CSE 534 Project Report: Implementing Machine Learning-based Intrusion Detection System in Software Defined Network

Yiran Luo, Zhen Zeng
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ, USA
{yluo97, zzeng22}@asu.edu

***Abstract*—We develop an Intrusion Detection System in a Softward Defined Network (SDN) topology using Machine Learning (ML) approaches. Within the time limit of this project, we first establish a specified SDN in the GENI test bed and generate various types of live Denial-of-Service (DoS) attack data. Independently, we offline-train several ML models with a labelled intrusion detection dataset NSL-KDD. We then validate the trained ML models with the generated DoS attack data from the SDN. In the end, we are able to find one model that performs strongly both with the offline dataset as well as the live attack data.**

***Index Terms*—Software Defined Network, Intrusion Detection, Machine Learning**

I. Introduction

In recent years, data-driven machine learning has been adopted, or at least has been studied, in nearly every field in Computer Science, thanks to its versatility and power in solving different problems. Meanwhile, the Internet, being a modern day backbone of global societies, generates a huge amount of data daily. We firmly believe we can also apply this promising technique for networking-related researches, making the best of supervised learning and extracting patterns of various behaviors out of the huge chunks of data we generate every day.

To make the project more concise, we only focus on one aspect, the detection of Denial of Service (DoS) attacks. And due to the general lack of available up-to-date intrusion detection datasets, we train our machine learning models on existing benchmark datasets offline. We also set up our own attack data collection network in the GENI test bed, where we conduct various intrusion experiments and sample our own attack data in order to validate our machine learning models.

II. Related Work

A. Machine Learning in Intrusion Detection System

An intrusion detection system (IDS) is a device or software application that monitors a network or system for malicious activity or policy violations. Generally, IDSs fall into two categories: 1) network intrusion detection, which is to be deployed in some specific points on the network, and is supposed to capture and analyze the stream of packets going through network links [5] by monitoring traffic packets; and 2) host-based intrusion detection, which is to analyze each host separately by monitoring important operating system files. Additionally, based on the type of input data, IDSs are also grouped as signature-based network intrusion detection(SNID) and anomaly-based network intrusion detection (ANID) [4]. The SNIDs detect the potential attacks in the network by comparing the identified patterns or signatures within the input data with a signature database that corresponding to the known attacks [4]. The ANIDs generate an anomaly alarm when detecting the difference between the observed behavior and the expected one that falls below a given limit [4].

Machine Learning (ML) itself is an umbrella term that covers a wide range of tools designed for different tasks in data analysis. Ranging all the way from the statistics-based traditional methods such as linear regression, logistic regression, or Support Vector Machine (SVM), to the more recently developed Artificial Neural Networks (ANN),ML techniques have been active in constant development. Modeling using ANN, also called Deep Learning (DL), in particular, has been widely sponsored by big industrial players, such as Google and Facebook. In recent years, they have founded many easy-to-learn ML libraries, such as TensorFlow and PyTorch, that furthermore push the popularity of machine learning-based techniques.

According to a recent survey by Wang et al. [15], the Machine Learning for Networking (MLN) mindset has also been long established for research in network-related problems, including performance prediction and intrusion detection. Although most of the existing approaches that combines machine learning in intrusion detection are still largely based on the aforementioned traditional methods [14], recent studies in intrusion detection have demonstrated great potential on challenging state-of-the-ark benchmarks by mixing up with Deep Learning techniques [1]. Moreover, related networking studies such as congestion control or resource management [7] [8] have also shown great improvement when deep learning is introduced. We thus firmly believe it is imperative to embrace

