EAI Endorsed Transactions
on Security and Safety

A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)

Quamar Niyaz, Weiqing Sun, Ahmad Y. Javaid*

College of Engineering, The University of Toledo, 2801 W. Bancroft St., Toledo, Ohio-43607, USA

Abstract

Distributed Denial of Service (DDoS) is one of the most prevalent attacks that an organizational network infrastructure comes across nowadays. Poor network management, low-priced Internet subscriptions, and readily available attack tools can be attributed to their rise. The recently emerged software-defined networking (SDN) and deep learning (DL) concepts promise to revolutionize their respective domains. SDN keeps the global view of the entire managed the network from a single point, i.e., the *controller*, thus making the network management easier. DL-based approaches improve feature extraction/reduction from a high-dimensional dataset such as network traffic headers. This work proposes a deep learning based multi-vector DDoS detection system in an SDN environment. The detection system is implemented as a network application on top of the SDN controller and can monitor the managed network traffic. Performance evaluation is based on different metrics by applying the system on traffic traces collected from different scenarios. A high accuracy with low false-positive rate is observed in attack detection for the proposed system.

Received on 27 November 2016; accepted on 2 July 2017; published on 28 December 2017

Keywords: Network security, Deep Learning, Multi-vector DDoS detection, Software Defined Networking

doi:10.4108/eai.28-12-2017.153515

1. Introduction

Distributed denial of service (DDoS) attacks results in unavailability of network services by continuously flooding its servers with undesirable traffic. Low-price Internet subscriptions and readily available attack tools led to a vast increase in volume, size, and complexity of these attacks in the recent past. According to the forecast of Cisco Visual Networking Index (VNI) [1], DDoS incidents will reach up to 17 million in 2020, a threefold increment compared to 2015. The nature of attacks has also changed to being multi-vector rather than having a single type of flooding. A study reported that 64% attacks until mid-2016 were multi-vectors that include TCP SYN floods and DNS/NTP amplification combined together [2]. Adversaries or hacktivists use DDoS attacks for extortion, revenge, misguided marketing, and online protest. Many financial, public sector, media, and social entertainment sites are recent victims [3–5] and suffered from monetary and reputation damages. Therefore, detection and mitigation of these attacks in real-time have become a prime concern for large organizations. However, detection of DDoS attacks is a challenging task nowadays due to the following reasons: i) it is difficult to distinguish the attack traffic from the legitimate traffic as the attackers craft the packet headers to make them appear legitimate, ii) a voluminous traffic due to *flash-crowd* event is not a DDoS attack, therefore a detection system should be able to distinguish it from the attack. iii) to distinguish a flash-crowd attack from an actual flash-crowd [6].

Recently, both software-defined networking (SDN) and deep learning (DL) have found several useful and interesting applications in the industry as well as the research community. SDN provides centralized management, global view of the entire network, and programmable control plane; makes network devices flexible for different applications. These features of SDN offer better network monitoring and enhanced the security of the managed network compared to traditional networks [7, 8]. On the other hand, DL based approaches outperformed existing machine learning techniques when applied to various classification

*Corresponding author. ahmad.javaid@utoledo.edu problems. They improve feature extraction/reduction from a high-dimensional dataset in an unsupervised manner by inheriting the non-linearity of neural networks [9]. Researchers have also started to apply DL for the implementation of various intrusion detection systems and observed desirable results discussed in Section 2. In this work, we implement a DDoS detection system that incorporates stacked autoencoder (SAE) based DL approach in an SDN environment and evaluate its performance on a dataset that consists of normal Internet traffic and various DDoS attacks. The motivation behind using DL is to overcome the above-mentioned challenges in DDoS detection. The DL model attempts to reduce a large set of features into an optimal feature set by self-learning and enhances the detection rate.

The organization of this paper is as follows. Section 2 discusses related work on DDoS detection in an SDN environment and use of DL for network intrusion detection. Section 3 gives an overview of SDN and SAE. In Section 4, we discuss the architecture of our proposed system. Section 5 presents experimental set-up and performance evaluation of the system. Finally, Section 6 concludes the paper with future work directions.

2. Related Work

Although a lot of research has been performed in the implementation of network intrusion detection system using machine learning techniques [10], we discuss the related work from two perspectives. First, where DL has been used for network intrusion detection and second where DDoS detection is addressed in an SDN environment.

2.1. Intrusion detection using DL

In [11], Mostafa et al. used deep belief network (DBN) based on restricted Boltzmann machine (RBM) for feature reduction with support vector machine (SVM) as a classifier to implement a network intrusion detection system (NIDS) on NSL-KDD [12] intrusion dataset. NSL-KDD is a benchmark network intrusion dataset to evaluate the performance of an NIDS and eliminates some of the issues observed in KDD Cup-99 dataset [13]. In [14], Ugo et al. used discriminative RBM (DRBM) to develop a semi-supervised learning based network anomaly detection and evaluated its performance in an environment where network traffic for training and test scenarios were different. They used real-world traffic traces and KDD Cup-99 [13] intrusion dataset in their implementation. In [15], Gao et al. used RBM based DBN with a neural network as a classifier to implement an NIDS on KDD-Cup 99 dataset. In [16], Kang et al. proposed an NIDS for the security of in-vehicular networks using DBN and improved detection accuracy compared to previous approaches. In [17], we implemented a deep learning based NIDS using NSL-KDD dataset. We employed self-taught learning [18] that uses sparse autoencoder instead of RBM for feature reduction and evaluated our model separately on training and test datasets. In [19], Ma et al. proposed a system that combines spectral clustering (SC) and sparse autoencoder based deep neural network (DNN). They used KDD-Cup99, NSL-KDD, and a sensor network dataset to evaluate the performance of their model.

2.2. DDoS detection in SDN environment

In [20], Braga et al. proposed a light-weight DDoS detection system using self-organized map (SOM) in SDN. Their implementation uses features extracted from flow-table statistics collected at a certain interval to make the system light-weight. However, it has limitation in handling traffic that does not have any flow rules installed. In [21], Giotis et al. combined an OpenFlow (OF) and sFlow for anomaly detection to reduce processing overhead in native OF statistics collection. As the implementation was based on flow sampling using sFlow, false-positive was quite high in attack detection. In [22], Lim et al. proposed a DDoS blocking application (DBA) using SDN to efficiently block legitimately looking DDoS attacks. The system works in collaboration with the targeted servers for attack detection. The prototype was demonstrated to detect HTTP flooding attack. In [23], Mousavi et al. proposed a system to detect DDoS attacks in the controller using entropy calculation. Their implementation depends on a threshold value for entropy to detect attacks which they select after performing several experiments. The approach may not be reliable since threshold value will vary in different scenarios. In [24], Wang et al. proposed an entropy based light-weight DDoS detection system by exporting the flow statistics process to switches. Although the approach reduces the overhead of flow statistics collection in the controller, it attempts to bring back the intelligence in network devices. Another recent work [25] used DL for intrusion detection in SDN environment. However, it used NSL-KDD dataset with only six features for the DL model development and provided an insight in the integration of the model within an SDN environment, without an actual implementation.

