffic based on a defined confidence level and transfers the results to the *flood defender*. Finally, the flood defender module drops malicious flows by updating appropriate rules in the OF switches. The *network status monitor* plots surveillance graphs by acquiring information from the statistical measurement and flood defender modules.

The architecture of CyberPulse++ consists of five main modules illustrated in Fig. 3. All the modules with the exception of the *flood defender* rely on sub-components to accomplish their operation. For example, the *ML module* consists of the *dataset*, *ML training*, and *ML classification* components.

LFA detection and elimination in CyberPulse++ is based on a periodic sampling of network traffic statistics, identification of malicious flows by utilizing ML techniques and finally, mitigation. CyberPulse++ utilizes supervised ML models for traffic classification which can be trained using any classification algorithm and dataset. To test this capability, we utilize 10 algorithms for classification in our experiments. CyberPulse++ modules are discussed in detail in the following sub-sections.

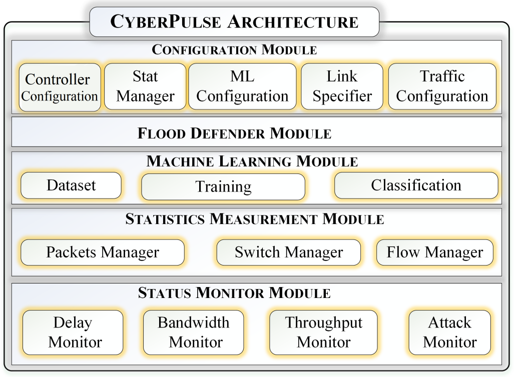


FIGURE 3: CyberPulse++ architecture.

A. CONFIGURATION MODULE

This module provides the ability to configure network parameters for CyberPulse++ via a JSON configuration file (illustrated in Fig. 6) by incorporating multiple sub-functions. This file provides flexibility to control the following network specifications:

1) Controller Configurations
2) Statistics
3) ML Specifications
4) Link Properties
5) Traffic Parameters

The *controller configuration* sub-module handles controller parameters, its IP address, and the REST and OF port numbers. The *statistics manager* defines the program duration, statistics collection interval, and flooding threshold of flows in the network. *ML configuration* contains the name of the algorithm to use, the confidence level for classification, and action to be taken while classification is complete. Similarly, the *link specifier* contains the link identifiers and the port number on switches where they are connected. Finally, the network topology and traffic parameters can be provided using the *traffic configuration* sub-module. This module manages the network parameters, such as the link capacity, number of switches, hosts per switch, traffic duration and traffic generation details.

B. STATISTICS MEASUREMENT MODULE

In SDN, when a flow arrives at any OF switch, its flow entries are matched with the switch-flow table and the flow is forwarded to the destination according to the flow entry in the switch. If a specific flow entry is not found, the flow is forwarded to the controller using a PACKET_IN message which results in a flow table entry being created in the switch for forwarding. A FLOW_REMOVED message is sent to the controller to remove the flow from the flow table. In addition to this, every second a thread running on the controller requests the statistics information of every flow from the switch. A FLOW_STATS reply message is sent to the controller that contains the statistics information of every flow entry in the table.

TABLE 1: Extracted features and definitions.

Extracted Feature	Definition	Extracted Feature	Definition
Node	Node generating flow	Used Bandwidth	This is what each node could reserve from the reserved bandwidth
Utilized Bandwidth	Bandwidth utilized by the node, normalized bandwidth used	Lost Bandwidth	The amount of lost bandwidth by each node
Packet Drop Rate	Packets dropped per unit time	Packet Size	Size of packets on the network
Full Bandwidth	Bandwidth of the network	Packets Received	Received packets in a given time interval
Percentage of packet lost Rate	Percentage of packets lost	Packets Lost	Packets lost in a given time interval
Percentage of lost byte rate	Percentage of lost byte rate	Transmitted Byte	Total bytes transmitted per unit time
Packets received rate	Packets received per unit time	Flooding Status	Percentage of flood per node based on Packet Drop Rate
Class	Class to which the instance belong to		

The *Statistics collection* module utilizes the above-mentioned capability of SDN to query data plane switches. The module defines all the required statistics to collect during its operation. CyberPulse++ uses the REST API of the Floodlight controller to collect 14 different statistics as shown in Table 1. *Statistics measurement* then utilizes *switch*, *flow*, and *packet* manager to collect these statistics.

The *switch manager* assigns a Data Path ID (DPID) to every switch and enlists the connected switches and their port numbers. It then identifies the critical switch and port numbers in the network based on the flooding rate on all the ports. The *Flow manager* handles all the flows in the network and uses the '/wm/core/switch/all/flow/JSON' API to manage these flows. It assigns a class label to the flows based on the predefined threshold of ByteCount. The *Packet manager* extracts the packet statistics in the network using the '/wm/core/switch/all/port/JSON' API. Next, it determines the number of transmitted and received packets on all the ports and calculates the packet loss rate. Subsequently, it calculates all the statistics required for traffic classification as described in Table 1. Once the statistics are collected, the *ML module* is invoked to classify malicious traffic. Since the statistics collection is an existing capability of the SDN architecture, it will not pose any extra performance overhead. Additionally, the statistics collection, ML classification and the monitoring are designed to run in their own separate threads without blocking the main controller functionality. This allows CyberPulse++ to classify incoming traffic continuously. Therefore the controller main functions of flow control and the classification and detection functions of CyberPulse++ can proceed concurrently without degrading the SDN controller's performance.

C. STATUS MONITOR MODULE

This module extracts the current traffic data from the *statistics measurement module* and plots the current status of the traffic in the network. This module contributes to the contemporary traditional defense mechanisms by offering detailed monitoring of network activity in real-time. The *network status monitor* draws the following plots:

1) **Flooding rate:** This is the bandwidth consumption status of a link under normal circumstances as well as when the network is under attack. This plot shows the data rate on important links connected to the switches, controller, servers, and other important devices. Typically switches themselves are not the terminal points but lead to hosts and servers so, we consider switches leading to important devices as critical switches as compared to those switches that connect to end stations.
2) **Packet drop:** When the flooding rate increases, the packet drop rate also escalates. This plot displays the packets dropped rate on the network at a specific time.
3) **Throughput:** This plot displays the aggregate throughput of the network as well as that of the individual links connected to switches or servers.
4) **Traffic delay:** When the flooding starts, the legitimate flows will face delay and packet drop. This graph plots the delay incurred on transferring files among hosts during LFA.
5) **Bandwidth saturation:** This plot shows the saturation of the bandwidth of a link with an increase in the number of attacks.
6) **Attack detection time with respect to the number of attackers:** This plot shows the efficiency of CyberPulse++ in detecting an LFA and analyzes the feasibility of deploying CyberPulse++ in the presence of such an attack.

D. MACHINE LEARNING MODULE

This module trains an ML classifier using the BHP training dataset on 10 classification algorithms [38]. It should be noted though that CyberPulse++ is not dependent on these particular algorithms only. To prove the feasibility of employing multiple algorithms, we perform the experiment using these. We provide the classification algorithms in the form of .pkl files. Since ML training and evaluation constitutes an extensive part of our research, section IV is dedicated to its explanation.