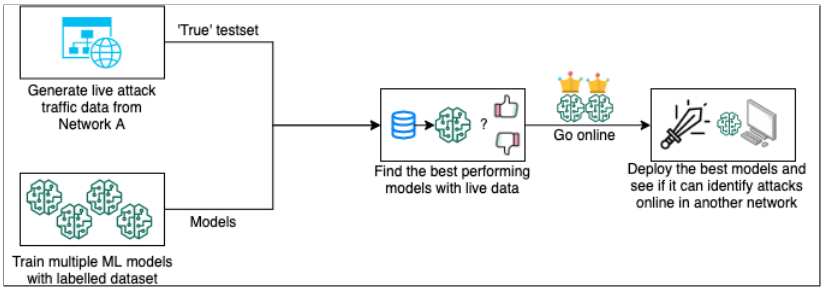

Fig. 1. A crude pipeline of our experiments

Machine Learning techniques and, at least, give it a try in various intrusion detection tasks.

The anomaly-based NIDs consists of three basic modules or stages: 1) Parameterization: where the observed instances of the target system are represented in a pre-established form; 2) Training stage: where the normal/abnormal behavior of the system is characterized and is related to a model; and 3) detection stage: in this stage,data is analyzed in the way of comparing withe the observed traffic. If the deviation found exceeds a given threshold, and alarm will be triggered. The detection techniques can be grouped into three main categories in general: 1) statistical-based: including univariate, multivariate, and time series model; 2) knowledge based: including FSM, description languages, expert systems; and 3) machine learning based: this approach includes all the machine learning algorithms, such as Bayesian networks, Markov models, neural networks, and fuzzy logic etc..

B. GENI platform

The Global Environment for Network Innovations (GENI) platform is a well developed online network research platform with rich education resources for conducting computer network related research. More specifically, two existing studies on network intrusion detection in SDN have been successfully implemented on GENI platform [2], [3]. This platform allow multiple users to work on the same slice, so it enable the collaboration among team members.However, when using GENI for this project, we experienced some limitation. This test bed does not give full administrator authority to user on configuration the virtual machines (VMs). For example, it doesn't allow user to shut down and turn on the interface of OVS, and also does not allow user to change and restart the local resolver of DNS service.

III. Design & Experiment Set-up

For this project, we plan to develop a machine learning-based system to perform intrusion detection in a SDN using GENI. In this section, we go through the core designs of our experiments. Fig. 1 offers a rough pipeline how the experiments are connected, and we define the main tasks as following:

- **Task 1: Machine learning modeling.** We apply supervised learning by training several machine learning classifier models based on a labelled intrusion detection dataset. These machine learning models will be designed for analyzing traffic so as to identify abnormal traffic.
- **Task 2: Constructing SDN network topology.** To construct network architecture, we use GENI as test bed.
- **Task 3: Generating attack traffic data.** The attack traffic data is DoS attack traffic. By labeling these data as abnormal, we construct a dataset for testing the trained machine learning model.
- **Task 4: Deploying machine learning model in SDN.** The trained machine learning model will be deployed in the SDN topology to evaluate its performance. Given the time being, we will only explain this part in theory.

A. Training Machine Learning Models Using a Labelled Intrusion Detection Dataset

This task is achieved by further dividing it into the following 3 steps: Choosing a labelled benchmark dataset, pre-processing the dataset, and choosing some candidate machine learning models.

*1) **The Benchmark Dataset**:* Since our intrusion detection system relies on machine learning pattern recognition, it is natural to apply supervised learning, obtaining certain models by training with a well-labelled intrusion detection dataset. We decide to use the richly labelled NSL-KDD dataset, which is such an improved version of the KDD'99 data-set, by deleting some redundant records in both the training and the test set, and is widely used in training machine learning model for intrusion detection tasks [10].