Time-series based detection is another approach towards this direction. Some of the works used this approach instead of machine learning where detection takes place by observing changes in traffic parameters for consecutive time windows. However, this approach may lead to increased false-positive rate as a small change or fluctuation may trigger alarms[26, 27].

In contrast to the discussed work, we use SAE based DL model to detect multi-vector DDoS attacks in SDN. We use a set of a large number of features extracted from network packet headers and then use DL to reduce this set in an unsupervised manner. We apply our model to traffic dataset collected in different environments. The proposed system attempts to detect attacks on both the SDN control plane and the data plane and is implemented completely on the SDN controller.

3. Background Overview

We discuss SDN and SAE before describing our DDoS detection system.

3.1. Software-Defined Networking (SDN)

As discussed earlier, the SDN architecture decouples the control plane and data plane from network devices, also termed as '*switches*', and makes them simple packet forwarding elements. The decoupling of control logic and its unification to a centralized controller offers several advantages compared to the current network architecture that integrates both the planes tightly. Administrators can implement policies from a single point, i.e. controller, and observe their effects on the entire network that makes management simple, less error-prone, and enhances security. Switches become generic and vendor-agnostic. Applications that run inside a controller can program these switches for different purposes such as layer 2/3 switch, firewall, IDS, load balancer using API offered by a controller to them [29].

Figure 1a shows the SDN architecture with its different planes and applications. Switches, end hosts, and communication between them form the data plane. The controller is either a single server or a group of logically centralized distributed servers. The controller can run on commodity hardware and communicates with switches using standard APIs called southbound interfaces. One of the *de facto* standards for southbound interfaces is OpenFlow (OF) protocol [30]. The controller servers communicate with each other using east-westbound interfaces. Network applications communicate with the controller using northbound interfaces.

The controller and switches exchange various types of messages using OF protocol over either a TLS/SSL encrypted or open channel. These messages set-up switch connection with the controller, inquire network status or manage traffic flows in the network. Switches have flow tables for flow rules that contain match-fields, counters, and actions to handle traffic flows in the network. SDN defines *flow* as a group of network packets that have same values for certain packet header fields. The controller installs flow rules for traffic flows based on the policies dictated by the network applications.

Flow rule installation takes place in switches either in reactive mode or proactive mode. The reactive mode works as follows. When a packet enters a switch, it looks up for a flow rule inside its flow tables that matches with the packet headers. If a rule exists for the packet, the switch takes an action that may involve packet forwarding, drop or header modification. If a table-miss happens, i.e., there are no flow rules for an incoming flow, the switch sends a *packet_in* message to the controller that encapsulates packet headers for the incoming flow. The controller extracts packet headers from the received message and sends a *packet_out* or *flow_mod* message to switches for the received packet's flow. The controller installs flow rules inside switches using *flow_mod* messages and switches perform actions on subsequent packets of the installed flow, without forwarding them to the controller. Figure 1b demonstrates the reactive mode set-up in SDN. Flow rules may expire after a certain time to manage the limited memory size of switches. The controller does not install rules in switches. Instead, it instructs them to forward the packet from a single or multiple port(s) using *packet_out* messages. In contrast, the controller pre-installs flow rules into switches in proactive mode.

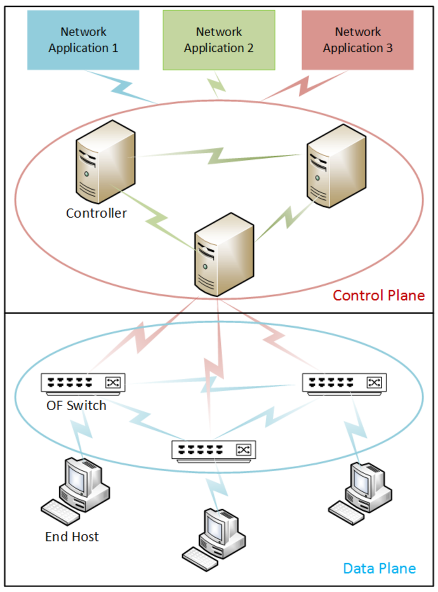
3.2. Stacked Autoencoder (SAE)

Stacked Autoencoder (SAE) is a DL approach that consists of stacked sparse autoencoders and soft-max classifier for unsupervised feature learning and classification, respectively. We discuss sparse autoencoder before SAE. A sparse autoencoder is a neural network that consists of three layers in which the input and output layers contain M nodes, and the hidden layer contains N nodes. The M nodes at the input represent a record with M features, i.e., $X = \{x_1, x_2, ..., x_m\}$. For the training purpose, the output layer is made an identity function of the input layer, i.e., $\hat{X} = X$ shown in Figure 2a. The sparse autoencoder network finds optimal values of weight matrices, $U \in \Re^{N\times M}$ and $U' \in \Re^{M\times N}$, and bias vectors, $b_1 \in \Re^{N\times 1}$ and $b_1{}' \in \Re^{M\times 1}$ while trying to learn an approximation of the identity function, i.e. $\hat{X} \approx X$ using back-propagation algorithm [31]. Many different functions are used for activation of hidden and output nodes, we use Sigmoid function, $g(z) = \frac{1}{1+e^{-z}}$, for the activation of g_{U,b_1} shown in Eqn. 1:

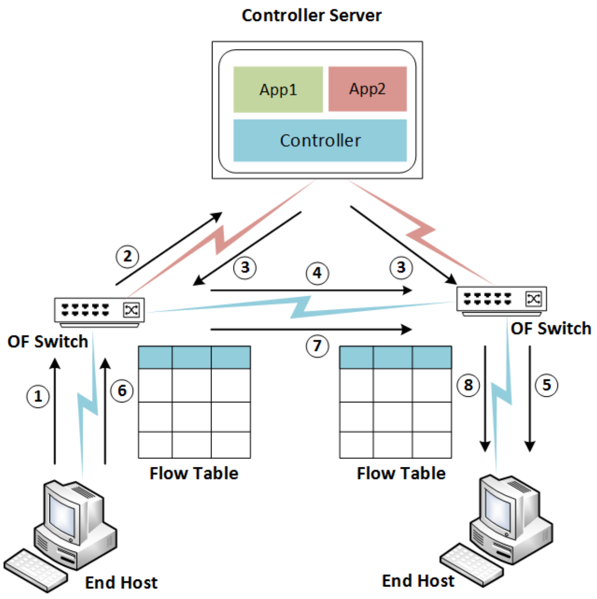
$$g_{U,b_1}(X) = g(UX + b_1) = \frac{1}{1 + e^{-(UX+b_1)}} \tag{1}$$

$$J = \frac{1}{2r}\sum_{i=1}^{r}\|X_i - \hat{X}_i\|^2 + \frac{\lambda}{2}\Big(\sum_{n,m} U^2 + \sum_{m,n} U'^2 + \sum_{n} b_1{}^2 + \sum_{n} b_1{}'^2\Big) + \beta\sum_{j=1}^{N} KL(\rho\|\hat{\rho}_j) \tag{2}$$

