A Defense System for Defeating DDoS Attacks in SDN based Networks

Adel Alshamrani
Arizona State University
Tempe, Arizona
aalsham4@asu.edu

Ankur Chowdhary
Arizona State University
Tempe, Arizona
achaud16@asu.edu

Sandeep Pisharody
Arizona State University
Tempe, Arizona
spishar1@asu.edu

Duo Lu
Arizona State University
Tempe, Arizona
duolv@asu.edu

Dijiang Huang
Arizona State University
Tempe, Arizona
dijiang@asu.edu

ABSTRACT

Software-Defined Networking (SDN) is a network architecture that aims at providing high flexibility through the decoupling of the network logic from the forwarding functions. The ease of programmability makes SDN a great platform implementation of various initiatives that involve application deployment, security solutions, and decentralized network management in a multi-tenant data center environment. Although this can introduce many applications in different areas and leads to the high impact on several aspects, security of SDN architecture remains an open question and needs to be revisited based on the new concept of SDN. Current SDN-based attack detection mechanisms have some limitations. In this paper, we investigate two of those limitations: *Misbehavior Attack* and *NewFlow Attack*. We propose a secure system that periodically collects network statistics from the forwarding elements and apply Machine Learning (ML) classification algorithms. Our framework ensures that the proposed solution makes the SDN architecture more self-adaptive, and intelligent while reacting to network changes.

CCS CONCEPTS

• **Networks** → **Denial-of-service attacks**; Denial-of-service attacks;

KEYWORDS

Software-Defined Networking; Machine Learning; DDoS mitigation; Misbehavior attack

MobiWac'17, November 21–25, 2017, Miami, FL, USA

ACM ISBN 978-1-4503-5163-8/17/11…$15.00
https://doi.org/10.1145/3132062.3132074

1 INTRODUCTION

Software-Defined Networking (SDN) is an emerging network architecture proposed to change the limitations of current network infrastructures. It decouples the network control logic and the network traffic to introduce the concepts of control plane and data plane. The control logic (controller) becomes centralized to manage the network logic and decision making process, while the data plane devices (network switches) become simple forwarding devices. In SDN, the controller makes the decision on how the traffic in the network should be managed and then push the decisions as flow rules to the data plane devices. Although this separation leads to many advantages such as high degree of flexibility, which is one of the main reasons for the widespread adoption of SDN even amongst big companies [1], ease of the adaptation of other techniques, programmability, scalability, etc., it can be the weakest point when its components, such as the SDN controller, OpenFlow enabled switches, are the main target of the attack since that can affect the whole network [2].

With the significant momentum toward adapting the SDN in the networking systems, many security defense mechanisms have been proposed. However, current attack detection mechanisms that are SDN-based, have some limitations. In this paper, two serious security issues are brought to light:

- *Misbehavior Attack*: Current SDN-based attack detection mechanisms, which are traffic inspecting based, usually inspect first packet of each flow and make processing decision to handle the following packets in the same flow. Thus, if the first packet represents normal behavior, then the following packets in the same flow will be considered normal traffic. However, if an attacker exploits this point by acting normally at the beginning and later sending malicious contents in the following packets. Then he/she can launch what we call *Misbehavior Attack*.
- *NewFlow Attack*: In SDN-based environment, if the switch cannot locate a flow that matches the packet, it sends the packet to the controller as a *packet_in* for instruction. Therefore, attackers can exploit this new feature in SDN paradigm to generate lots of unmatched new flows to the SDN-enabled switches. As consequence, this will result in generating uncontrolled

packet_in messages to the controller for instructions, which may result in DoS attack.

Because we assume the misbehavior attack is launched by fooling the defense system with legitimate traffic, but later, attackers act maliciously, then we need our system to distinguish between legitimate and malicious traffic. Therefore, we identify both legitimate and malicious traffic through the deployment of anomaly detection of DDoS using Machine Learning (ML) techniques. Detecting and preventing DDoS attacks is not a new research area and huge amounts of research has been conducted [3–6]. However, what is different in this paper is that, unlike most of the existing ML based approaches [7–11], we are using extensive range of prediction features to ensure covering more types of DDoS attacks as well as ensuring better DDoS detection accuracy.

In this paper, our framework considers three security modules which are: *typical DDoS attack detection module, misbehavior attack detection module, and new flow attack detection module*. It is important to note that these attacks have the same potential effect which cause performance degradation and massive interruption of the target system. Our designed framework is not restricted to wired or wireless infrastructure, it can support both as long as they are SDN-enabled infrastructures. Our contribution can be summarized as follows:

- We present the impact of integrating SDN and ML to support security mechanisms for detecting and mitigating DDoS attacks and evaluate their integration through a study case. Our SDN/ML defense system is based on observed knowledge from the network traffic and achieve a classification accuracy of 99.40% when training and testing on NSL dataset.
- We design another two defense modules to detect the introduced new attacks: *Misbehavior Attack* and *NewFlow Attack*. Both are implemented in SDN environment contains Open vSwitch and POX controller, and multiple VMs. Our experiment demonstrates the attack detection accuracy and how our defense modules interact with the SDN controller.

The structure of the paper as follows: Section 2 will discuss the related works. In Section 3 we discuss our motivation to propose this work. The system details is described in Section 4. The experimental results is illustrated in Section 5. Finally, Section 6 concludes our work and provides the future work.

2 RELATED WORK

Distributed denial of service (DDoS) attack has been well studied and many approaches have been proposed over the last few years to counter it. However, most of these works target the traditional network, which is not the focus of this work. Our focus is oriented toward security issues with respect to SDN based infrastructure. Although SDN shares several of the same concepts as traditional networks, it has its own unique traits [12, 13]. Current works to detect DDoS attack can be broadly categorized under two types of detection mechanisms, either based on packet inspection or flow entries inspection. In [14], Giotis et al. proposed an anomaly based detection method, with the proposed mechanism basically inspecting the flow entries and applying an entropy-based algorithm to analyze the collected information. Consequently, detected abnormal traffics lead to blocking the source of the attack.