The 2-part NSL-KDD dataset consists of a training set and a test set. The training set contains 125,973 labelled TCP connection data entries, and the test set 22,544. Each connection describes the behaviour of a sequence of TCP packets within a 2-second capture window. The data are labelled under the following categories:

- **Normal**. This indicates the connection is good. This category takes up roughly 50 percent of all entries.
- **DoS**. Variant Denial of Service attacks, e.g. neptune, ping-of-death, or smurf. This category takes up about 40 percent of all entries.
- **R2L**. Attacks from unauthorized remote access, e.g. passing word guessing.
- **U2R**. Attacks from unauthorized local root user privileges, e.g. buffer overflow attacks. And

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.1.2	10.10.2.2	ICMP	98	Echo (ping) request id=0x0442, seq=1/256, ttl=64
2	5.102565	MS-NLB-PhysServer-3…	02:4a:f5:a2:c0:2d	ARP	42	Who has 10.10.1.2? Tell 10.10.2.2
3	5.138306	02:4a:f5:a2:c0:2d	MS-NLB-PhysServer-3…	ARP	42	10.10.1.2 is at 02:4a:f5:a2:c0:2d
4	38.018997	MS-NLB-PhysServer-3…	02:4a:f5:a2:c0:2d	ARP	42	Who has 10.10.1.2? Tell 10.10.2.2
5	38.022882	02:4a:f5:a2:c0:2d	MS-NLB-PhysServer-3…	ARP	42	10.10.1.2 is at 02:4a:f5:a2:c0:2d
6	72.774923	139.90.229.192	10.10.2.2	TCP	54	1645 → 80 [SYN] Seq=0 Win=512 Len=0
7	72.774932	60.77.114.48	10.10.2.2	TCP	54	1646 → 80 [SYN] Seq=0 Win=512 Len=0
8	72.775500	57.217.63.219	10.10.2.2	TCP	54	1647 → 80 [SYN] Seq=0 Win=512 Len=0
9	72.775503	165.234.219.231	10.10.2.2	TCP	54	1648 → 80 [SYN] Seq=0 Win=512 Len=0
10	72.786349	124.35.31.244	10.10.2.2	TCP	54	1649 → 80 [SYN] Seq=0 Win=512 Len=0
11	72.786354	212.215.235.212	10.10.2.2	TCP	54	1650 → 80 [SYN] Seq=0 Win=512 Len=0
12	72.787024	196.134.204.213	10.10.2.2	TCP	54	1651 → 80 [SYN] Seq=0 Win=512 Len=0
13	72.787029	67.211.90.190	10.10.2.2	TCP	54	1652 → 80 [SYN] Seq=0 Win=512 Len=0
14	72.787032	98.114.165.109	10.10.2.2	TCP	54	1653 → 80 [SYN] Seq=0 Win=512 Len=0
15	72.787735	35.248.187.171	10.10.2.2	TCP	54	1654 → 80 [SYN] Seq=0 Win=512 Len=0
16	72.787740	135.234.63.67	10.10.2.2	TCP	54	1655 → 80 [SYN] Seq=0 Win=512 Len=0
17	72.797623	115.20.128.72	10.10.2.2	TCP	54	1656 → 80 [SYN] Seq=0 Win=512 Len=0
18	72.798048	102.68.171.184	10.10.2.2	TCP	54	1657 → 80 [SYN] Seq=0 Win=512 Len=0
19	72.798051	238.192.4.189	10.10.2.2	TCP	54	1658 → 80 [SYN] Seq=0 Win=512 Len=0
20	72.798706	73.6.171.244	10.10.2.2	TCP	54	1659 → 80 [SYN] Seq=0 Win=512 Len=0
21	72.798710	85.112.249.237	10.10.2.2	TCP	54	1660 → 80 [SYN] Seq=0 Win=512 Len=0