Eqn. 2 represents the cost function for optimal weight learning in sparse autoencoder. It is minimized using back-propagation. The first term in the RHS represents an average of sum-of-square errors for all the input



(**a**) Different planes and network applications in SDN



(**b**) Reactive traffic flow set-up in SDN [28]

Figure 1. An SDN architecture and basic traffic flow in SDN

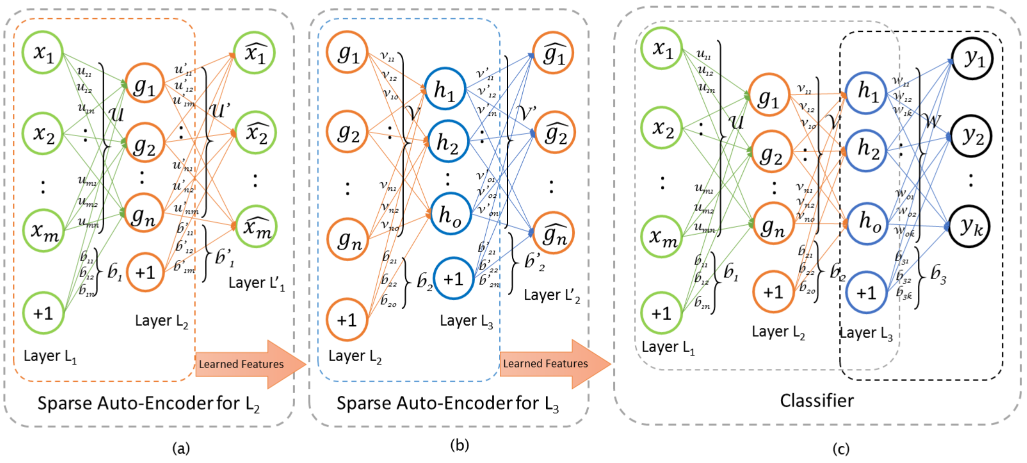


Figure 2. A stacked autoencoder based deep learning model

values and their corresponding output values for all r records in the dataset. The second term is a weight decay term with λ as the decay parameter to avoid over-fitting. The last term is a sparsity penalty term that puts a constraint on the hidden layer to maintain low average activation values and expressed using Kullback-Leibler (KL) divergence shown in Eqn. 3:

$$KL(\rho \| \hat{\rho}_j) = \rho log \frac{\rho}{\hat{\rho}_j} + (1-\rho) log \frac{1-\rho}{1-\hat{\rho}_j} \qquad (3)$$

where $\rho \in \{0, 1\}$ is a sparsity constraint parameter and β controls the sparsity penalty term. The $KL(\rho \| \hat{\rho}_j)$ becomes minimum when $\rho = \hat{\rho}_j$, where $\hat{\rho}_j$ is the average activation value of a hidden unit j over all the training inputs.

Multiple sparse autoencoders are stacked with each other in a way that the outputs of each layer is fed into the inputs of the next layer to create an SAE. Greedy-wise training is used to obtain optimal values of the weight matrices and bias vectors for each layer. For illustration, the first layer, g, on raw input x is trained to obtain U, U', b_1, $b_1{}'$. The layer g encodes the raw input, X, using U and b_1. Then, the encoded values are used as inputs to train the second layer to obtain parameters V, V', b_2, $b_2{}'$ shown in Figure 2b. This process goes further until the last hidden layer is trained. The output of last hidden layer is fed into a classifier. Finally, all layers of SAE are treated as a single model and fine-tuned to improve the performance of the model shown in Figure 2c.

4. Implementation of DDoS Detection System

In an SDN, attacks can occur either on the data plane or control plane. Attacks on the former are similar to traditional attacks and affect a few hosts. However, attacks on the latter attempt to bring down the entire network. In this second kind of attack, adversaries fingerprint an SDN for flow installation rules and then send new traffic flows, resulting in flow table-misses in the switch [32]. This phenomenon forces the controller to handle every packet and install new flow rules in switches that consume system resources on the controller and switches. In our previous work [28], we empirically evaluated the impact of SDN adversarial attacks on network services. In the current work, we implement a DDoS detection system as a network application in SDN to handle attacks for both cases.

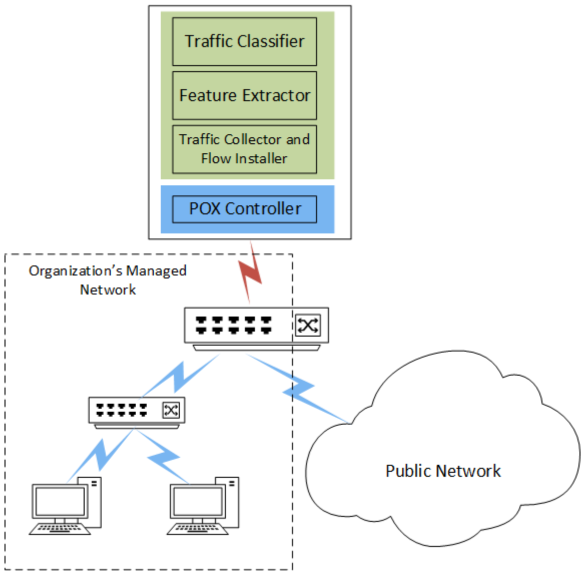

Traffic Classifier
Feature Extractor
Traffic Collector and Flow Installer
POX Controller
Organization's Managed Network
Public Network

Figure 3. A DDoS detection system implemented in SDN

The detection system consists of three modules as shown in Figure 3: i) Traffic Collector and Flow installer (**TCFI**), ii) Feature Extractor (**FE**), and iii) Traffic Classifier (**TC**). It should be emphasized here that to minimize false-positives, our system relies on every packet for flow computation and attack detection instead of sampling flows using some tools such as *sFlow*.