E. FLOOD DEFENDER MODULE

The *Flood defender module* eliminates malicious flows by dropping them. This is implemented as a function in CyberPulse++ which takes statistics of an active flow (e.g. source IP, destination IP, switch ID, flow ID) and compares the confidence level of the current flow with the threshold value set in the configuration file. If the flow has a confidence interval greater than the defined value in the threshold, then the flow is dropped with the help of a drop flow rule in the data plane switches.

TABLE 2: Overview of training dataset.

Node	Utilized Bandwidth	Pack Drop Rate	Bandwidth	Packet Lost Rate %	Lost Byte Rate %	Packet Received Rate	Used Bandwidth
3	0.82	0.19	1000	19.03	19.03	0.81	822.04
9	0.27	0.73	100	72.89	72.91	0.27	27.55
3	0.92	0.09	900	9.03	9.03	0.91	831.34
3	0.37	0.64	100	63.74	63.77	0.36	36.88

IV. ML MODEL TRAINING AND TESTING

The algorithms and the quality of training of an ML model is the backbone of this research, hence this section presents its extended detail. CyberPulse++ has been designed to accommodate any set of algorithms and trained models, thereby providing flexible operation and quality of traffic classification. We have selected a set of ten algorithms for network traffic classification because they provide better classification results for SDN traffic analysis. Some recent research [39] also categorizes these algorithms as suitable for SDN traffic classification. CyberPulse++ is efficient when configured in a multi-threaded application paradigm in such a way that the classification of the network traffic is performed concurrently in each thread using any of the available algorithms. The main motivation for selecting the ten algorithms is that it provides a holistic solution for network traffic analysis where the user is not limited to using the capabilities of a fixed algorithm for network defense. He may simply export his trained model of choice as a .pkl file and employ it for traffic classification.

The traffic classification component of our presented solution benefits from ML models trained using the Jupyter note- book [40]. Jupyter Notebook is an open-source web application popular among data scientists for creating and integrating live code and visualizations. We select Jupyter because it supports a variety of programming languages including Python, R, and MATLAB, is organized as a JSON file which is easy to annotate and supports tasks required for our research such as statistical modeling, machine learning and deep learning. CyberPulse++ employs BHP flooding attack dataset for training [38]. We used Python 3.6 for CyberPulse++ development because it provides support for the required classification frameworks. Table 2 provides an overview of 8 features of one of the datasets used in this research which contains, 1075 instances and 22 features. The features that were important for detection were mostly related to bandwidth and packet rate although others were considered such delay and bytes transmitted and received. Bandwidth related features include the *utilized*, *reserved*, *used* and *lost* bandwidth. Packet rate features include the packet *drop*, *lost* and *received* rate. The dataset is trained using the following algorithms.

1) Adaptive Boosting
2) Bagging Classifier
3) Decision Tree
4) K-Nearest Neighbors
5) Logistic Regression
6) Multi-Layer Perception
7) Naïve Bayes
8) Random Forest
9) Stochastic Gradient Descent
10) Support Vector Classification

Extensive data pre-processing steps are performed before the actual training using the above algorithms. These steps are explained in the following subsections.

A. CLASS TRANSFORMATION

The actual class distribution in the BHP training dataset corresponds to Block, NB-No Block, No Block, and NB-Wait which relates to different flooding stages based on flow statistics. However, our study aims to classify traffic into flooding and legitimate flows. Therefore, we customize the interpretation of these classes according to our research problem. The dataset was transformed into flooding and legitimate classes, where Block corresponds to flooding and the rest of the classes are assigned to legitimate. The final classes after transformation consist of 88% legitimate and 12% flooding instances.

B. DATA PRE-PROCESSING

The dataset was loaded in the Jupyter notebook and an expert analysis method was applied to summarize the features. The method provides an overview of the dataset to understand the statistics of different features. The Data description in Table 3 shows the statistics of all the attributes including, count, mean, standard deviation, minimum, maximum, and 25%, 50% and 75% percentiles.

The analysis shows that the packet loss rate feature contains some missing values so, we replace them with median values of the same feature to ensure justifiable distribution of all the features.

1) Log Transformation of Skewed features

Data visualization demonstrated that bandwidth consumed, bandwidth lost, packets received, received byte, and flood status were not equally distributed and skewed from the normal curve. More reliable predictions can be made if the predictors are normally distributed. Several methods are useful for handling skewed data such as log, square root and box-cox transforms. We applied the log transform to remove this skewness using the Numpy (Numerical Python extensions) library. Numpy provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

TABLE 3: Expert analysis of the training dataset.

Statistic	Utilized Bandwidth	Pack Drop Rate	Bandwidth	Packet Lost Rate %	Lost Byte Rate %	Packet Received Rate	Bandwidth Consumed
Mean	0.59	0.41	540.46	41.16	41.19	0.59	340.78
Std	0.19	0.18	289.14	18.35	18.37	0.18	232.14
Min	0.24	0.09	100	8.61	8.61	0.23	27.55
0.25	0.45	0.25	300	24.75	24.75	0.43	138.4
0.5	0.58	0.44	500	43.8	43.8	0.56	291.59
0.75	0.76	0.56	800	56.67	56.67	0.75	515.18
max	0.93	0.77	1000	76.79	76.79	0.91	867.03

2) Feature Normalization

In this step, scaling was used to standardize the ranges of independent features. The features were rescaled in a way that they acquired the characteristics of a standard normal distribution having a mean value of zero and a standard deviation of one. Feature scaling has a high impact on the results because our training dataset contains features consisting of diverse ranges, units, and magnitudes. In this regard, Mini-Max scaling [41] was performed on the dataset which transformed the values in the range of 0 and 1. Table 4 shows the dataset after normalization.

3) Feature Selection

A Recursive Feature Elimination (RFE) technique was used for feature selection which considers the wrapper method built on top of various other algorithms i.e. SVM or regression. This helped in model development based on different data subsets. RFE repeatedly constructs a model and chooses from the best performing features. The data was split into training and test sets in the form of an array of features where 80% data was used for training and 20% for testing. Overall the dataset consists of 21 features, however the following were not selected because they were either highly redundant or not present in the testing data.

- Average Delay Time Per Sec
- Received Byte
- 10 Run AVG Drop Rate
- 10 Run AVG Bandwidth Use
- 10 Run Delay
- Node Status
- Packets Transmitted

4) Model Training and Hyper-Parameter Tuning

The results of cross-validation are then printed, the model is fit to the training data using slicing with sample size. Input parameters applied to the model are given in Table 5.