Fig. 2. A TCP SYN flooding attack sample data

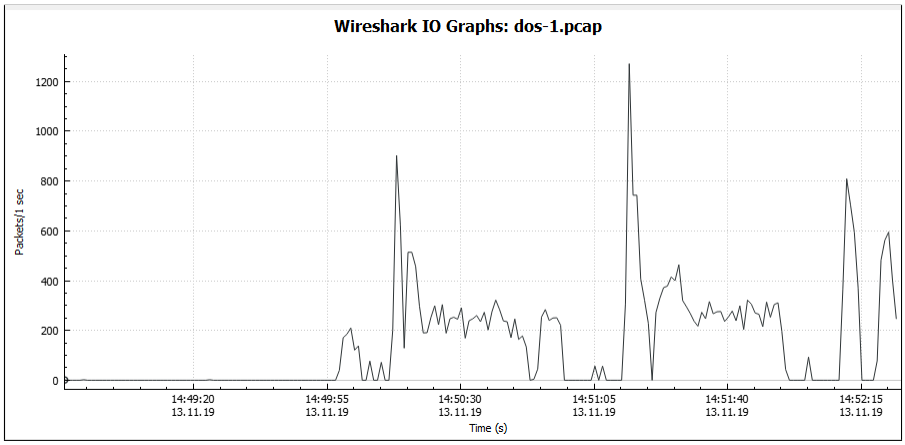

Fig. 3. The Wireshark I/O graph of the TCP SYN flooding attack in Fig. 2

- **Probing**. Probing and surveillance activities e.g. port scanning with nmap.

For the simplicity of the project, we only focus on the DoS attack category in our experiments.

Besides, each data entry consists of 41 descriptive features (columns). These features can be further categorized into the following three classes:

- **Basic TCP features**. Essential properties with regard to a connection, e.g. duration, protocol_type, flag, etc.
- **Content features**. What and how many operations are conducted during a connection, e.g. num_failed_logins, num_shells, is_guest_login, etc.
- **Traffic features**. Statistics with regard to the behavior within 2 seconds, e.g. percentage of SYN error, percentage of REJ errors, etc.

Due to the fact that it is difficult to extract the aforementioned content features from live captured tcpdump files, we abandon the 13 content features and thus ultimately we take in 28 TCP connection features for training our machine learning models.

Out of the 28 taken features, 3 are string-typed, all others numerical. All the numerical values are no less than 0.

*2) **Pre-processing the Dataset**:* Before we may set up the classifier models, we need to further pre-process the NSL-KDD dataset, turning the values into reasonable numerals. We first apply one-hot encoding on the string-typed columns and thus we increase the number of features to 109. Also, we apply max-min normalization on the non-string-typed columns to reduce the total variance.

As for the classification labels, since we only consider DoS attacks in our deployment, we re-label all the data using two binary labelling schemes.

- **Attack-or-not**. If a connection falls under any of the 4 categories of the attacks, it is labelled positive and considered an attack.
- **DoS-or-not**. Only if a connection is labelled DoS, it is labelled positive.

*3) **Candidate Machine Learning Models**:* Eventually, we settle on the candidate machine learning models. Considering the limited time allowed for this project, we choose to train the following 5 binary statistical machine learning classifiers mainly because of their low numbers of model parameters: Logistic Regression (LR), Random Forest (RF), Adaptive Boosting (AdaBoost), Gradient Boosting (GradBoost), and Multi-layer Perceptrons (MLP). All the models are implemented using the Scikit-learn [11] library in Python.

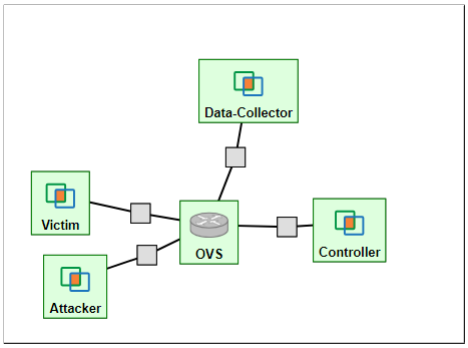

Fig. 4. The SDN topology for collecting data

B. Capturing Attack Data from a Network

After using the public dataset to train the machine-learning model, we then need to collect the actual test dataset from a SDN on GENI. This approach is inspired by a GENI tutorial from Dr. Mountrouidou and her paper on SDN-supported collaborative approach for DoS flooding detection [2], [3]. Their approach is to use the rule-based tool, such as Snort or some self-defined rule-based python script, to detect the attack in the network. From the side of topology design, previously, they use a single Monitor and Correlator to detect a single attacker [3]. Later, they extend their study to multiple Monitors and Correlators to detect the distributed DoS attack [2]. The topology design in study is inspired by these previous study, different from the previous work [2], [3] that only has a TCP SYN flooding attack, more types of DoS attacks have been covered in this study including TCP FIN flooding, TCP RST flooding, TCP PSH+ACK flooding, UDP flooding, and ICMP flooding. We explain the experiment setup in the following 2 steps.