4.1. Traffic Collector and Flow Installer (TCFI)

The TCFI module runs concurrently with the FE and TC modules which are triggered using a *timer* function. It examines OF message type for an incoming packet at

TCP		**UDP**	**ICMP**
Src IP	Window	Src IP	Src IP
Dst IP	SYN	Dst IP	Dst IP
Src Port	ACK	Src Port	ICMP Type
Dst Port	URG	Dst Port	ICMP Code
Protocol	FIN	Protocol	Protocol
Data Size	RST	Data Size	Data Size
TTL	PUSH	TTL	TTL

Table 1. Different headers extracted from TCP, UDP, and ICMP packets

Algorithm 1: TCFI Module

```
Data: Incoming network packets at the controller
Result: List of extracted packet headers for TCP,
        UDP, and ICMP
begin
    $packets\_list \longleftarrow \emptyset$
    $flows\_list \longleftarrow \emptyset$
    while $Timer$ for the FE is not triggered do
        Receive a packet from switch
        Store headers in $packets\_list$
        if Packet arrives due to flow table miss then
            Compute $flow$ for the packet
            Compute symmetric flow, $symflow$, for
              $flow$
            if $symflow \in flows\_list$ then
                Remove $symflow$ from $flows\_list$
                Install flow rule for $symflow$ in
                  switch(es)
                Install flow rule for $flow$ in
                  switch(es)
            else if $flow \notin flows\_list$ then
                Add $flow$ in $flows\_list$
                Output the packet to desired port
            else
                Output the packet to desired port
```

the controller. A message type determines the reason for a packet's arrival which is either due to a flow table-miss or an installed flow rule that forwards a packet towards the controller and desired physical ports. The TCFI extracts various header fields from a packet to identify its *flow*. A *flow* in TCP or UDP traffic is a group of packets having same values for protocol type, source, and destination IP addresses, and source and destination port numbers. An ICMP *flow* has similar header fields, except for port numbers it has ICMP message type and code. The TCFI extracts few more header fields from a packet that help in features extraction from flows. It stores all of these extracted headers in a list for every packet coming to the controller. Table 1 shows the headers for TCP, UDP, and ICMP traffic that the TCFI extracts. It performs this task when a packet arrives due to pre-installed flow rules.

However, when a packet arrives due to a flow table-miss, it performs following tasks in addition to the one mentioned above. It looks up a symmetric flow corresponding to the packet's flow in the flow list. Two flows are *symmetric* for TCP or UDP traffic if the source IP address and port number of one flow are similar to the destination IP and port number of the other, and vice-versa. For ICMP traffic, two flows are *symmetric* if they are request and response types. If a symmetric flow exists for an incoming flow, then it installs forwarding rules for both of them in SDN switches and removes the symmetric one from the list. The rules include an action that forwards packets to desired physical ports and the controller for the incoming and its symmetric flows. The reason for installing rules only for symmetric flows is built on the assumption that attackers, in general, spoof their IP addresses to prevent responses towards them from victims. Therefore, the TCFI installs flow rules for legitimate traffic and avoids any flow table saturation attacks in switches. If it does not find any symmetric flow for an incoming packet, it looks up whether a flow already exists in the list for the same. If a flow exists, it forwards the packet from switches without installing any rules. Otherwise, it adds the packet's flow in the list and then forwards it. Algorithm 1 shows various steps involved in the TCFI. Although the algorithm appears similar to maximum entropy detector in [33], i) it considers flow in general instead of flags based ii) a packet arrives at the controller either due to a table-miss or a forwarding rule towards the controller. iii) it stores packet headers for each packet arrives at the controller.

#	Feature Description
1	# of incoming TCP flows
2	Ratio of TCP flows over total incoming flows
3	# of outgoing TCP flows
4	Ratio of TCP flows over total outgoing flows
5	Ratio of symmetric incoming TCP flows
6	Ratio of asymmetric incoming TCP flows
7	# of distinct src IP for incoming TCP flows
8	Entropy of src IP for incoming TCP flows
9	Bytes per incoming TCP flow
10	Bytes per outgoing TCP flow
11	# of packets per incoming TCP flow
12	# of packets per outgoing TCP flow
13	# of distinct window size for incoming TCP flows
14	Entropy of window size for incoming TCP flows
15	# of distinct TTL values for incoming TCP flows
16	Entropy of TTL values for incoming TCP flows
17	# of distinct src ports for incoming TCP flows
18	Entropy of src port for incoming TCP flows
19	# of distinct dst ports for incoming TCP flows
20	Entropy of dst ports for incoming TCP flows
21	Ratio of dst ports $\leq$ 1024 for incoming TCP flows
22	Ratio of dst port $>$ 1024 for incoming TCP flows
23	Ratio of TCP incoming flows with SYN flag set
24	Ratio of TCP outgoing flows with SYN flag set
25	Ratio of TCP incoming flows with ACK flag set
26	Ratio of TCP outgoing flows with ACK flag set
27	Ratio of TCP incoming flows with URG flag set
28	Ratio of TCP outgoing flows with URG flag set
29	Ratio of TCP incoming flows with FIN flag set
30	Ratio of TCP outgoing flows with FIN flag set
31	Ratio of TCP incoming flows with RST flag set
32	Ratio of TCP outgoing flows with RST flag set
33	Ratio of TCP incoming flows with PUSH flag set
34	Ratio of TCP outgoing flows with PUSH flag set

Table 2. Features extracted for TCP flows

4.2. Feature Extractor (FE) and Traffic Classifier (TC)

The detection system triggers the FE module using a *timer* function. The FE takes packet headers from the packets list populated by the TCFI and extracts features from them for a set interval and resets the packet list to store headers for the next interval. Table 2, 3, and 4 show the list of 68 features that the FE extracts for TCP (34), UDP (20), and ICMP (14) flows, respectively. We derived this feature set after detailed literature survey and use SAE to reduce it. The FE computes these features for all hosts in a network which has incoming traffic flows for that particular interval. Although we perform computations on all packets in the network, we extract features by grouping them in flows. The FE computes median for a number of bytes and packets per flow in feature # 9-12, 43-46, and 63-67. It computes the entropy, $H(F)$, for feature # 8, 14, 16, 18, 20, 42, 48, 50, 54, 62, and 68 which is defined as follows:

$$H(F) = -\sum_{i=1}^{n} \frac{f_i}{\sum_{j=1}^{n} f_j} \times log_2 \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{4}$$

where set F=$\{f_1, f_2, ..., f_n\}$ denotes the frequency of each distinct value. Once the FE extracts these features, it invokes the TC module implemented using SAE. It classifies traffic in one of the eight classes which includes one normal and seven types of DDoS attack classes based on TCP, UDP or ICMP vectors that adversaries launch either separately or in combinations.