C. RESULTS OF TRAINING AND TESTING

In this sub-section, we discuss the results of the training and testing of all the implemented ML algorithms. We use accuracy and F_Beta as evaluation metrics for training given in equation 1 and 2 respectively. In equation 1, TP, TN, FP, and FN corresponds to true positive, true negative, false positive, and false negative respectively.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

F_Beta denotes the weighted harmonic mean of precision and recall. A F_Beta score of 1 is considered to be the best performing value, and a score of 0 corresponds to a worst case scenario. The accuracy score is the ratio between correct predictions divided by the total number of predictions.

$$F\_Beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2}$$

Similarly, the Receiver Operating Characteristic (ROC) curve and the learning curve was plotted for the evaluation of each algorithm. The ROC curve is the relationship between true positive and false positive values, which represents the accuracy of class discrimination in a binary classifier. The Area Under Curve (AUC) is important in the ROC curve, where an AUC value of 1, represents a perfect separation of the classes. The Learning curve shows the cross-validation and training scores of a model on a varying number of samples. It depicts the behavior of a model upon adding more training instances. If the training and cross-validation scores converge on a small training sample then adding more data will not benefit the model.

An evaluation of the training of the classification algorithms is conducted separately to provide a results comparison during training with the data set and subsequently testing using malicious traffic. This is detailed in the next sub-sections.

1) Adaptive Boost (AdaBoost)

AdaBoost utilizes an ensemble learning method to develop a precise learning algorithm. Initially, it chooses a baseline algorithm e.g. Naïve Bayes and iteratively increases its performance by taking into consideration the incorrectly classified instances [42]. AdaBoost's evaluation is illustrated in Fig. 4a and Fig. 4b. Both evaluation metrics on AdaBoost correspond to 1.0 which speaks to the excellent performance of the trained model. The learning curve plot shows that the cross-validation and training scores converge on a sufficient number of training samples. The area under the ROC curve is equal to 1, which shows a perfect separation of the flooding and legitimate classes.

2) Bagging Classifier

The Bagging classifier is one of the bootstrap methods which creates individual estimators towards ensemble by

TABLE 4: Dataset state after applying normalization.

Node	Utilized Bandwidth Rate	Pack Drop Rate	Bandwidth	Packet Lost Rate %	Lost Byte Rate %	Packet Received Rate	Used Bandwidth
3	0.85	0.15	1	0.15	0.15	0.85	0.98
9	0.058	0.94	0	0.94	0.94	0.06	0
3	0.99	0.0062	0.89	0.06	0.01	0.99	0.99
3	0.19	0.81	0	0.81	0.81	0.19	0.083

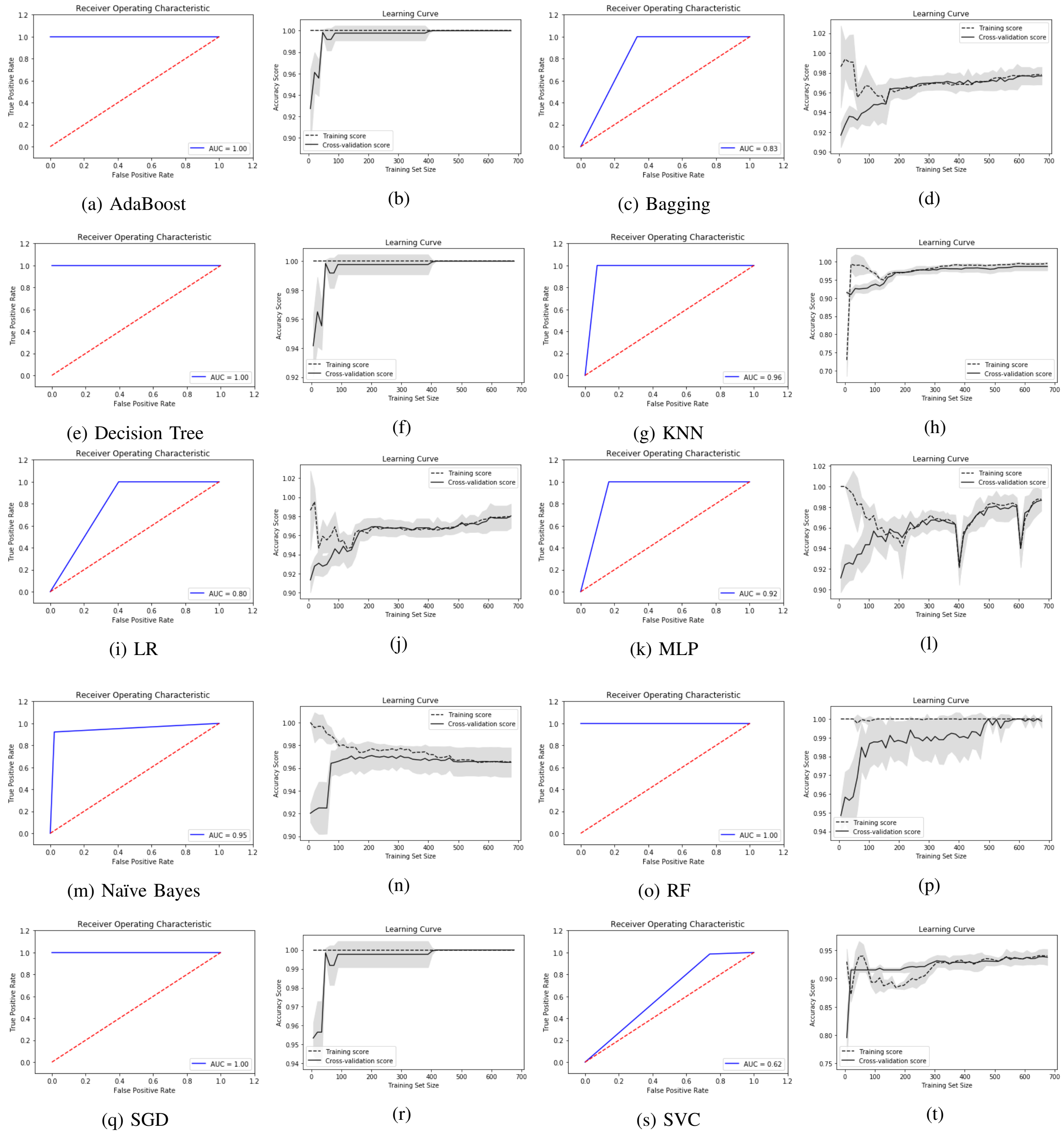


FIGURE 4: Accuracy graphs of ML models.

TABLE 5: Inputs parameters applied to the model.

Input	Detail
Learner	The learning algorithm to be trained
X_train	The features training set
y_train	The training set
X_test	Features testing set
y_test	The testing set

TABLE 6: Evaluation results of ML classification.

Algorithms	F_Beta		Accuracy	
	Training	Testing	Training	Testing
Adaptive Boosting	1	1	1	1
Bagging Classifier	0.98	0.96	0.98	0.96
Decision Tree	1	1	1	1
K Nearest Neighbours	1	0.99	1	0.99
Logistic Regression	0.98	0.95	0.98	0.95
Multi-Layer Perception	0.99	0.98	0.99	0.98
Naive Bayes	0.97	0.98	0.88	0.93
Random Forest	1	1	1	1
Stochastic Gradient Descent	1	1	1	1
Support Vector Classification	0.94	0.91	0.82	0.89

training each of the classifiers and subsequently aggregating their results. It performs this operation by randomly redistributing the instances of the training dataset [43]. The results of the Bagging Classifier can be observed in Fig. 4c, Fig. 4d and Table 6 respectively. The learning curve converges at lower training instances which shows that adding more training data will not benefit the trained model. The AUC value on the ROC curve corresponds to 0.83 which shows good separation of the predicted classes.