In [15], authors aimed to handle intrusion and DDoS attacks in SDN environment applying machine learning techniques. However, they only analyzed various machine learning techniques, such as Support Vector Machine (SVM), fuzzy logic, decision tree, neural networks, and Bayesian networks, which can be used to detect DDoS attacks in the networking system and no further explanation of how to detect and mitigate DDoS attack was given.

Jankowski et al. in [7] presented an intrusion detection approach using self-organized maps (SOM) as a Machine Learning (ML) method. This approach is based on inspection of flow entries, where eleven features extracted from flow entries generated by multiple Virtual Machines (VM). However, this approach focuses more on the features of flows and ignores features that are related to the attack traffic.

Wang et al. in [8] proposed DDoS attack detection model based on a graph model. In this model, known attack patterns are stored as a relational graph between patterns and their labels to distinguish between normal and abnormal traffic. Once the abnormal traffic is detected, then the sources of the attacks are blocked. Authors in [9] proposed an approach to detect the DDoS attack that targets the SDN controller resources. The proposed solution is based on analyzing the entropy variation of the destination IP address and probability calculation between different hosts. Although the proposed solution tries to solve attack detection, it introduces additional overhead on the controller which can make it more vulnerable to future DDoS attacks especially as there was no prevention countermeasure considered.

Li et al. in [10] proposed prototype called Drawbridge that makes the customers to subscribe to the desired traffic provided by ISPs to apply the rules enforced by the ISPs on their SDN switches. Work presented in [11] and references therein focus on attacks targeting the SDN controller only. Although the SDN controller might be the most preferred point of attack, other entities in the SDN paradigm can lead to severe issues due to the broad attack surface. For instance, attacks could target the southbound interface between controller and SDN switches, or vulnerable applications.

To that end, we propose an approach that considers the major limitations, discussed in Section 3, in a SDN based environment.

3 MOTIVATION

New security defense systems designed to detect and mitigate security threats in SDN based infrastructure have been recently adopted in literature [16–18]. However, most of the existing approaches have severe limitations such as:

A) inspecting only new flow, packets that do not belong to existing flows, and based on the inspection result the SDN

controller generates flow rules and instructs SDN-enabled switches to install them. However, if an attacker, by penetrating a targeted system, finds out that the targeted system is deploying such a defense strategy, then he/she can act normally at the beginning of the flow rule deployment by sending first packets of the flow as legitimate traffic, to fool the defense mechanism, and once the flow rules have been installed he/she can act maliciously and change his/her behavior.

B) Flow entries usually are removed from flow tables in either two ways: the controller requests a switch to do so or via implementing flow expiry mechanism at the switch. The switch flow expiry mechanism is run independantly by the switch and based on the state and configuration of flow entries [19]. The flow expiry mechanism comes in two forms: *idle_timeout* and *hard_timeout*. If the value of either form, is non-zero, then the switch must monitor the flow entry's arrival time, as it may need to remove the entry later. We found this is a potential threat, in SDN-based environment, that can be exploited by attackers especially if they can determine the exact value of either *idle_timeout* or *hard_timeout*. A knowledgeable attacker of flow entry's timeout, can maintain space in the flow table by only keeping his flow within the timeout. As consequence, the flow table installed in OpenFlow Switch (OFS) has large risk to be overflowed due to un-removed and faked flow entries.

C) Existing DDoS detection mechanisms usually deploy periodic triggers to start the detection of DDoS attack. However, periodic triggers based mechanisms have some limitation in regard to considering how long or short the chosen period (time window) should be. Another possible way to start DDoS detection is to consider inspecting all new flow through the receiving of *packet_in* messages which is only designed in SDN paradigm. However, the problem with *packet_in* messages is that attackers can flood the SDN-enabled switches with large number of new flows that result in unmatched flow at the receiving switch which then result in generating uncontrolled *packet_in* messages to the controller plane for instructions. Thus, utilizing the *packet_in* as a trigger should be carefully deployed to avoid such a problem. We argue that if this problem is left unsolved then deploying DDoS attack detection mechanisms will be useless because of flooding the SDN-based network with new flow attack can undermine the deployed defense systems.

D) Considering only a few types of DDoS attack such as flooding based attacks: TCP SYN flood, UDP flood, ICMP flood, etc.

4 SYSTEM DESCRIPTION

Our framework as shown in Figure 1 consists of several modules that together achieve the three previously mentioned motivations, detection of: *typical DDoS attack, misbehavior attack, and new flow attack.* These modules are precisely explained in the rest of this paper.

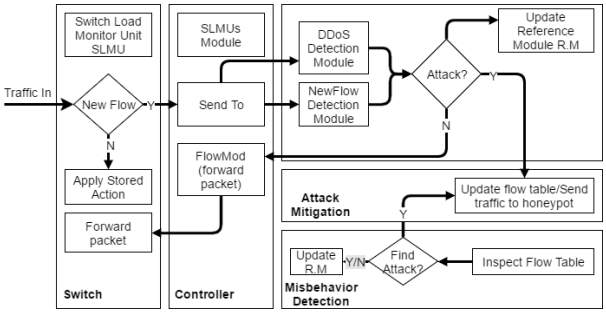

Figure 1: System Architecture.