5. Experimental Set-up, Results, and Discussion

To evaluate our system, we collected network traffic from a real network and a private network testbed. We discuss them along with the performance evaluation

#	Feature Description
35	# of incoming UDP flows
36	Ratio of UDP flows over total incoming flows
37	# of outgoing UDP flows
38	Ratio of UDP flows over total outgoing flows
39	Ratio of symmetric incoming UDP flows
40	Ratio of asymmetric incoming UDP flows
41	# of distinct src IP for incoming UDP flows
42	Entropy of src IP for incoming UDP flows
43	Bytes per incoming UDP flow
44	Bytes per outgoing UDP flow
45	# of packets per incoming UDP flow
46	# of packets per outgoing UDP flow
47	# of distinct src ports for incoming UDP flows
48	Entropy of src ports for incoming UDP flows
49	# of distinct dst ports for incoming UDP flows
50	Entropy of dst ports for incoming UDP flows
51	Ratio of dst port ≤ 1024 for incoming UDP flows
52	Ratio of dst port > 1024 for incoming UDP flows
53	# of distinct TTL values for incoming UDP flows
54	Entropy of TTL values for incoming UDP flows

Table 3. Features extracted for UDP flows

	Normal		Normal with Attack	
	Incoming	Outgoing	Incoming	Outgoing
TCP	2501685	6219236	2501685	24581489
UDP	1204789	2395943	1204852	23494744
ICMP	834	1147	834	26309690

Table 5. Number of incoming and outgoing packets for different protocols in normal and normal along with attack traffic

Traffic class		**# of records**	
		Training	Test
Normal (N)		49179	21076
Attack	TCP (T)	5471	2344
	UDP (U)	5273	2260
	ICMP (I)	1602	686
	TCP & UDP (TU)	4694	2011
	TCP & ICMP (TI)	4739	2031
	UDP & ICMP (UI)	4437	1902
	All (A)	5615	2407

Table 6. Number of records in the training and test datasets for normal and different attack traffic

#	Feature Description
55	# of incoming ICMP flows
56	Ratio of ICMP flows over total incoming flows
57	# of outgoing ICMP flows
58	Ratio of ICMP flows over total outgoing flows
59	Ratio of symmetric incoming ICMP flows
60	# of asymmetric incoming ICMP flows
61	# of distinct src IP for incoming ICMP flows
62	Entropy of src IP for incoming ICMP flows
63	Bytes per incoming ICMP flow
64	Bytes per outgoing ICMP flow
65	# of packets per incoming ICMP flow
66	# of packets per outgoing ICMP flow
67	# of distinct TTL values for incoming ICMP flows
68	Entropy of TTL values for incoming ICMP flows

Table 4. Features extracted for ICMP flows

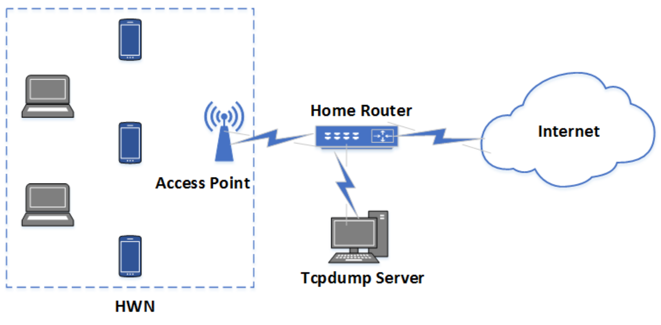


Figure 4. Home Wireless Network (HWN) for normal traffic collection

results. It is known that the KDD Cup99 dataset contains several types of attacks including the DDoS and our system being DDoS focused, this dataset was not chosen for system evaluation. Similarly, NSL-KDD was also not selected as it is an improved version of the KDD Cup99 dataset still having various attacks including DDoS. In addition, both of these datasets have a different feature set compared to our derived set of features.

5.1. Experimental Set-up

We used a home wireless network (HWN) connected to the Internet for normal traffic collection. The HWN comprised of around 12 network users connected with the Internet using their laptops and smart-phones shown in Figure 4. These users were not uniformly active for all the time which led to variation in the traffic intensity. We saved HWN traffic of three days in a Linux system using `tcpdump` [34] and port mirroring at a Wi-Fi access point. The traffic of first two days were used as normal flows. The traffic of the third day was mixed with the attack data that we collected separately and it was labeled as an attack. The motivation behind doing this was to model scenarios where hosts/networks may have attack traffic in presence of normal traffic, thus making the detection task challenging for an IDS. To mix traffic, we used bit-twist [35] that modifies packet headers in a traffic trace file. We replaced some IP addresses with those of found in the attack traffic trace files. Finally, we replayed the normal and attack traffic traces together described later in this section. Table 5 shows the traffic mix for normal traffic along with the attack traffic. The collected traffic comprises data from web browsing, audio/video streaming, real-time messengers, and online gaming. To collect attack traffic, we created a private network in a segregated laboratory environment using VMWare ESXi host. The private network consists of 10 DDoS attacker and 5 victim hosts. We used `hping3` [36] to launch different kinds of DDoS attacks with different packet frequencies and sizes. We launched one class of attack at a time so that it can be labeled easily while extracting features. After traffic collection in trace files, we created an SDN testbed on the same ESXi host similar to [37] that consists of an SDN controller, an OF switch, and a network host using Ubuntu Linux systems. We used `POX` [38], a Python based controller, with our DDoS detection application running on it in the controller system and installed OpenvSwitch [39] in the switch to use it as an OF switch. In the host system, we used `tcpreplay` [40] to replay traffic traces for normal and attack traffic one at a time. We saved features computed by the FE for each interval in dataset files for the training of TC module. We set the interval 60s in the *timer* function to trigger the FE module for feature extraction. We referred [20, 21, 41] to set the interval. Braga et al. [20] used 3s for polling interval, we found it too frequent for polling and invoking the FE. While Giotis et al. and Jin et al. used interval of 30s and 20s, respectively. We concluded that there was no standard polling interval used in the literature and therefore, selected a polling interval of 60s. We divided the dataset files into training and test datasets. Table 6 shows the distribution of records in the dataset. Traffic features in the datasets are real-valued positive numbers. We normalize them using $max - min$ $normalization$ shown in Eqn. 5, before passing them to the TC module.