3) Decision Tree

The Decision tree is a non-parametric ML technique, used for classification and regression problems. A decision tree provides hierarchical and sequential decisions about the class outcomes on the basis of the training data [44]. The ROC curve in Fig. 4e shows excellent results where AUC is 1, which is a perfect separation of the flooding and legitimate classes. The evaluation values of F_Beta score and accuracy correspond to 1 which demonstrate that the model has very high performance. The learning curve in Fig. 4f expresses the results of cross-validation and training scores. Both curves converge at lower training instances which shows that adding more training instances will not benefit the trained model.

4) K-Nearest Neighbors (KNN)

This is one of the most widely used classification algorithms which predicts the likelihood of a data point to be a member of the group on the basis of the nearest data points [45]. Upon evaluation, KNN provided optimum results having values of F_Beta and accuracy of 1. Fig. 4g and Fig. 4h represents the ROC and learning curve of the KNN model respectively, the learning curve converges around 200 instances of the training set. The ROC curve shows excellent results where the AUC is 0.96, which expresses a perfect separation of the flooding and legitimate classes.

5) Logistic Regression (LR)

The LR classifier predicts the probability of an outcome and generates a logistic curve limited by binary values [46]. Evaluation results of the LR classifier are given in Table 6, Fig. 4i and Fig. 4j respectively. F_Beta and accuracy values for the testing set are around 95%. The cross-validation showed a sharp increase from 92% to 97% when the training set size approached 200 and continued to do so thereafter. The ROC curve shows good results where the AUC is 0.8 which provides a good separation of the flooding and legitimate classes.

6) Multi-Layer Perceptron

A Multi-Layer Perceptron is a feed-forward neural network, which generates a set of outputs characterized by multiple layers of input that are connected as directed graphs between input and output layers [47]. The evaluation metrics for the testing set are greater than 97% which shows the effectiveness of the training model. In Fig. 4l the cross-validation score follows a zig-zag pattern which dropped sharply at two points which represents momentary degradation in accuracy which otherwise increased with the training set size from 91% to 95%. However, when the training set size reached 400, the accuracy dropped instantaneously due to the variations in the accuracy of the training set which subsequently affected the cross-validation. The ROC curve in Fig. 4k represents excellent results where AUC is 0.92, which corresponds to a perfect separation of the flooding and legitimate classes.

7) Naïve Bayes

Naïve Bayes employs the Bayes algorithm to classify objects using the concept of naïve or strong independence among the attributes of data points [42]. The evaluation metrics of the trained model can be observed in Table 6. The accuracy score is 93% for the test set, which is lower than the previously discussed algorithms. In Fig. 4n the learning curve converges at around 500 training instances which demonstrates that the increase in training instances will have a positive effect on the performance of the classifier. The ROC curve in Fig. 4m shows excellent results where AUC is 0.92, showing a perfect separation of the flooding and legitimate classes.

8) Random Forest (RF)

This algorithm develops random decision trees by analyzing a set of variables and generating a class which corresponds to the mean prediction of the individual decision trees [48]. Table 6 shows that the evaluation metrics for the training and testing sets were 1.0 depicting an excellent efficiency of the trained model in accurately classifying the dataset. The ROC curve shows exceptional results where AUC is 1, which is a perfect separation of the flooding and legitimate classes. The accuracy of the cross-validation curve increases continuously and converges with the training score at around 600 instances. The results are given in Fig. 4o and Fig. 4p.

9) Stochastic Gradient Descent (SGD)

The Stochastic Gradient Descent (SGD) is an iterative algorithm to optimize an objective function which is a stochastic approximation of gradient descent optimization [49]. For the SGD classifier, the evaluation metrics corresponds to 1.0. The ROC curve also shows excellent results where the AUC score is 1. The learning curve shows that initially, the accuracy of the test set was lower which increased sequentially with the increase of training data and touched 100% when the cross-validation converged at 400 training instances. The evaluation results of training and testing are given in Fig. 4q, Fig. 4r, and Table 6 respectively.

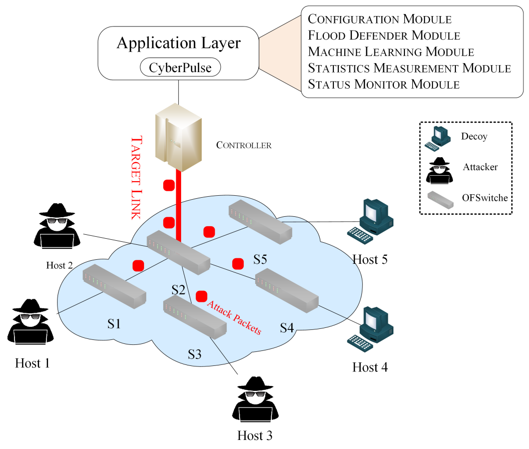


FIGURE 5: An example of a network applying CyberPulse++.

10) Support Vector Classification (SVC)

The Support Vector Classification (SVC) algorithm works by sorting the data into one or two categories and constructs a map of the sorted data by adding possible margins between them. It then classifies an unknown data point based on the distance from the sorted categories [50]. Table 6 shows the evaluation metrics of SVC. The accuracy score is 0.89 which is the lowest among all the implemented models. Overall the evaluation results of this classifier were poor. The value of AUC is 0.62 given in Fig. 4s, demonstrating a poor separation of flooding and legitimate classes. The learning curve (Fig. 4t) converges at around 300 training set instances.

D. SUMMARY OF CLASSIFICATION

A comparative summary of the training and testing results is provided in Table 6 which clearly shows that the AdaBoost, Decision Tree, RF, and SGD classifiers provided better results as compared to the rest. KNN yielded good accuracy, very close to 1. MLP performed a close second with an accuracy of 0.98 and F_Beta of 0.99. There is a vast difference among performance metrics of Naïve Bayes, with the accuracy being 0.9288 as compared to F_Beta which is around 0.9803. These results demonstrate the selection of specific classification method, is problem based and suitable ones are expected to provide better results.

V. CYBERPULSE++ EXPERIMENTAL SETUP

In Section III and IV, we discussed the architecture of CyberPulse++ and its components. In this section, we evaluate the performance of CyberPulse++ in a link flooding environment. We provide testbed configuration, deployment, tools used, and the workflow of CyberPulse++.