4.1 Typical DDoS Attack Detection

When a packet arrives at the switch, the switch checks its flow table entry and if there is a rule assigned to it, then it takes the stored action. Otherwise, the *packet_in* message will be sent out to the controller. In our system, the *packet_in* message triggers the DDoS attack detection module resides in the controller. Upon receiving the trigger, the controller will obtain the information of the received packets and extract the predefined features which will be sent to the trained ML model to classify if the traffic is malicious or not.

Based on the predication's result of the trained ML model, the controller will be notified with the necessary information such as (IP source address, IP destination address, source port number, destination port number, and the protocol type) to take an action. However, if the trained model cannot distinguish either the received packet is benign or malicious, then the traffic will be considered as suspicious (unknown). The controller will instruct the switches to forward unknown traffic to the deployed honeypot until a clear decision is made. Meanwhile, the controller will forward unknown traffic to the deep investigation module where we deploy the unsupervised learning (clustering technique). The clustering technique helps to summarize and explain key features of the unknown traffic and compare it with the classes that are already known by the trained model. Therefore, we can distinguish based on the traffic if the new cluster shares similar features with known attack classes or not. Based on the obtained result, the SDN controller updates the Open vSwitch (OVS) with the new flow rule, and update the ML trained model as well. Our detection module includes two algorithms for offline dataset processing and ML model generating, and online DDoS detection that are illustrated in Algorithm 1 and Algorithm 2, respectively. Figure 2 depicts the process flow of our DDoS attack detection module.

To ensure better anomaly detection of DDoS attack, we carefully focus on the feature selection part, so our ML model is trained on most effective features. During the designing of our ML model, we tried multiple supervised algorithms for better detection accuracy. Our machine learning module covers the feature selection and the ML model training, testing, and prediction.

Algorithm 1: Offline DDoS Detection

```
Procedure OfflineDetect()
    Analyze traffic collected in the dataset
    Select multiple features as recommended by Weka
    Train model with the output values generated in Step 2
    Store attacks patterns in local database
```

Algorithm 2: Online DDoS Detection

Input : Packets p, MLFeatures f
Output : $FlowTable\ t$

```
Procedure OnlineDetect()
    FlowTable t ← ∅
    predictionResults ← MLModel (p, f)
    t ← FlowRuleGenerate (predictionResults)
    trainingModel ← updateModel (trainingModel, t)
    return t
Procedure MLModel()
    predictionResults ← DetectAttacks (analyze (p, f), trainingModel)
    return predictionResults
```

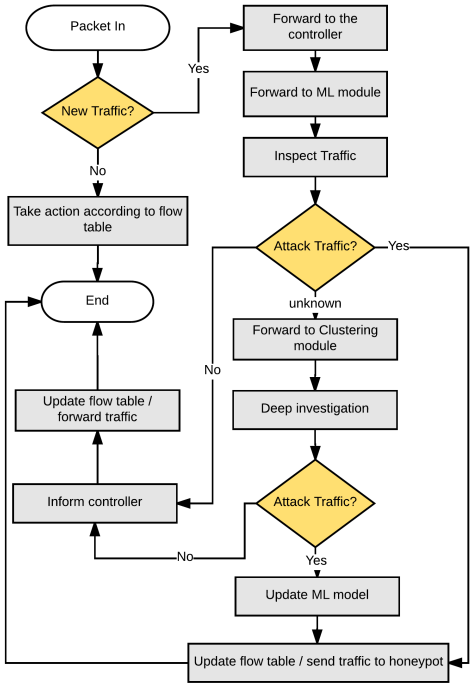

Figure 2: Attack detection module workflow.

4.1.1 Feature Selection. The dataset used in this work, NSL [20], contains forty one features as shown in Table 1. Instead of using all these features which would increase the computational complexity while training and decrease detection accuracy, only some are selected that are observable and their values can be directly extracted from the flow statistics or the packet content. Our feature selection technique ensures the effectiveness of the selected subset of features in distinguishing between normal and abnormal traffic. Our feature extractor module ensures all predefined features are extracted from each flow either from the first packet or from few more packets. We run three feature selection algorithms, namely ranker algorithm, genetic algorithm, and greedy algorithm to obtain the best fit subset of features. The details of these algorithm can be refereed to [21–23], and it was skipped due to space constraints.

If f_a is the set of all forty one features in NSL, there are f features that can be selected to ensure the best model comes out. Out of these f possible features, we fit our model containing n most frequently selected features by all subset selection algorithms, which is possible in $\binom{f}{n}$ ways. Thus, we can select the features that best match the ranking algorithm as $f_x = \{f_1, f_2, \dots f_n\}$. Similarly, we obtain the best match features for genetic algorithm, and denote it f_y and greedy algorithm denoted as f_z. Once f_x, f_y and f_z are obtained, the complete list of selected features is simply $f_f = \{f_x\} \cup \{f_y\} \cup \{f_z\}$. Algorithm 3 shows this logic in pseudocode.

Algorithm 3: Best Subset Selection

Input : Initialize all selection algorithms = $\{alg_1, alg_2, ..., alg_n\}$
Output : finalFeatureSet f_f

```
Procedure bestSubsetSelect()
    final feature set f_f ← ∅
    while overall do
        for each feature of group g_n do
            f_a ∈ g_n ← {f_1, f_2, ..., f_b}
            Select a set of features f_x, f_y, f_z from f_a such that f_x,y,z ⊆ f_a
            Feature selection is done using three search algorithms
        for each group g_i ∈ {g_1, g_2, ..., g_m} do
            \\Best Subset Selection Step
            Pick the features appear in all groups
            f_f = {f_x ∪ f_y ∪ f_z}
            Add the selected features into the Best Subset
            Calculate the overall accuracy, running time, cross validation
return f_f
```