$$X_{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \ \forall x_i \in X \tag{5}$$

$$x_{min} = \text{smallest value in } X$$

$$x_{max} = \text{largest value in } X$$

5.2. Results

We evaluated the performance of our system on the datasets specified in Table 6 using parameters including *accuracy*, *precision*, *recall*, *f-measure*, *ROC*. We use confusion matrix to calculate precision, recall, and f-measure. A confusion matrix, M, is an $N \times N$ matrix where N is the number of classes. Each column in the matrix represents prediction for a particular class as all possible classes including the correct class (itself). Each row represents prediction of all possible classes as a particular class including that class. The diagonal elements of the matrix represent the true-positive (TP) for each class, sum of the matrix elements along a row excluding the diagonal element represents the number of false-positive (FP) for a class corresponding to that row, sum of the matrix elements along a column excluding the diagonal element represents the number of false-negative (FN) for a class corresponding to that column. Following are definitions of various performance parameters:

- *Accuracy (A)*: percentage of accurately classified records in a dataset

$$A = \frac{Accurately\ classified\ records}{Total\ records} \times 100 \tag{6}$$

- *Precision (P)*: number of accurately predicted records over all predicted records for a particular class. Using the confusion matrix, M, precision for each class, j, can be defined as follows:

$$\begin{aligned} P_j &= \frac{TP_j}{TP_j + FP_j} \times 100 \\ &= \frac{M_{j,j}}{M_{j,j} + \sum_{\substack{i=1 \\ i \neq j}}^{N} M_{j,i}} \times 100 \end{aligned} \tag{7}$$

- *Recall (R)*: number of accurately predicted records over all the records available for a particular class in the dataset. Using the confusion matrix, M, recall for each class, j, can be defined as follows:

$$\begin{aligned} R_j &= \frac{TP_j}{TP_j + FN_j} \times 100 \\ &= \frac{M_{j,j}}{M_{j,j} + \sum_{\substack{i=1 \\ i \neq j}}^{N} M_{i,j}} \times 100 \end{aligned} \tag{8}$$

- *F-measure (F)*: It uses precision and recall for the holistic evaluation of a model and is represented as the harmonic mean of them. For each class, j, it is defined as follows:

$$F_j = \frac{2 \times P_j \times R_j}{P_j + R_j} \times 100 \tag{9}$$

- *Receiver Operating Curve (ROC)*: It helps in visualizing a classifier's performance by plotting the true-positive rate against false-positive rate of the classifier. The area under the ROC gives an estimate of an average performance of the classifier. Higher the area, greater is the performance.

We used the training dataset to develop the SAE classification model for the TC module and test dataset for performance evaluation. First, we developed the model for 8-class traffic classification including normal and seven kinds of DDoS attack that occur in combination with TCP, UDP, and ICMP based traffic. To make a better comparison, we also developed separate attack detection models with soft-max and neural network (NN) which are building blocks of SAE. As observed from Table 7, the SAE model achieved better performance compared to the soft-max and neural network model in terms of accuracy. We computed precision, recall, f-measure for each traffic class. Figure 6 shows their values which are derived

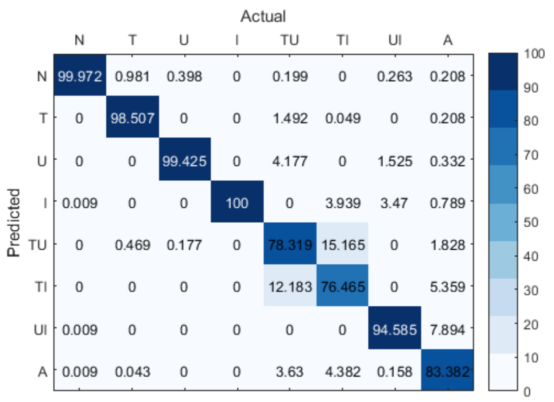


Figure 5. Confusion matrix for 8-class classification in the SAE model

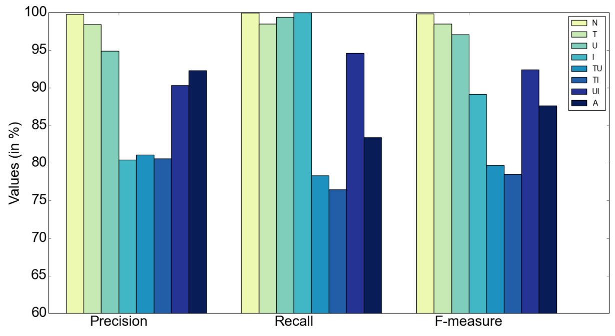


Figure 6. Precision, recall, and f-measure for 8-class

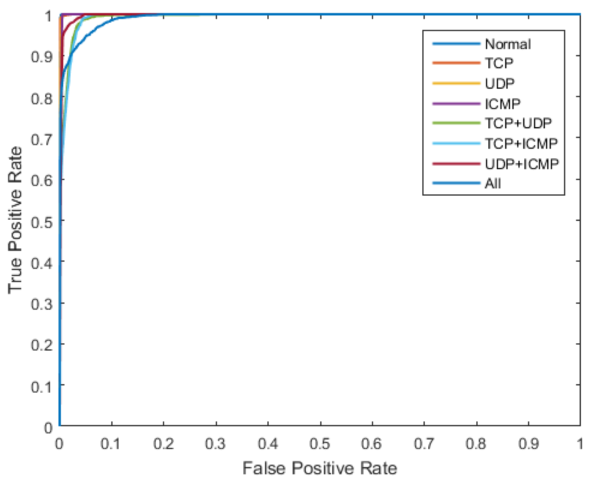


Figure 7. ROC curve for 8-class classification

from the confusion matrix shown in Figure 5. As seen from the figure, the model has f-measure value above 90% for normal, TCP, UDP, and UDP with ICMP attacks traffic. It has comparatively low values of f-measure for TCP with ICMP and TCP with UDP attacks due to their classification of other kinds of attacks as observed from

Method	Accuracy (in %)
Soft-max	94.30
Neural Network	95.23
SAE	**95.65**

Table 7. Classification accuracy comparison among soft-max, neural network, and SAE based models

Classification Models	Accuracy	False-positive Rate
8-class	95.65%	0.5%
2-class	99.82%	0.3%

Table 8. Accuracy and False-positive Rate for 8-class and 2-class SAE models

the Figure 5. However, it is observed from the same figure that the fraction of their classification as normal traffic is less than 0.2. Figure 7 shows the ROC curve for 8 different classes. From the figure, we observe that the true positive rate is above 90% with a false-positive rate of below 5% for all kinds of traffic that results in the area under the ROC curve close to unity.