A. EVALUATION ENVIRONMENT

We implemented a prototype of CyberPulse++ in Python v3.6 and deployed it on the application plane of SDN, using the REST API to communicate with the controller. Our simulation environment consists of the Floodlight controller v1.2 and Mininet running on the Ubuntu operating system. The design of the network is illustrated in Fig. 5 which imitates a linear topology of 5 servers and 5 OF switches having delay and jitter values of 0. The link capacity of the network was set to 10Mb. CyberPulse++ bootstraps the network and allocates the defined link capacity and checks the connectivity of every link using the ping command. During the experiment, Mininet, Floodlight, and CyberPulse++ run on the same Ubuntu machine. We use an in-band controller paradigm where the controller is not connected with all the switches of the network. The switch ID and the port number settings in the configuration file allow us to select respectively any link on which we want to perform the surveillance and a port associated with the link that we want to secure.

The attack is launched using an Iperf server in a host-client setup, which sends low-rate traffic to the decoys. In our setup hosts 1, 2, and 3 generate the low-rate traffic towards two clients, host 4 and host 5. The LFA causes packet miss in the OF switches which proceed to communicate with the controller for the new flow rule installation. Subsequently, a continuous flow rule installation causes flooding on the control channel which instigates a delay in legitimate traffic flow. The program duration was set to 60 seconds, the statistics collection interval was 5 seconds and the flooding threshold was altered for the evaluation. We use MLP for the traffic classification with a confidence threshold of 95%.

B. CYBERPULSE++ CONFIGURATION

A JSON file (see Fig. 6) provides the user with a flexible interface to input the CyberPulse++ configuration parameters organized into categories. There exists a category containing information regarding controller management and allows assignment of the controller IP, controller and REST API port numbers. The next category is for specifying the statistics collector information including program duration, collection interval, and flooding threshold values. The ML configuration category contains a model name and a confidence level. The model name corresponds to the path for the **.pkl** files generated during training and the confidence

```
{
    "programme_loop_config":
    {
        "rerun":0,
        "rerun_delay_sec":5
    },
    "controller":
    {
        "controller_ip": "127.0.0.1",
        "controller_openflow_port": "6653",
        "controller_rest_port": "8080"
    },
    "stat_collector_config":
    {
        "programme_duration_min":1,
        "collection_interval_sec":5,
        "flooding_threshold_byteCount":1000,
        "generate_graph":1
    },
    "machine_learning_config":
    {
        "model_name":"ML/Pickle_files/MLP.pkl",
        "action":"drop_host",
        "confidence_level":95
    },
    "important_link":
    {
        "switch_id":"00:00:00:00:00:00:00:01",
        "port_number":1
    },
    "traffic_config":
    {
        "link_capacity_in_Mbit":10,

        "switch_count":5,
        "host_per_switch":1,
        "iperf_server":"10.0.0.1",
        "traffic_duration_min":1,
        "iperf_client":[
            {"host_ip":"10.0.0.3"},
            {"host_ip":"10.0.0.4"}
        ]
    }
}
```

FIGURE 6: CyberPulse++ configuration file.

level is a threshold between 0-100 to classify the traffic into legitimate and flooding flows. It may be noted that increasing confidence level results in expanding the false positive rate. The traffic configuration category allows the link capacity, switch count, host per switch, iperf server, and traffic duration to be specified.

C. CYBERPULSE++ IMPLEMENTATION

This subsection includes the implementation set-up and operation of CyberPulse++. CyberPulse++ utilizes a Mininet testbed allowing network researchers to simulate threats in a secure environment and develop, simulate, and deploy security solutions. The ML module provides an interface to train ML classifiers using the training dataset and outputs **.pkl** files. The **.pkl** is an output file generated by Jupyter notebook which enables files to be serialized onto the disc and de-serialized back using a byte stream representing objects. The generation of **.pkl** files reduces network overhead of transferring huge files. The Python *dump ()* and *load ()* methods were used to create and load the **.pkl** files respectively. The ML training module trains 10 algorithms discussed in the previous section. The respective algorithm can be loaded using the *load ()* method in python. Fig. 7 provides the sequence of steps in CyberPulse++ to perform network surveillance and defense.

Once the desired configuration has been provided, CyberPulse++ can be executed and the link listener module starts collecting traffic statistics by employing flow, switch, and packet listeners. The Link listener forwards the collected statistics to the classifier module containing the **.pkl** file of the pre-trained algorithms. Only one algorithm is selected for the classification of the network traffic at any time. The ML module identifies malicious flows using the selected classifier and confidence interval. The flood defender module subsequently drops the malicious flows and the network status monitor draws real-time network status graphs. These graphs effectively depict the impact of LFA on different network parameters including, bandwidth, throughput, delay, and packet drop rate.

D. CYBERPULSE++ DEPLOYMENT

The source files of CyberPulse++ can be downloaded from the link given in the appendix section. The Floodlight controller is executed first and CyberPulse++ source files are loaded on a separate terminal. CyberPulse++ is executed using the **/run.sh** file which starts by creating the network topology and assigning the link bandwidth, jitter, and delay. Then it analyzes the connectivity of the components using the ping command. Subsequently, the traffic is generated by the Iperf server and flooding of the links is started. In the mean time the statistics collector starts to measure the statistics of the network and classification is activated. The flood-drop module drops the flows identified by the classification module. Finally, the network status monitor draws graphs for the visual analysis of traffic on the network.

For statistics collection using Flood light controller, some parameters need to be configured in the Floodlight resource files. These are given in Fig. 8. Steps to run CyberPulse++ are as follows:

- cd floodlight-1.2; make
- sudo rm -rf /var/lib/floodlight/
- cd floodlight-1.2
- sudo java -jar target/floodlight.jar
- unzip mlsdn.zip; cd mlsdn
- ./run.sh; tail -f statcollector.log

VI. RESULTS AND DISCUSSION

In this section, we present and discuss our experiments to measure the performance of CyberPulse++ when defending LFA on the SDN control channel. Due to the novelty of our solution we were unable to find any related works whose results could be directly compared with ours. The mostly closely related works [33], [34] and [35] that utilize machine learning techniques either employ a significantly different implementation architecture or target specific attack types making it hard to create a direct comparison. For example [33] uses an OpenStack based cloud implementation, [34] employ cooperative communications over wifi and [35] examines specific attacks like DoS, HTTP and SSH credential brute force. Therefore, the performance of CyberPulse++

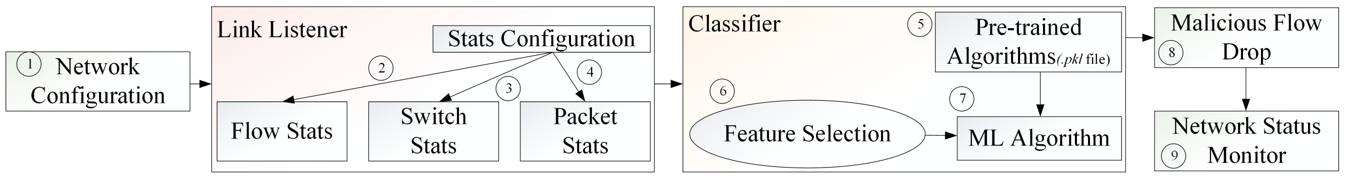
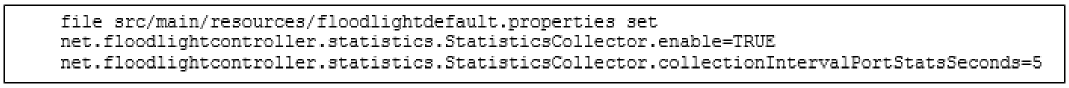