4.1.2 System Training and Testing. We employ Sequential Minimal Optimization (SMO) [24], a supervised ML algorithm to offline train and test our system. The labeled dataset of the network traffic NSL is used for the training and testing. The NSL dataset contains five categories (classes), four represent these different attacks: DoS, Probe, Remote to Local (R2L), and User to Root (U2R), and one class represents

Table 1: Features in NSL Dataset

#	Feature Name	#	Feature Name
1	Duration	22	Is_guest_login
2	Protocol Type	23	Count
3	Service	24	Srv_count
4	Flag	25	Serror_rate
5	Src_byte	26	Srv_serror_rate
6	Dst_byte	27	Rerror_rate
7	Land	28	Srv_rerror_rate
8	Wrong_frangment	29	Same_srv_rate
9	Urgent	30	Diff_srv_rate
10	Hot	31	Srv_diff_host_rate
11	Num_failed_logins	32	Dst_host_count
12	Logged_in	33	Dst_host_srv_count
13	Num_compromised	34	Dst_host_same_srv_rate
14	Root_shell	35	Dst_host_diff_srv_rate
15	Su_attempted	36	Dst_host_same_src_port_rate
16	Num_root	37	Dst_host_srv_diff_host_rate
17	Num_file_creations	38	Dst_host_serror_rate
18	Num_shells	39	Dst_host_srv_serror_rate
19	Num_access_shells	40	Dst_host_rerror_rate
20	Num_outbound_cmds	41	Dst_host_srv_rerror_rate
21	Is_hot_login		

normal traffic. The NSL dataset consists of several training and testing files, thus, during the training phase, we use the 20 percent training file of 25192 samples to train our model. To test our model we use a separate testing file that not seen during the training. To online evaluate our trained model, which has been deployed as REST API on the top of the SDN controller as a web service, we use a different NSL file.

4.2 Misbehavior Attack Detection

Making the SDN controller inspecting all packets in the same flow will result in heavy computation and communication overhead which can lead to introducing DoS attack and making the SDN controller itself a bottleneck. Thus, legitimate traffic will suffer from heavy delay. Most of the existing flow analysis and packet content analysis based mechanisms usually inspect first packet of each new flow and once that flow has a rule assigned, any incoming traffic that match this flow mostly will not be inspected again and only matched to the installed flow rules. This logic has a serious limitation. For example, if the attacker exploits this point by sending normal traffic in the first packets of the flow and later sending malicious contents in the following packets, then he/she can perform the DDoS attack easily. Our proposed typical DDoS attack detection module has the same limitation as other mechanisms have, which is reasonable to avoid making the controller a bottleneck itself when inspecting every single packet. Therefore, our proposed misbehavior attack detection module aims to overcome this limitation. This module only inspects packets that belong to known flow and have already considered as normal traffic. For example, flow assigned rules with value of FORWARD.

Usually, during a DDoS attack, the number of outgoing packets is greater than the number of incoming packets which can lead to higher ratio of TCP traffic than usual. Since it is a TCP based detection technique it is not applicable to other protocols. In regard to ICMP and UDP which usually utilize small amount of bandwidth. However, sudden change in the amount of transferred ICMP or UDP byte/sec are good indication of attacks. This change increases the packet rate to formidable rate [25]. Considering that the normal traffic usually has same incoming and outgoing traffic volume, while in some types of the DDoS flooding attacks the volume of incoming and outgoing traffic must have huge difference impact on the traffic ratio and increase amount of bandwidth [3, 4]. Therefore, DDoS flooding attacks such as ICMP, UDP, TCP, can be noticed by considering the variance of traffic volume and amount of bandwidth of network traffic from/to source/destination.

To detect misbehavior attack, we apply statistical DDoS detection methods in time intervals, where we collect number of samples of traffic from each IP source address in small amount of time, e.g., one second (samples/sec). The samples of traffic for each IP address is represented as: Samples (IP) = $\{S1, S2, \ldots, Sn\}$ where n denotes number of samples, where sample is the set of all incoming packets per second. We compute the mean and the standard deviation (SD) as follows:

$$\mu = \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}. \tag{2}$$

Where x_i is the number of packets in each collected i^{th} sample, and $\overline{x}$ is the average of all collected samples of packets. The value of n is equal to the total number of collected samples. We easily collect these statistic such as x_i through the SDN enabled switches by activating the counter of the received packets per flow entry. The way how we treat the collected samples is similar to the sampling method presented in [25]. However, with SDN this task becomes much easier since such statistics are supported by OpenFlow counters at SDN-enabled switches. Thus, these statistics can be easily queried by the controller. Moreover, we also consider that the attacker may change his behavior at unpredictable and uncertain moment. Thus, unlike the work in [25], our sampling method is not limiting the number of samples to collect.

As result of collecting the samples of each source IP address and calculating their corresponding mean and standard deviation, we consider the greatest of mean as attack. According to [25], if $normal\_rate < S1 < S2 < \ldots < Sn$, then there might be increasing rate DDoS attack, while if $normal\_rate < S1 = S2 = \ldots = Sn$, then there might be constant rate DDoS attack. To avoid false positive when legitimate traffic are considered malicious traffic, we add another stage of verification through our reputation system in our Reference Module engine (RefM) which is described later in this section.