We evaluated our model for 2-class classification by considering all kinds of DDoS attacks as a single attack class to make a comparison with other works. Due to the unique nature of this work involving deep learning based attack detection in an SDN, and unavailability of existing literature in this specific domain, it was difficult to compare our work with other works. Figure 9 shows the performance for 2-class classification. The model achieved detection accuracy of 99.82% with f-measure values as 99.85% and 99.75% for normal and attack classes, respectively, derived from the confusion matrix shown in Figure 8. On the contrary, the two closely related works [20] and [23], achieved a detection accuracy of 99.11% in data plane and 96% in control plane, respectively. It should be noted that both of these works did not address attack detection in the other plane. Table 8 shows the accuracy and false-positive rate for 2-class and 8-class SAE models. The accuracy and false-positive rate achieved for 2-class model exceeded some of the multi-agent based computational intelligence approaches for intrusion detection listed in [10].

We also measured the computational time for training and classification in our model using a machine with Intel (R) Core i7 CPU @ 3.40 GHz processor and 16 GB RAM running Matlab 2016a on Windows 7. Table 9 shows computational time for the training of 81,010 records and classification of 34,717 records specified in Table 6.

Training time	Classification time
524s	.0835s

Table 9. Average computational time for the training and classification in the SAE model

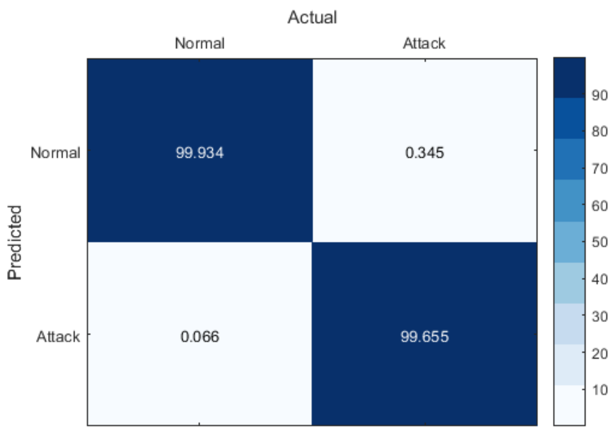


Figure 8. Confusion matrix for 2-class classification

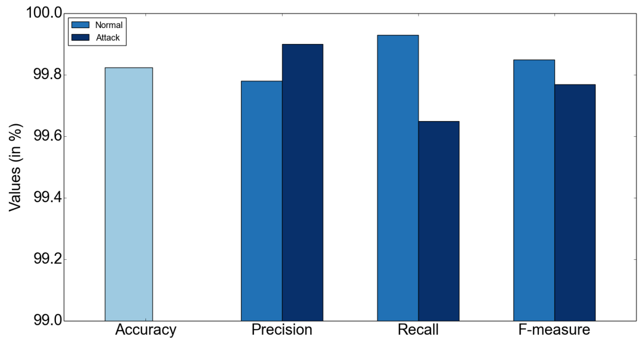


Figure 9. Accuracy, precision, recall, and f-measure for 2-class classification

5.3. Discussion

With our DDoS detection system, we identify individual DDoS attack class and also determine whether an incoming traffic is normal or attack. A clear advantage in identifying each attack traffic type separately is enabling the mitigation technique to block only a specific type of traffic causing the attack, instead of all kinds of traffic coming towards the victim(s). Although we implemented a detection system, we separately extracted features for each host which has incoming traffic for an interval. Therefore, we can identify the hosts with normal traffic and the ones with attack traffic. Accordingly, the controller can install flow rules inside the switches to block the traffic for a particular host if it undergoes an attack.

Our proposed system has a few limitations in terms of processing capabilities. The TCFI and FE modules collect every packet to extract features and are implemented on the controller for low false-positive in detection. However, this approach may limit the controller's performance in large networks. We can overcome it by adopting a hybrid approach that can either use flow sampling or individual packet capturing based on the observed traffic in the organizational network. Another approach that could be employed to handle DDoS attacks in the data plane is to deploy the TCFI and FE modules in another host, send all packets to it instead of the controller for features processing, and then periodically notify the controller with extracted features for the TC module. To reduce the time in feature extraction, we can also apply distributed processing similar to our another previous work [42].

6. Conclusion

In this work, we implemented a deep learning based DDoS detection system for multi-vector attack detection in an SDN environment. The proposed system identifies individual DDoS attack class with an accuracy of 95.65%. It classifies the traffic in normal and attack classes with an accuracy of 99.82% with very low false-positive compared to other works. In the future, we aim to reduce the controller's bottleneck and implement an NIDS that can detect different kinds of network attacks in addition to DDoS attacks such as network scanning, malware propagation, application layer attacks. We can use the ensemble-based approach in distributed settings to efficiently detect various attacks similar to the approaches discussed in [43–46]. We also plan to use deep learning for feature extraction from raw bytes of packet headers instead of feature reduction from the derived features in future NIDS implementation.

References

[1] Cisco Visual Networking Index Predicts Near-Tripling of IP Traffic by 2020. https://newsroom.cisco.com/press-release-content?articleId=1771211 Accessed April 8, 2016.

[2] Verisign Q2 2016 DDoS Trends: Layer 7 DDoS Attacks a Grwoing Trend. https://blog.verisign.com/security/verisign-q2-2016-ddos-trends-layer-7-ddos-attacks-a-growing-trend/ Accessed April 8, 2017.

[3] The Recent DDoS Attacks on Banks: 7 Key Lessons. https://www.neustar.biz/resources/whitepapers/recent-ddos-attacks-on-banks Accessed April 8, 2017.

[4] DDoS Attack on BBC May Have Been Biggest in History. http://www.csoonline.com/article/3020292/cyber-attacks-espionage/ddos-attack-on-bbc-may-have-been-biggest-in-history.html Accessed April 8, 2017.

[5] 'World Of Warcraft: Legion' Goes Down As Blizzard Servers Hit With DDoS. http://www.forbes.com/sites/

erikkain/2016/09/01/world-of-warcraft-legion-goes-down-as-blizzard-servers-hit-with-ddos/ Accessed April 8, 2017.

[6] Oikonomou, G. and Mirkovic, J. (2009) Modeling Human Behavior for Defense Against Flash-crowd Attacks. In *Proceedings of the 2009 IEEE International Conference on Communications*, ICC'09: 625–630.

[7] Schehlmann, L., Abt, S. and Baier, H. (2014) Blessing or Curse? Revisiting Security Aspects of Software-Defined Networking. In *10th International Conference on Network and Service Management (CNSM) and Workshop*: 382–387. doi:10.1109/CNSM.2014.7014199.

[8] Shin, S., Xu, L., Hong, S. and Gu, G. (2016) Enhancing Network Security through Software Defined Networking (SDN). In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*: 1–9.

[9] Schmidhuber, J. (2015) Deep Learning in Neural Networks: An Overview. *Neural Networks* **61**: 85 – 117.

[10] Shamshirband, S., Anuar, N.B., Kiah, M.L.M. and Patel, A. (2013) An Appraisal and Design of a Multi-Agent System based Co-operative Wireless Intrusion Detection Computational Intelligence Technique. *Engineering Applications of Artificial Intelligence* **26**(9): 2105 – 2127.