*1) **SDN topology and tool installation**:* In our topology design, we use a POX based controller and an OpenFlow-based OVS to construct a simple SDN. The reason to implement SDN is to copy the traffic among attacker and victim to the independent VM – Data-collector. This topology design is inspired by Dr.X's tutorial on intrusion detection system [9]. In her lab design, the traffic is analyzed by a rule-based intrusion detection system – Snort in the VM of Monitor to detect the attack traffic within the network. In its implementation, the SDN controller duplicate the network traffic to the port of OVS that connected to the VM of Monitor. Meanwhile, the VM of Monitor is connect to the VM of controller, to send signal to controller for future analysis.

In this project, we use the similar way to duplicate traffic of network through OVS to the VM of Data-collector. In the VM of Data-Collector, we run tcpdump command to capture all traffic data. More specifically, in this study, we only generate the DoS attack data. we construct a simple SDN topology with one attacker, one victim,one data-collector, one OVS and one controller. The topology of network shown in Fig. 4 is to test a scenario of Denial-of-Service (DoS) attacks in SDN, which is to disrupt normal traffic of a targeted server, service or network by overwhelming the target with a flood of traffic from multiple sources. Due to the unique capabilities of SDN, the control planes and data planes are separated, so in this project, we are able to duplicate the network traffic to a single VM for future analysis. More specifically, we use the VM of controller to setup the flow table of the SDN control plane in the VM of OVS. After configuration, we bridge the traffic among the VMs of Attacker, Victim and Data-collector, and then duplicate this traffic to the port of OVS that connected with the VM of Data-collector. In this way, we can record all the traffic among attacker and victim in the VM of Data-collector.

In this study, we use an OpenFlow protocol based Open Virtual Switch (OVS) and a POX protocol based controller to construct the key components of SDN. In our topology, there are 5 nodes shown as following:

- **Openflow Virtual Switch (OVS):** The OVS is a virtual multilayer network switch, which supports the network automation through programmatic extensions.
- **Controller:** The controller is one of the key components of SDN architecture, which works as brain of the switch. It can manage the flow direction, drop or duplicate flow traffic on the switch. In this project, we use controller to duplicate the network traffic to a single VM to record traffic data.
- **Attacker:** The malicious user in the network, who performs the DoS attack on the victim. The attacker's IP is 10.10.1.2/16.
- **Data-Collector:** This VM is to capture and save network traffic in a .pcap file. The data-collector's IP is 10.10.3.2/16.
- **Victim:** The victim of DoS attack. The victim's IP is 10.10.2.2/16.

In addition, we also need to install several tools in our network for different purposes:

- **SDN foundation: 1) POX** is a networking software platform written in Python, and works as an OpenFlow controller; The POX used in this study is POX 0.5.0. **2) OpenFlow**: it is a protocol that installed on OVS to give access to the forwarding plane. The Open vSwitch used in this study is Open vSwitch 2.3.1 with OpenFlow 1.1.
- **DoS attack generation: 1) iperf** : used to set up a server listening to a certain TCP port that receives an attack; **2) hping3**: Flooding the victim server with a high number of ping packets; **3) tcpdump**: Listening to and capturing all the packets that goes through the flooded link.
- **Traffic Analysis: 1) WinSCP**: Downloading the data captured in GENI to local computer. **2) Wireshark:** Generating the I/O graphs with regard to every attack conducted.