4.3 NewFlow Attack Detection

We assume that knowledgeable attackers can penetrate a protected system to understand the protection methods being deployed. For instance, when an attacker finds out that

Algorithm 4: Sampling Method

```
Input  : Packets p, set of IP addresses have rules in OVS S
Output : FlowTable, AttackerIPaddress
Procedure samplingMethod()
    while 1 do
        foreach IP ∈ S do
            check incoming flow
        if IP has rule "FORWARD" then
            for f_i in flow do
                N = 3 //Number of samples
            for n_i in N do
                collect number of packets
            M: Calculate the mean
            SD: Calculate the standard deviation
```

a protected system is deploying typical DDoS attack detection method that inspect traffic flow and/or packet content. Then, he/she starts a new type of attack via injecting the network with large numbers of new flows that are unmatched to any flow entries in the switches. The way how the attacker launches this *NewFlow_Attack* cannot be detected by the typical DDoS mechanisms since new flow packets can be designed to be normal flows. Therefore, the switches will keep generating *packet_in* messages to the controller for instructions. As consequence, both the switches and controller will suffer from processing attack traffic which will consume their resources in a short time [26]. Many believe that with suitable flow rule timeout value, flow rules entries are removed before overflowing the switch's flow table. However, this is not usually true, as we believe that if an attacker knows the rule timeout, for example, by using such an existing tool called SDN scanner [27], he/she can send its first packet of a new flow to trigger the process of flow rule installation. Later, it sends packets in intervals of less than the timeout. Thus, the corresponding rule will be fired up and then will not be removed by the rule idle or hard timeout mechanism.

In addition, some works to deal with DoS designed against SDN-based infrastructure [26], base their solutions on measuring only real-time rate of *packet_in* messages. This can be effective if we believe the rate of new flow corresponding to SDN-DoS attack can be as expected higher than normal new flow rate. However, the attacker can launch this attack by using as many as possible compromised machines (zombies) where each can generate new flow at low and slowly rate. Therefore, only measuring the rate of *packet_in* messages can not be enough to detect *new flow attack* [26].

Our proposed solution for this attack contains multiple points that when are combined can achieve better *new flow attack* detection rate: *a*) Historical Data; *b*) *packet_in* rate; *c*) Flow Reputation. As mentioned above, all incoming packets that belong to existing flows will be matched to the installed flow rules. Therefore, unlike packets that belong to new flows, these packets will not trigger *packet_in* messages in most cases. However, the amount of processed packets (matched packets, and received packets) belonging to the same flow will be higher when compared to packets that belong to new flows [19]. We design RefM engine on the controller that contain *flow reputation* and *historical data* units. This RefM is a comprehensive module that can record received packets of the same flow during the flow duration (in seconds). Also, it obtains information about the flows of each switch, and records the historical data of each flow to classify the flows and their corresponding data into normal or malicious. It collects data from all security modules to update the reputation unit. When any attack module reports malicious flow, then the necessary information of that flow will be logged into the RefM as well as the reputation value either will be increased or decreased.

Traffic with good reputation can be analyzed to extract normal flow signature. This can help us to identify a threshold that we can use to measure normal and malicious flow which can contribute to set the baseline for allowed *packet_in* request rate of each switch. Also, based on the reputation we assign a priority for processing corresponding traffic. On each switch we design a Switch Load Monitor Unit (SLMU) which is used to monitor the switch current status. On the controller, we deploy a SLMU module that gather status information of each connected switch to finds, for example, which switch is under attack. The SLMU module queries all connected switches to gather necessary statistical information. Most importantly, the number of active entries, their corresponding matched packets, and duration, in seconds, of each existing flow entry. Moreover, the SLMU module monitors *packet_in* and *flow_removed* messages. The controller then can inspect the frequency of received messages from each switch using the following equation:

$$I.M_{s_i} = \frac{I.M^t_{S_i}}{time} \tag{3}$$

$$R.M_{s_i} = \frac{R.M^t_{S_i}}{time} \tag{4}$$

$$M.P_{s_i} = \frac{M.P^t_{S_i}}{time} \tag{5}$$

$I.M_{s_i}$, $R.M_{s_i}$, and $M.P_{s_i}$ represent the total *packet_in* messages, *flow_removed* messages, and number of *matched_packets* of switch s_i, respectively. Whereas, t is a set of time slots that have equal length. Therefore, based on the collected statistics of each switch s_i, we mostly compare the following values: First, we compare the values of $I.M_{s_i}$ among switches with respect to a time slot t. Thus, if the values of received *packet_in* from one switch is higher than others, then that indicates this switch is suspicious and might be under new flow attack. Second, compare the values of $I.M_{s_i}$ within suspicious switch over time interval to find and confirm whether there is abnormal burst of *packet_in* messages. Further inspection is done based on the collected $R.M_{s_i}$, and $M.P_{s_i}$. Therefore, if the value of matched packets $M.P_{s_i}$ is smaller than the value of received *flow_removed* messages $R.M_{s_i}$ then this indicates switch s_i is under new

flow attack. Measuring the value of $M.P_{s_i}$ of each flow from each switch along with $I.M_{s_i}$ and $R.M_{s_i}$ can help to detect the new flow attack even it is launched at slow rate.

The SLMU model communicates with the RefM module to update and gather useful historical data about the legitimate and suspicious flows in the system. This leads to the ability of detecting NewFlow attack, and which switch is under attack. Moreover, the controller can decide how to redirect the traffic of a switch to another. For instance, if switch A flow table is full and then the controller needs to redirect switch A's traffic to another switch, it can install the rules to other switches immediately. However, we notice that this can lead to security policy and rules conflict. In our work [28] this issue has been already addressed. Therefore, we deploy that solution in our current system to avoid such an issue.