[11] Salama, M.A., Eid, H.F., Ramadan, R.A., Darwish, A. and Hassanien, A.E. (2011) Hybrid Intelligent Intrusion Detection Scheme. In *Soft Computing in Industrial Applications* (Berlin, Heidelberg: Springer Berlin Heidelberg): 293–303.

[12] Tavallaee, M., Bagheri, E., Lu, W. and Ghorbani, A. (2009) A Detailed Analysis of the KDD CUP 99 Data Set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*: 1–6. doi:10.1109/CISDA.2009.5356528.

[13] KDD Cup 99. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html Accessed April 8, 2017.

[14] Fiore, U., Palmieri, F., Castiglione, A. and De Santis, A. (2013) Network Anomaly Detection with the Restricted Boltzmann Machine. *Neurocomputing* **122**: 13 – 23.

[15] Gao, N., Gao, L., Gao, Q. and Wang, H. (2014) An Intrusion Detection Model Based on Deep Belief Networks. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*: 247–252.

[16] Kang, M.J. and Kang, J.W. (2016) Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. *PLoS ONE* **11**(6): 1–17.

[17] Javaid, A., Niyaz, Q., Sun, W. and Alam, M. (2016) A Deep Learning Approach for Network Intrusion Detection System. *EAI Endorsed Transactions on Security and Safety* **16**(9).

[18] Raina, R., Battle, A., Lee, H., Packer, B. and Ng, A.Y. (2007) Self-taught Learning: Transfer Learning from Unlabeled Data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07 (New York, NY, USA: ACM): 759–766.

[19] Ma, T., Wang, F., Cheng, J., Yu, Y. and Chen, X. (2016) A Hybrid Spectral Clustering and Deep Neural Network Ensemble Algorithm for Intrusion Detection in Sensor Networks. *Sensors* **16**(10): 1701. URL http://www.mdpi.com/1424-8220/16/10/1701.

[20] Braga, R., Mota, E. and Passito, A. (2010) Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10 (Washington, DC, USA: IEEE Computer Society): 408–415.

[21] Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D. and Maglaris, V. (2014) Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments. *Computer Networks* **62**: 122 – 136.

[22] Lim, S., Ha, J., Kim, H., Kim, Y. and Yang, S. (2014) A SDN-oriented DDoS Blocking Scheme for Botnet-based Attacks. In *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*: 63–68.

[23] Mousavi, S.M. and St-Hilaire, M. (2015) Early detection of DDoS attacks against SDN controllers. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*: 77–81.

[24] Wang, R., Jia, Z. and Ju, L. (2015) An Entropy-Based Distributed DDoS Detection Mechanism in Software-Defined Networking. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15 (Washington, DC, USA: IEEE Computer Society): 310–317.

[25] Tang, T.A., Mhamdi, L., McLernon, D., Zaidi, S.A.R. and Ghogho, M. (2016) Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In *The international conference on wireless networks and mobile communications*, WINCOM'16.

[26] Li, L. and Lee, G. (2005) Ddos attack detection and wavelets. *Telecommunication Systems* **28**(3): 435–451. doi:10.1007/s11235-004-5581-0, URL http://dx.doi.org/10.1007/s11235-004-5581-0.

[27] Shinde, P. and Guntupalli, S. (2007) Early DoS Attack Detection using Smoothened Time-Series andWavelet Analysis. In *Third International Symposium on Information Assurance and Security*: 215–220. doi:10.1109/IAS.2007.16.

[28] Niyaz, Q., Sun, W. and Alam, M. (2015) Impact on SDN Powered Network Services Under Adversarial Attacks. *Procedia Computer Science* **62**: 228 – 235.

[29] Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S. (2015) Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE* **103**(1): 14–76.

[30] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. *et al.* (2008) OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* **38**(2): 69–74.

[31] Ng, A. (2011) Sparse Autoencoder .

[32] Shin, S. and Gu, G. (2013) Attacking Software-defined Networks: A First Feasibility Study. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13 (New York, NY, USA: ACM): 165–166.

[33] Mehdi, S.A., Khalid, J. and Khayam, S.A. (2011) *Revisiting Traffic Anomaly Detection Using Software Defined Networking*, 161–180.

[34] Tcpdump. http://www.tcpdump.org Accessed April 8, 2017.

[35] Bit-Twist. http://bittwist.sourceforge.net/ Accessed 8 April, 2017.

[36] Hping3. http://wiki.hping.org Accessed April 8, 2017.

[37] Hartpence, B. (2015) The RIT SDN Testbed and GENI .

[38] POX Wiki: Open Networking Lab. https://openflow.stanford.edu/display/ONL/POX+Wiki Accessed April 8, 2017.

[39] Open vSwitch. http://www.openvswitch.org Accessed April 8, 2017.

[40] Tcpreplay. http://tcpreplay.synfin.net Accessed April 8, 2017.

[41] Jin, S. and Yeung, D.S. (2004) A Covariance Analysis Model for DDoS Attack Detection. In *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, **4**: 1882–1886 Vol.4. doi:10.1109/ICC.2004.1312847.

[42] Karimi, A.M., Niyaz, Q., Sun, W., Javaid, A.Y. and Devabhaktuni, V.K. (2016) Distributed Network Traffic Feature Extraction for a Real-time IDS. In *2016 IEEE International Conference on Electro Information Technology (EIT)*.

[43] Cruz, T., Rosa, L., Proença, J., Maglaras, L., Aubigny, M., Lev, L., Jiang, J. *et al.* (2016) A Cybersecurity Detection Framework for Supervisory Control and Data Acquisition Systems. *IEEE Transactions on Industrial Informatics* **12**(6): 2236–2246. doi:10.1109/TII.2016.2599841.

[44] Maglaras, L.A., Jiang, J. and Cruz, T.J. (2016) Combining ensemble methods and social network metrics for improving accuracy of {OCSVM} on intrusion detection in {SCADA} systems. *Journal of Information Security and Applications* **30**: 15 – 26. doi:http://doi.org/10.1016/j.jisa.2016.04.002, URL http://www.sciencedirect.com/science/article/pii/S2214212616300229.

[45] Kevric, J., Jukic, S. and Subasi, A. (2016) An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications* : 1–8doi:10.1007/s00521-016-2418-1, URL http://dx.doi.org/10.1007/s00521-016-2418-1.

[46] Ma, T., Wang, F., Cheng, J., Yu, Y. and Chen, X. (2016) A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. *Sensors* **16**(10). doi:10.3390/s16101701, URL http://www.mdpi.com/1424-8220/16/10/1701.