*2) **Generating Attack Data**:* In this study, we demonstrate the DoS attack which is similar to the previous study on SDN-supported approach for DoS attack detection [2]. Besides generating the same TCP SYN flooding attack used by that study, we extend the types of DoS flooding to a broader

Attack Types	Packet Size	Packet Rate	Source IP Range
TCP SYN flooding	small	high	large
TCP ACK & PUSH flooding	small	high	large
TCP RST flooding	large	high	large
TCP FIN flooding	large	high	large
UDP flooding	small	very high	very large
ICMP flooding	variable	very high	very large

TABLE I
THE COMPARISON OF DOS ATTACKS COVERED IN THIS STUDY [13]

Attack Name	Description	Duration
dos-1	TCP SYN flooding with random source IP, twice	60s for each segment
dos-2	TCP FIN flooding, then TCP RST flooding	60s for each segment
dos-3	TCP PSH+ACK flooding, then UDP flooding	60s for each segment
dos-4	ICMP flooding	60s
dos-5	TCP SYN flooding with random source IP, twice	120s for each segment
dos-6	TCP PSH+ACK flooding	120s
dos-7	ICMP flooding	120s

TABLE II
LIVE DOS ATTACKS GENERATED

range, which covers more types of DoS flooding attacks. The following parts, we summarize the attack type generated in our GENI platform, which works as data to test our machine learning model. An attack log is used to record the process of generating DoS attack traffic in this study. In this study, we perform 6 types of DoS attacks that in the network layer (layer 3) and the transport layer (layer 4). Table I shows the comparison of these 6 DoS attacks [13]. These attacks include the TCP based DoS attack, UDP based DoS attack and the ICMP based DoS attack, the packet size ranges from small to large, the packet rate covers high to very high, and the source IP range covers large to very large. We explain these DoS attacks in more detail in the following part:

- **TCP SYN flooding:** it works at the transport layer. The attacker sends lots of SYN packets to overwhelm the victim server, thus, the legitimate user can not set up TCP connection with this victim server.
- **TCP PSH & ACK flooding:** it works at the transport layer. The attacker sends a bunch of PUSH and ACK packets to the victim server, so that the victim server can not respond to the legitimate requests.
- **TCP RST flooding:** it works at the transport layer. The attacker sends RST packets with proper values(eg, source IP, destination IP, source port, destination port, sequence number etc.) to kill the TCP connection between source and destination. This attack makes it is impossible to establish TCP connection.
- **TCP FIN flooding:** the attacker keeps sending junk FIN packets to consume the resource of server, since the server needs some resource to process each package to see if the package is redundant.
- **UDP flooding:** the attacker creates and sends large amount of UDP datagrames from spoofed IPs to the victim server. This attack consumes network bandwidth with sending ICMP "destination unreachable" packets.
- **ICMP flooding:** it is similar to UDP flooding. The attacker creates and sends large amount of ICMP packets to make the server overwhelmed from responding to the legitimate request.

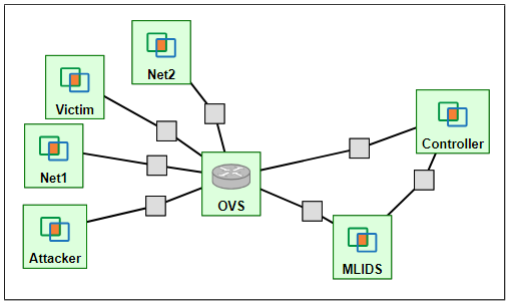

Fig. 5. The SDN topology for deploying ML-IDS

We use an attack log to track and record the attack data that generated in this project. Table II summarizes the attack data that used for testing the machine learning model. When generating attack data, we combine different attacks together, and run attack for 60s and 120s separately to capture as many as possible features of attack for future analysis. Fig.2 and Fig. 3 shows the insight of captured TCP SYN flooding attack. The attacker (IP: 10.10.1.2/16) creates and sends a large amount of TCP SYN packets from various of fake IPs to the victim (10.10.2.2/16), thus the victim server can't response to the legitimate request.