Algorithm 5: NewFlow Attack Detection

```
Input  : I.M_{s_i}, R.M_{s_i}, and M.P_{s_i}
Output : NewFlow Attack
Procedure NewFlowAttackDetect()
    while 1 do
        foreach s_i ∈ C do
            \\for every switch connected to the controller
            Collect flow statistic I.M_{s_i}, R.M_{s_i}, and
              M.P_{s_i}
        for s_i ∈ S do
            \\Collect I.M_{s_i} and Find MAX
            if Max(I.M_{s_i}) = True then
                Att_Switch = Max(I.M)
            \\switch might be under attack
            call Function F(Att_Switch)
            if M.P_{s_i} < R.M_{s_i} then
                switch under attack
    Function F(Att_Switch)
    foreach t_i ∈ Att _Switch do
        \\for every time slot in under attack switch
          confirm abnormal burst of I.M_{Att_Switch}
        if Max(I.M_{t_i}) = True then
            switch under attack
```

4.4 System Logical Zones

The flow table at the controller logically partitioned into three zones. For example, *Zone one* "Whitelist" with highest priority. This zone will have specific information about the sources, and the services they want to connect to at every tenant. This information is provided by the tenant. *Zone two* "Blacklist" with lower priority. This zone will have flow rules that address traffic to vulnerable services on the tenant from ANY source. *Zone three* "general" with lowest priority. This zone will be a generic permit from ANY source to a tenant address. Thus, any service not deemed vulnerable and placed in the blacklist can be accessed by this rule.

This concept helps the controller to easily look up the source of *packet_in* and its predefined actions that can be immediately pushed to the corresponding switch. As mentioned above that, our security modules contribute to the source's reputation method in the RefM module. Therefore, based on the source's reputation we update the logical system zones by including the necessary source' information with respect to the proper zone configuration.

In effect, this would deflect the new flow attack from the controller to the tenant. Anomaly based traffic detection running at the controller can determine when the traffic destined to a reputable tenant is above threshold and then take a countermeasure.

4.5 Attack Mitigation

This process starts after our system has detected an attack in the network. The REST application compiles rule-set based on recommendation from any detection module, and adds rules corresponding to attack traffic, normal traffic and unknown traffic in Flow Rule Generator application. The rules are pushed to the SDN controller via PUSH based REST API. The SDN controller interacts with switches using Openflow protocol. The flow table of Openflow switch are updated based on flow rules provided by the SDN controller. Most of the current attack mitigation approaches block the source of the attack. However, in this work, we do not block the source of the detected attack, but the malicious and suspicious traffics will be forwarded to the honeypot model that is deployed on a separate VM machine and it is responsible for deeply learning and analyzing the attackers' behaviors. This helps wasting the attackers time and resources and allow us to study the new attacks and update our training model for future defending.

5 EXPERIMENTAL RESULTS

In this Section, we have divided the evaluation of our work into three major parts: *a*) Evaluation of a ML system which involves features selection, training, and classification. *b*) Evaluation of our overall secure system to detect and mitigate typical DDoS when it is deployed online in real SDN environment using ML system. *c*) Evaluation of our deployed mechanisms for *misbehavior attack, and new flow attack.*

5.1 Experimental Setup

We run our experiment into software environment using Mininet [29] emulation environment which creates virtual hosts connected to OVS switch. The network is controlled by POX, a Python-based SDN controller [30]. The security modules are implemented as applications upon the controller. We connect four switches to a single remote controller where each switch connects several virtual hosts to the network. Some of these hosts generate benign traffic and others generate attack traffic. To evaluate our ML security module, we use an existing NSL testing file that have our predefined features and replay this traffic on interfaces that our OVS

will send their traffic to the controller for instruction. The speed of traffic replay can be chosen during experimental analysis based on requirement.

5.2 Experimental Result

5.2.1 Feature Selection and Model Training. In this part, we cover five categories where four are representing attack traffic and one represents normal traffic. The attack categories are: DoS, Probe, R2L, and U2R. Each has multiple types of attacks. In order to validate our model, during the training process not all these attacks were included. We left some attacks from each category to classify them as unknown, so our system could detect normal and abnormal traffic. To perform this part Weka 3.8.0 was used. The details of these attacks can be refereed to [20], and it was skipped due to space constraints. Table 2 indicates each feature selection algorithm against its candidates features that are required to construct our feature selection algorithm. Table 3 shows the best detection rate in regarding to the features selected based on each feature selection algorithm. We evaluate all best selected subset features using three classification algorithms SVM, J48, and Naïve Bayes. Genetic and greedy algorithms both are used in conjunction with correlation-based feature selector (CFS) which ranks features subsets according to a correlation heuristic evaluation function [31]. The ranker algorithm is used in conjunction with InfoGain selector which evaluates the worth of an attribute by measuring the information gain with respect to the class.

Table 2: Feature selection algorithm against its candidates features

Feature Selection Alg.	Best Selected Subset Features	Counts
Genetic	3, 4, 5, 6, 7, 9, 12, 24, 25, 26, 27, 29, 30, 32, 34, 39	16
Ranker	3, 4, 5, 6, 12, 23, 25, 26, 28, 29, 30, 31, 33, 34, 35, 36, 38, 39	17
Greedy	3, 4, 5, 6, 12, 14, 26, 29, 30, 37, 38	11
Our Alg. (BestSelectedSubsetFeatures f_f)	3, 4, 5, 6, 7, 9, 12, 14, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39	25

5.3 DDoS Attack Prediction and Mitigation

In this part, we deployed our trained model as a web service on the SDN controller, and we evaluate our success in DDoS attack prediction and mitigation. Figure 3 depicts the amount of traffic in Mbps at the honeypot and the SDN controller before and after redirecting malicious and suspicious traffics to the honeypot. As can be observed from the figure, the SDN controller receives the packets destined for target host with huge traffic burst, approximately 484 Mbps, which is classified as DDoS type of traffic by our classification algorithm, while honeypot only receives normal amount of traffic, approximately 0.024 Mbps. The SDN controller identifies the traffic as malicious and as new packets arrive with pattern similar to DDoS attack, they are redirected to honeypot. Since there is huge volume of traffic that is directed at target host, the SDN controller takes about 60 secs to redirect all the malicious traffic to honeypot where it can be inspected further to analyze source of attack, and other relevant header information. After about 60s, we can observe that honeypot receives traffic, approximately 415.6 Mbps. Most of the traffic flagged as DDoS attack (with 97% accuracy) based on detection algorithm used is directed to honeypot, and only small part of traffic within limit of error, approximately 13 Mbps reaches the target host.