FIGURE 7: CyberPulse++ component interaction diagram.



```
file src/main/resources/floodlightdefault.properties set
net.floodlightcontroller.statistics.StatisticsCollector.enable=TRUE
net.floodlightcontroller.statistics.StatisticsCollector.collectionIntervalPortStatsSeconds=5
```

FIGURE 8: Floodlight code configuration.

has been demonstrated by comparing the results in different experimental configurations. Multiple configuration parameters as well as multiple sources were setup to send LFA. We measure the performance under two threshold frequencies i.e. 50Kbps and 30Kbps and the control channel bandwidth is set to 10 Mbps. As mentioned in the ML Training and Testing section, MLP has a reasonably good degree of accuracy (greater than 97%) when the training set is sufficiently large. This is further corroborated by our previous work [12] where we evaluated the accuracy of different algorithms and determined that the MLP provides the best overall results in terms of precision, recall and F-measure. In the MLP deep learning technique, there are multiple layers including, the input sensory layer, output layer, and one or more hidden layers that collaborate to extract salient features of the problem space. MLP has the ability to model and learn complex non-linear relationships of the given domain. Therefore, it was best suited in our case where some attributes of the network traffic were not linearly dependent on each other such as maximum bandwidth and packet drop rate results. Hence, we perform the evaluation of CyberPulse++ using MLP with a confidence interval of 95%. We run the experiments five times and plot the cumulative standard error during the experiments. The results of the evaluation are as follows.

A. RESULTS FOR END-TO-END DELAY

The graph in Fig. 9a shows the delay incurred during LFA, with the x-axis representing time in seconds and the y-axis corresponding to the delay (in ms) incurred . For delay measurement, we send FTP traffic over TCP using a traffic generation tool to transfer files over the network. We consider all traffic to be part of the calculations including background and foreground traffic. When LFA occurs, the legitimate traffic faces delays which increases source to destination delivery time of the packets. We use the ping command to measure the RTT between the hosts in two traffic intensities and measure the end-to-end network traffic delay during the attack. In the low traffic intensity, the maximum delay incurred was around 173ms. Moreover, when the attack was mitigated, the delay decreased significantly and remained around 71ms at around 90seconds time. The maximum delay observed at the increased attack frequency was around 400ms. Subsequent to the attack defense, the delay became stable around 75ms. This shows that the delay decreased significantly during both the experiments which illustrates the efficiency of CyberPulse++ against LFA.

B. RESULTS FOR BANDWIDTH SATURATION

Bandwidth saturation measures the percentage of link capacity consumption or degradation during the attack by flooding flows. Fig. 9b demonstrates that the bandwidth saturation increases with time. The adversaries initiate LFA traffic towards the targets as soon as CyberPulse++ is executed on the Ubuntu machine terminal. During LFA, bandwidth saturation follows a proportional pattern with respect to time. When the attack was mitigated, the saturation approached a normal level. The figure shows that bandwidth saturation dropped significantly when the attack was mitigated in both the attack thresholds. It remained around 4.5Mbps after the mitigation due to the legitimate traffic in the network which does not pose a threat as it was still lower than the link bandwidth. The impact of bandwidth saturation was continuous which can exhaust the available bandwidth in the absence of mitigation as can be observed in both graphs (Fig. 9b).

C. RESULTS FOR ATTACK DETECTION TIME

When the flooding rate is low, flows in the network may not consume a large amount of network resources thereby allowing the attack to remain undetected and increases the detection time. A large number of attackers consume more network resources and can be easily detected hence, attack detection time will be low. For this experiment, we added more attackers, iteratively increasing the number of attack hosts from 1 through 8. The attack detection time with one host was around 147ms. The flooding intensity increased with the increase in the number of attack hosts due to the increase in the packet drop rate and bandwidth saturation. The traffic intensities are increased which makes the flooding flows prominent and can be detected in a shorter period of time. The attack detection time with 8 attacking

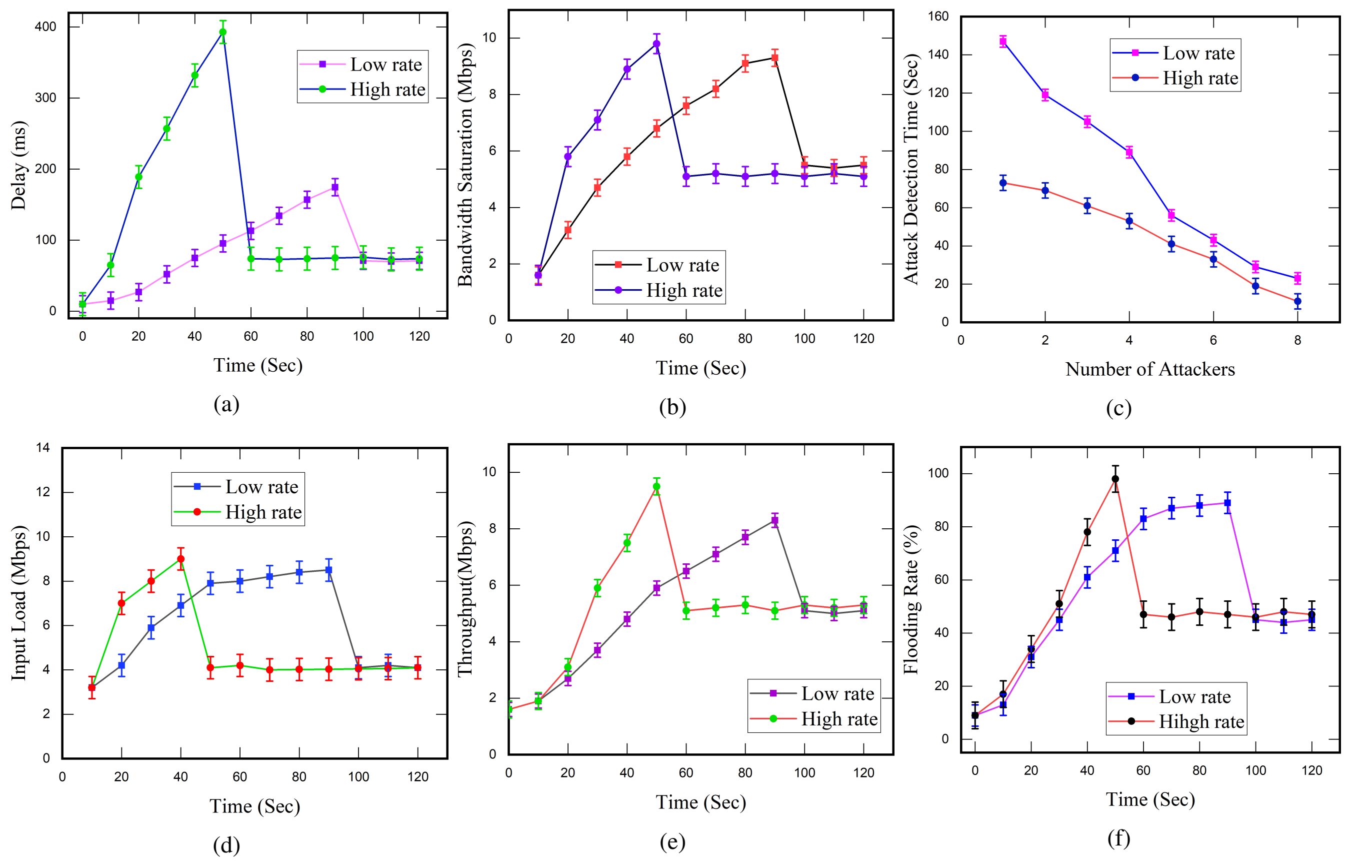


FIGURE 9: CyberPulse++ evaluation graphs.

hosts was around 23ms. Fig. 9c shows that with the increase in the number of attackers the detection time decreases.