Once the training of all models are done, the models are converted into binary files and ready for re-deployment.

C. Live Data Extraction and Testing

We orchestrate the 7 DoS attacks aforementioned in Table II in the data collector SDN. Once we capture their tcpdump files, we then use a 3rd-party tool to convert the tcpdump packets into connection data in the exact NSL-KDD format [12]. Since we do not care about normal connections, all connections

labelled as normal are dropped and the remaining ones are all labelled as DoS attacks. These converted live attack data serve as the *true* test sets in order to validate the performance of our trained machine learning models. We feed these *true* test sets into the trained models obtained in Task 3 without tuning the parameters of the models.

D. Deploying the Intrusion Detection System

Due to time limit of this project, we are unable to experiment with deploying the ML-based models in an actual SDN. But in theory, to deploy the ML-based intrusion detection system, we construct an another SDN with two subnets as shown in Fig. 5. The topology design in study is inspired by these previous study in DoS detection in SDN [2], [3]. In this study, since the target attack is DoS attacks, we only adopt the single monitor-and-correlator architecture [3]. The machine learning-based model is deployed in the VM of MLIDS. This VM of MLIDS connects to controller and OVS directly. The controller duplicates all network traffic to the port of OVS that connected with MLIDS, and the machine learning-based IDS monitors and analyzes the incoming traffic and signal any anomaly detection results to the controller. Once any attack is detected, the controller shuts down the port that connect with attacker or drop all traffic that comes from malicious sources.

IV. Experimental Results

In this section, we report the actual experiments in developing the intrusion detection system. The first part includes training/testing the machine learning models exclusively with the NSL-KDD dataset. And the second part describes how the trained systems perform with the live-captured attack data from the SDN.

A. Offline ML Model Training

We train the selected 5 models under the two labelling schemes: Logistic Regression (LR), Random Forest (RF), Adaptive Boosting (AdaBoost), Gradient Boosting (GradBoost), and Multi-layer Perceptron (MLP). Their accuracies when applying the NSL-KDD's test set are shown in Table III. Judging from these offline test results, we may tell that the 5 models do not vary too much on the NSL-KDD dataset for the sake of accuracy, and therefore validating tests with live data become necessary.

B. Testing with Live Attack Data

Using the true test tests from the 7 DoS attacks defined in Table II, we are able to obtain the test results in Table IV and in Table V. Most noticeably, the performance of Gradient Boosting remain outstanding in both labelling schemes, granting near-perfect accuracy with all the live attack data in both label schemes. We also notice some classifiers have 0% accuracy in several occasions, e.g. RF with Attack-or-not labels and AdaBoost with DoS-or-not labels. This indicates such models may entirely miss the mark when placed in an actual network under attack.

Scheme\Model	LR	RF	AdaBoost	GradBoost	MLP
Attack-or-not	0.7663	0.7671	0.7720	0.7788	0.8056
DoS-or-not	0.9234	0.8952	0.9068	0.9352	0.9238

TABLE III
ATTACK-OR-NOT AND DOS-OR-NOT ACCURACY OF EACH CLASSIFIER USING THE ORIGINAL TEST SET FROM NSL-KDD

Attack\Model	LR	RF	AdaBoost	GradBoost	MLP
dos-1	0.1854	0.0000	0.6513	**1.0000**	0.0000
dos-2	0.2012	0.0000	0.9813	**1.0000**	0.0000
dos-3	0.1667	0.0001	0.9784	**1.0000**	0.0000
dos-4	0.0354	0.0001	0.9875	**1.0000**	0.0000
dos-5	0.1369	0.0000	0.6908	**0.9997**	0.0782
dos-6	0.0303	0.0002	0.2335	**1.0000**	0.0000
dos-7	0.0987	0.0000	0.7566	**1.0000**	0.0000