Table 3: Best detection rate in regarding to the features selected based on each feature selection algorithm

Number of Features	SMO Classification Detection Accuracy	J48 Classification Detection Accuracy	Naive Bayes Classification Detection Accuracy
All 41 features	97.31	98.76	95.11
Genetic Alg. + CfsSubsetAttributeEval	96.85	98.33	82.62
Ranker Alg.+ InfoGainAttributeEval	97.14	97.77	82.03
Greedy Alg. + CfsSubsetAttributeEval	95.71	92.74	83.92
Our Alg. (BestSelectedSubsetFeatures f_f)	99.40	99.75	95.87

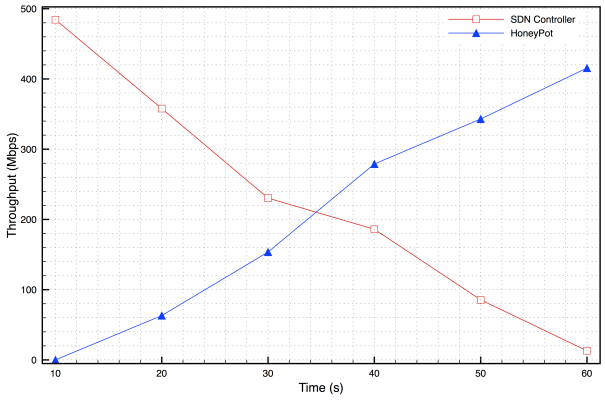

Figure 3: Traffic burst at SDN controller and Honeypot vs time.

We define some terminologies that will be used in quality measure: *precision*, *recall*, and *f-measure*, and the details of these measures can be refereed to [32], and it was skipped due to space limitation. We can see from Figure 4 that DoS attack has F_1 measure 0.933 and normal traffic has F_1 measure value 0.96, showing us that the quality of algorithm is high enough to make reasonable prediction of DoS attacks, Probe traffic, R2L, and normal traffic.

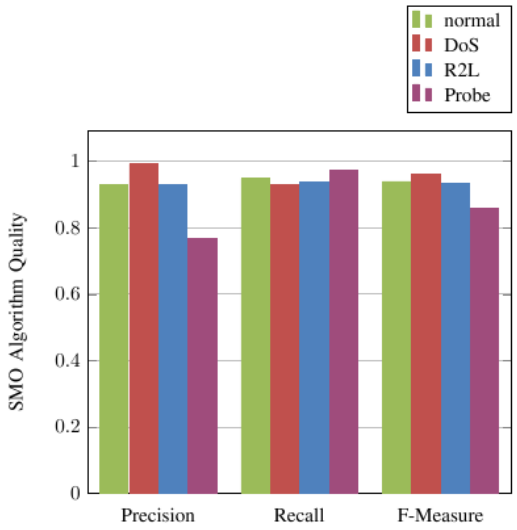

Figure 4: Precision, Recall, F-measure for our classification algorithm.

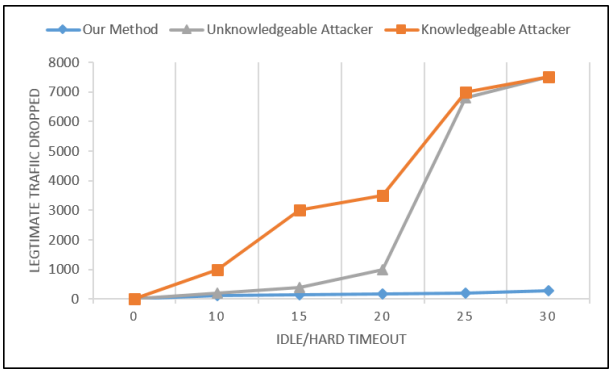

Figure 5: Dropped legitimate traffic.

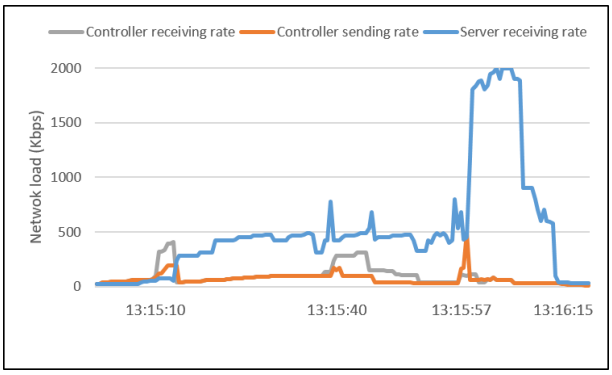

Figure 6: Misbehavior attack over time.

5.4 Misbehavior and New Flow Attack Evaluation

To show how an attacker can bypass the short configured idle/hard timeout, we configured different timeouts to remove flow entries when the timeout is excessed. We configured the malicious clients to resend two packets within each idle timeout where the first packet triggers the creation of new flow rule and the second packet fires up the flow rule before the timeout excessed to avoid flow entries removing. This part shows the violation of idle timeout strategies that applied by some existing defense systems to prevent occupying flow table by malicious clients. Figure 5 shows the total amount of dropped legitimate packets when the system under the attack. It shows both results when the attacker is knowledgeable and not of the small amount of timeout deployment. It can be noticed that when the amount of hard timeout is decreased, the number of dropped legitimate traffic also decreased. However, this is not valid when we assume that the attacker knows the deployed defense mechanism and start sending lifesaver packet to avoid its flow from being removed. This figure also shows how our *NewFlow attack* detection mechanism can prevent this kind of attack and noticeably reduce the number of dropped packet toward small amount.