D. RESULTS ON INPUT LOAD

Input load denotes the total traffic in the network including flooding and benign. In our experiment, we can observe that the input load increases with the increase in flooding threshold. Fig. 9d shows the effect of flooding on the input load which starts to increase when flood traffic is continuously sent on the control channel link. This is because the adversary was sending carefully crafted packets which cause packet miss in the data plane switches. This instigates new flow rule installation by requesting the controller. Thus a frequent interaction with the controller causes the input load on the control channel to increase significantly. Subsequently, the attack was mitigated which can be observed at the maximum peak points in both of the graphs. Finally, we can observe that there is still a certain amount of input load on the links which is due to the benign traffic in the network. Both graphs share a similar pattern during the experiment.

E. RESULTS FOR THROUGHPUT

Throughput can be defined as the data delivered per second on a specific link. In this experiment, we measure network throughput during LFA and illustrated in Fig. 9e. It can be observed that the data has continuously been delivered on the links which increased the throughput continuously. However, CyberPulse++ was able to detect the attack and mitigate causing the throughput to drop immediately. This can be observed at the peak points in the graph.

F. RESULTS FOR FLOODING RATE

In this experiment, we observe the effect of LFA on the SDN control channel. When flooding occurs, it increases the number of flow rule installation requests to the controller. With a sufficient number of such requests the control channel bandwidth is exhausted and this results in a performance hit. We run the experiment with low and high attack traffic and the results are presented in Fig. 9f. Both scenarios demonstrate that the flooding rate instantaneously rises in the start and approaches a peak value. The peak has been observed because the flooding rate was increasing continuously. CyberPulse++ incurs some delay in first capturing and then classifying the flood traffic. Upon mitigation, a decrease in the flooding threshold is observed.

The experimental results demonstrate that CyberPulse++ effectively defends LFA and provides seamless network operation. Recent solutions against LFA like LinkScope [6] and LFADefender [5] employ measurement agents for network traffic measurements. CyberPulse++ does not rely on agents thus avoiding measurement overhead on the network. CyberPulse++ induces a smaller amount of time in the flood traffic classification which may cause unwanted resource consumption during the start of the attack.

VII. CONCLUSION

Software Defined Networks have revolutionized traditional networking by offering a flexible and centralized solution to managing backbone networks. As ISPs and large organizations increasingly embrace the SDN revolution, new challenges have come to the light. Among these, perhaps the biggest one is that of securing networks against malicious attacks that exploit the inherent vulnerabilities of the SDN paradigm. Towards this end, this paper contributes to state-of-the-art by presenting a novel framework and a complete prototype: CyberPulse++, for the detection and mitigation of LFA in SDN by utilizing ML-based classification techniques. CyberPulse++ is a robust solution that mitigates Link Flooding Attacks in real-time. Existing solutions in the area mandate complex hardware for detection and defense, but CyberPulse++ offers a unique advantage in that it operates on real-time traffic scenarios as well as utilizes multiple ML classification algorithms for LFA traffic classification without necessitating complex and expensive hardware.

Extensive experiments have been conducted to evaluate the efficiency and effectiveness of CyberPulse++ on a testbed. The ML component independently trains ten classification algorithms with high precision. The traffic generation and defense mechanism provide the implementation flexibility to use any of the algorithms to classify the LFA traffic and perform mitigation. CyberPulse++ provides real-time network status graphs which enable efficient network surveillance. CyberPulse++ is a novel yet practical solution which can be easily integrated into existing deployments for securing network environments against LFAs. Due to the loosely coupled components, CyberPulse++ is highly scalable and provides real-time network monitoring.

In the future, we plan to implement CyberPulse++ on large testbeds and extend it by training on multiple datasets for tackling a variety attacks, while simultaneously conducting surveillance and analyzing the network behavior in multiple attack conditions. Additionally, we aim to carry out a comparative performance evaluation of CyberPulse++ against another solution with common attributes. To further improve the performance we will consider pulling the flow manager and classification modules from the controller and performing them as part of a separate software module that runs in parallel with the controller to reduce overhead. If the controller and the additional software run on the same machine, the communication time overhead between the two components should be minimal. Once the efficiency and effectiveness of the CyberPulse++ have been established, we aim to contribute to the research community by releasing the updated prototype binaries as well as the sources. We believe that CyberPulse++ can be converted into an easily deployable and robust multi-layer and multi-attack defense mechanism.

APPENDIX.

CyberPulse++ can be downloaded by the following link:
https://github.com/raihanrasool/Cyberpulse

REFERENCES

[1] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[2] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *In proceedings of the IEEE Symposium on Security and Privacy, (SP)*, 2013, pp. 127–141.

[3] D. Gkounis, V. Kotronis, C. Liaskos, and X. Dimitropoulos, "On the interplay of link-flooding attacks and traffic engineering," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 2, pp. 5–11, 2016.

[4] M. S. Kang, V. D. Gligor, and V. Sekar, "Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks," in *Proc. of the Network and Distributed System Security Symposium, (NDSS)*, 2016, San Diego, CA, USA, pp. 1–16.

[5] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao, and F. Yu, "Detecting and mitigating target link flooding attacks using sdn," *IEEE Transactions on Dependable and Secure Computing*, 2018, In press.

[6] L. Xue, X. Ma, X. Luo, E. W. Chan, T. T. Miu, and G. Gu, "Linkscope: Toward detecting target link flooding attacks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2423–2438, 2018.

[7] A. Studer and A. Perrig, "The coremelt attack," in *In proceedings of the European Symposium on Research in Computer Security*, 2009, pp. 37–52.

[8] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt, "Amplification and drdos attack defense a survey and new perspectives," *arXiv preprint arXiv:1505.07892*, 2015.

[9] T. Hirayama, K. Toyoda, and I. Sasase, "Fast target link flooding attack detection scheme by analyzing traceroute packets flow," in *Proc. IEEE Int. Workshop on Information Forensics and Security (WIFS)*, 2013, Rome, Italy, pp. 1–6.