TABLE IV
ATTACK-OR-NOT ACCURACY WITH THE TRUE TEST SETS

Attack\Model	LR	RF	AdaBoost	GradBoost	MLP
dos-1	0.9998	0.0000	0.0000	**0.9999**	0.4970
dos-2	0.8562	0.6988	0.0000	**1.0000**	0.0006
dos-3	0.8350	0.7207	0.0000	**1.0000**	0.0011
dos-4	0.9460	0.8552	0.0000	**1.0000**	0.0002
dos-5	0.8933	0.0000	0.0000	**1.0000**	0.5115
dos-6	0.0878	0.0626	0.0000	**1.0000**	0.0062
dos-7	0.9999	0.0000	0.0000	**1.0000**	0.2433

TABLE V
DOS-OR-NOT ACCURACY WITH THE TRUE TEST SETS

V. Conclusions & Future Plan

In this project, we have successfully built a Denial-of-Service intrusion detection system with machine learning-based classifiers. We have trained 5 machine learning-based models offline with the NSL-KDD dataset. We have generated and captured different live DoS attack data after establishing an intrusion data collector SDN in GENI. And we have validated all the trained models with the captured attack data. Ultimately, we are able to locate one model, Gradient Boosting, that performs extremely well within the scopes of the NSL-KDD dataset and the live attack data we have designed.

Due to the time restriction, we are yet to fully deploy our best model into the ML-IDS network topology. We still need overcome the consistency bottleneck to accurately translate live network traffic packets into machine learning-friendly data in the NSL-KDD dataset format, which our machine learning models only accepts as of now. In future, we plan to keep working on this part to perform the machine-learning based network intrusion detection. And we may further more extend this project to the Distributed DoS (DDoS) attack scenarios with more complex network configurations.

References

[1] Almseidin, Mohammad, et al. "Evaluation of machine learning algorithms for intrusion detection system." 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY). IEEE, 2017.

[2] Chin, T., Mountrouidou, X., Li, X. and Xiong, K., 2015, October. An SDN-supported collaborative approach for DDoS flooding detection and containment. In MILCOM 2015-2015 IEEE Military Communications Conference (pp. 659-664). IEEE.

[3] Chin, T., Mountrouidou, X., Li, X. and Xiong, K., 2015, June. Selective packet inspection to detect DoS flooding using software defined networking (SDN). In 2015 IEEE 35th international conference on distributed computing systems workshops (pp. 95-99). IEEE.
[4] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G. and Vázquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. computers security, 28(1-2), pp.18-28.
[5] Hodo, E., Bellekens, X., Hamilton, A., Tachtatzis, C. and Atkinson, R., 2017. Shallow and deep networks intrusion detection system: A taxonomy and survey. arXiv preprint arXiv:1701.02145.
[6] "KDD Cup 1999 Data." https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[7] Li, Wei, et al. "QTCP: Adaptive congestion control with reinforcement learning." IEEE Transactions on Network Science and Engineering (2018).
[8] Mao, H., et al., "Resource Management with Deep Reinforcement Learning," Proc. HotNets 2016, pp. 50–56.
[9] Mountrouidou, X., Intrusion Detection Lab, http:// mountrouidoux. people.cofc.edu/ CyberPaths/ intrusiondetectionsystem _v2.html
[10] "NSL-KDD dataset" https://www.unb.ca/cic/datasets/nsl.html
[11] Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.
[12] Rajasinghe, Nadun Samarabandu, Jagath Wang, Xianbin. 2018. INSecS-DCS: A Highly Customizable Network Intrusion Dataset Creation Framework. 10.1109/CCECE.2018.8447661.
[13] Riorey, Types of DDoS attacks, http://www.riorey.com/types-of-ddos-attacks/
[14] Sun, Y., et al., "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction," Proc. SIGCOMM 2016, ACM, pp. 272–85.
[15] Wang, Mowei, et al. "Machine learning for networking: Workflow, advances and opportunities." IEEE Network 32.2 (2017): 92-99.