Figure 6 shows the experiment we performed to simulate the misbehavior attack to show its feasibility and that SDNs are vulnerable to such attacks. At the beginning, the traffic caused the generation of *packet_in* messages from 13:15:10 to 13:15:40. As it can be seen from the figure that during this time the controller sends back *flow_mod* messages to the switch which slightly increase the network load rate. Since the flows were considered legitimate flows, generated by normal clients, then all will be forwarded to the server. Now normal clients can change their behavior and start flooding the network server (DDoS attack). It can be seen that at 13: 15: 57, the receiving rate rapidly increased at the server side. However, when our misbehavior attack detection module is deployed, it takes around 18 second to detect and mitigate this type of attack.

6 CONCLUSION AND FUTURE WORK

In this paper, an adaptive and intelligent secure system has been proposed to detect and mitigate the DDoS attack based on the integrating of SDN and ML. Both, when combine, can provide automated and intelligent network control. We developed a feature selection algorithm to select an effective number of features from the NSL which has provided better accuracy for attack detection. We also introduced the two serious attacks that are rarely considered in literature; *NewFlow attack* and *Misbehavior attack*. Our experimental results show that we can limit attacker capacity to very small amount, while keeping services up and running for the legitimate users. This is in particular useful when considering cost-benefit analysis for attack mitigation. Our experimental results show that we can limit attacker capacity to very small amount, while keeping services up and running for the legitimate users. This is in particular useful when considering

cost-benefit analysis for attack mitigation. Our future work will cover the analysis of our unsupervised ML model which will deeply inspect all unknown traffic.

ACKNOWLEDGMENT

This research is supported by NSF Secure and Resilient Networking (SRN) Project (1528099) and NATO Science for Peace & Security Multi-Year Project (MD.SFPP 984425). S. Pisharody is supported by a scholarship from the NSF CyberCorps program (NSF-SFS-1129561).

REFERENCES

[1] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[2] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.

[3] Prasanta Gogoi, DK Bhattacharyya, Bhogeswar Borah, and Jugal K Kalita. A survey of outlier detection methods in network anomaly identification. *The Computer Journal*, page 26, 2011.

[4] Monowar H Bhuyan, Hirak Jyoti Kashyap, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Detecting distributed denial of service attacks: methods, tools and future directions. *The Computer Journal*, page 31, 2013.

[5] Qiao Yan, F Richard Yu, Qingxiang Gong, and Jianqiang 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*, 18(1):602–622, 2016.

[6] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069, 2013.

[7] Damian Jankowski and Marek Amanowicz. Intrusion detection in software defined networks with self-organized maps. *Journal of Telecommunications and Information Technology*, 2015.

[8] Bing Wang, Yao Zheng, Wenjing Lou, and Y Thomas Hou. Ddos attack protection in the era of cloud computing and software-defined networking. *Computer Networks*, 81:308–319, 2015.

[9] Seyed Mohammad Mousavi and Marc St-Hilaire. Early detection of ddos attacks against sdn controllers. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 77–81. IEEE, 2015.

[10] Jun Li, Skyler Berg, Mingwei Zhang, Peter Reiher, and Tao Wei. Drawbridge: software-defined ddos-resistant traffic engineering. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 591–592. ACM, 2014.

[11] Ali Hussein, Imad H Elhajj, Ali Chehab, and Ayman Kayssi. Sdn security plane: An architecture for resilient security services. In *Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on*, pages 54–59. IEEE, 2016.

[12] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[13] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[14] Kostas Giotis, Christos Argyropoulos, Georgios Androulidakis, Dimitrios Kalogeras, and Vasilis Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62:122–136, 2014.

[15] Javed Ashraf and Seemab Latif. Handling intrusion and ddos attacks in software defined networks using machine learning techniques. In *Software Engineering Conference (NSEC), 2014 National*, pages 55–60. IEEE, 2014.

[16] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*, 2015.

[17] Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *Usenix Security*, pages 817–832, 2015.

[18] Min Suk Kang, Virgil D Gligor, and Vyas Sekar. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. *NDSS'16*, 2016.

[19] OpenFlow Switch Specification. Version 1.3. 2 (wire protocol 0x04), 2013.

[20] Nsl-kdd data set for network-based intrusion detection systems. available on: http://nsl.cs.unb.ca/nsl-kdd/, march 2009.

[21] Zhongyu Wei, Wei Gao, Tarek El-Ganainy, Walid Magdy, and Kam-Fai Wong. Ranking model selection and fusion for effective microblog search. In *Proceedings of the first international workshop on Social media retrieval and analysis*, pages 21–26. ACM, 2014.

[22] Witold Bednorz. Advances in greedy algorithms. *Wienna: I-Tech Education and Publishing KG*, 2008.

[23] Gareth Jones. Genetic and evolutionary algorithms. *Encyclopedia of Computational Chemistry*, 2:1127–1136, 1998.

[24] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[25] J Udhayan and T Hamsapriya. Statistical segregation method to minimize the false detections during ddos attacks. *IJ Network Security*, 13(3):152–160, 2011.

[26] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: a dos attack prevention extension in software-defined networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 239–250. IEEE, 2015.

[27] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166. ACM, 2013.

[28] Sandeep Pisharody, Ankur Chowdhary, and Dijiang Huang. Security policy checking in distributed sdn based clouds. In *Communications and Network Security (CNS), 2016 IEEE Conference on*, pages 19–27. IEEE, 2016.

[29] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[30] Murphy McCauley. Pox wiki. *openflow, https://openflow. stanford. edu/display/ONL/POX+ Wiki# POXWiki-FAQs*, 2014.

[31] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[32] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.