[10] Q. Wang, F. Xiao, M. Zhou, Z. Wang, and H. Ding, "Mitigating link-flooding attacks with active link obfuscation," *Networking and Internet Architecture*, pp. 1–14, 2017, arxiv.

[11] R. ur Rasool, M. Najam, H. F. Ahmad, H. Wang, and Z. Anwar, "A novel json based regular expression language for pattern matching in the internet of things," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1463–1481, 2018.

[12] R. U. Rasool, U. Ashraf, K. Ahmed, H. Wang, W. Rafique, and Z. Anwar, "Cyberpulse: A machine learning based link flooding attack mitigation system for software defined networks," *IEEE Access*, vol. 7, pp. 34 885–34 899, 2019.

[13] M. Zhou and J. Liu, "A two-phase multiobjective evolutionary algorithm for enhancing the robustness of scale-free networks against multiple malicious attacks," *IEEE Transactions on Cybernetics*, vol. 47, no. 2, pp. 539–552, 2017.

[14] S. Wang and J. Liu, "Designing comprehensively robust networks against intentional attacks and cascading failures," *Information Sciences*, vol. 478, pp. 125–140, 2019.

[15] D. Gkounis, "Cross-domain dos link-flooding attack detection and mitigation using sdn principles," *MSc Th.,Information Technology and Electrical Engineering Department, ETH Zurich, Zurich, Switzerland*, 2014.

[16] X. Ma, J. Li, Y. Tang, B. An, and X. Guan, "Protecting internet infrastructure against link flooding attacks: A techno-economic perspective," *Information Sciences*, vol. 479, pp. 486 – 502, 2019.

[17] M. S. Kang and V. D. Gligor, "Routing bottlenecks in the internet: Causes, exploits, and countermeasures," in *Proc. ACM SIGSAC Conf. on Computer and Communications Security*, 2014, Scottsdale, Arizona, USA, pp. 321–333.

[18] H. Luo, Z. Chen, J. Li, and A. V. Vasilakos, "Preventing distributed denial-of-service flooding attacks with dynamic path identifiers," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1801–1815, 2017.

[19] F. Gillani, E. Al-Shaer, S. Lo, Q. Duan, M. Ammar, and E. Zegura, "Agile virtualized infrastructure to proactively defend against cyber attacks," in *IProc. IEEE Conference on Computer Communications (INFOCOM)*, 2015,San Francisco, CA, USA, pp. 729–737.

[20] P. Xiao, Z. Li, H. Qi, W. Qu, and H. Yu, "An efficient ddos detection with bloom filter in sdn," in *Proc. IEEE Conference on Trustcom BigDataSE, I SPA*, 2016, Tianjin, China, pp. 1008–1015.

[21] A. Aydeger, N. Saputro, K. Akkaya, and M. Rahman, "Mitigating crossfire attacks using sdn-based moving target defense," in *Proc. 41st Conference on Local Computer Networks, (LCN)*, 16, Dubai, United Arab Emirates, pp. 627–630.

[22] L. Wang, Q. Li, Y. Jiang, X. Jia, and J. Wu, "Woodpecker: Detecting and mitigating link flooding attacks via sdn," *Computer Networks*, vol. 147, pp. 1–13, 2018.

[23] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2016, Atlanta, GA, USA, 2017, pp. 1–9.

[24] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *Proc. 45th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, 2015, Rio de Jeneiro, pp. 239–250.

[25] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC conf. on Computer & communications security*, 2013, Berlin, Germany, pp. 413–424.

[26] R. Mohammadi, R. Javidan, and M. Conti, "Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 487–497, 2017.

[27] M. Zhang, J. Bi, J. Bai, and G. Li, "Floodshield: Securing the sdn infrastructure against denial of service attacks," in *Proc. IEEE 17th Int. Conf. On Trust, Security And Privacy In Computing and Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, New York, NY, USA, pp. 687–698.

[28] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, and L. Gong, "Detection and defense of ddos attack based on deep learning in openflow based sdn," *International Journal of Communication Systems*, vol. 31, no. 5, p. e3497, 2018.

[29] T. Wang, Z. Guo, H. Chen, and W. Liu, "Bwmanager: Mitigating denial of service attacks in software defined networks through bandwidth prediction," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, p. 12351248, 2018.

[30] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu, "Realtime ddos defense using cots sdn switches via adaptive correlation analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1838–1853, 2018.

[31] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding ddos mitigation and traffic management with software defined networking," in *Proc. IEEE Int. Conf. on Cloud Networking (CloudNet)*, 2015,Niagara Falls, Canada, pp. 248–254.

[32] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[33] Z. He, T. Zhang, and R. B. Lee, "Machine learning based ddos attack detection from source side in cloud," in *Proc. IEEE 4th Int. Conf. on Cyber Security and Cloud Computing (CSCloud)*, 2017, New York, USA, pp. 114–120.

[34] X. N. Zeng, A. Ghrayeb, and M. Hasna, "Joint optimal threshold based relaying and ml detection in network coded two way relay channels," *IEEE Transactions on Communications*, vol. 60, no. 9, pp. 2657–2667, 2012.

[35] G. A. Ajaeiya, N. Adalian, I. H. Elhajj, A. Kayssi, and A. Chehab, "Flow-based intrusion detection system for sdn," in *Proc. IEEE Symposium on Computers and Communications (ISCC)*, 2017, Heraklion, Greece, 2017, pp. 787–793.

[36] Y.-H. Chen, Y.-C. Lai, P.-T. Jan, and T.-Y. Tsai, "A spatiotemporal-oriented deep ensemble learning model to defend link flooding attacks in iot network," *Sensors*, vol. 21, no. 4, p. 1027, 2021.

[37] X. Chen, W. Feng, N. Ge, and X. Wang, "Defending link flooding attacks under incomplete information: A bayesian game approach," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[38] A. Rajab, C.-T. Huang, and M. Al-Shargabi, "Decision tree rule learning approach to counter burst header packet flooding attack in optical burst switching network," *Optical Switching and Networking*, vol. 29, pp. 15 – 26, 2018.

[39] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: a comprehensive overview," *IET Networks*, vol. 8, pp. 79–99, 2018.

[40] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.

[41] R. A. Cuninghame-Green, *Minimax algebra*. Springer Science & Business Media, 2012, vol. 166.

[42] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *Proc. Springer European Conf. on Machine Learning (ECML)*, 1998,Chemnitz, Germany, pp. 4–15.

[43] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[44] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[45] O. Kramer, *Dimensionality reduction with unsupervised nearest neighbors*. Springer, 2013.

[46] P. Peduzzi, J. Concato, E. Kemper, T. R. Holford, and A. R. Feinstein, "A simulation study of the number of events per variable in logistic regression analysis," *Journal of clinical epidemiology*, vol. 49, no. 12, pp. 1373–1379, 1996.

[47] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[48] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[49] L. Bottou, *Large-scale machine learning with stochastic gradient descent*. Physica-Verlag HD, 2010, Paris France, pp. 177–186.

[50] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," *Taipei: Paper available at http://www. csie. ntu. edu. tw/ cjlin/papers/guide/guide. pdf*, 2003.

•••