Performance of Botnet Detection by Neural Networks in Software-Defined Networks

Ivan Letteri, Massimo Del Rosso, Pasquale Caianiello, and Dajana Cassioli

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, via Vetoio snc, 67100 L'Aquila, Italy
ivan.letteri@graduate.univaq.it,massimo.delrosso@gmail.com,
pasquale.caianiello@univaq.it,dajana.cassioli@univaq.it

Abstract

The recent evolution of Internet to new paradigms such as network function virtualization and software defined networking poses new relevant challenges to the detection of Botnet attacks, calling for innovative approaches. In this work we propose a detection mechanism based on an Artificial Neural Net classifier trained by available data sets collected in conventional networks. We apply such detection mechanism to the timely use case scenario of a software defined network infected by the dangerous Botnet *Mirai*, circulating in October 2016. Experimental results show an accuracy of Botnet detection higher than 99%, thus outperforming available Botnet detection mechanisms currently used in conventional networks.

Keywords: Botnet, Malware, Software Defined Network, Mininet, Network Intrusion Detection and Prevention System, OpenDaylight, Machine Learning, Neural Network.

1 Introduction

Nowadays, botnets represent one of the most important and dangerous threats in Internet [10]. A Botnet consists of a set of machines compromised by a malware (referred to as bots), managed by a BotMaster. The main goal of a Botnet is to carry out malicious activities, executing the commands given by the "Command & Control" (C&C) server, managed by the Botmaster. A major challenge in current schemes for intrusion detection and/or prevention systems lies in the analysis of the time-varying behaviour of interactions between the *bot masters* and the *bots*, followed by relevance by the Botnet Deception and Obfuscation techniques.

The emerging paradigm of software defined networks (SDNs) represents a very promising framework for the implementation of effective botnet detection and/or prevention strategies. Actually, the decoupling of the control plane from the network switching infrastructure (data plane) provides a directly programmable and centralised logic to monitor and manage the network status, in particular for the acquisition of statistics relevant to security threats detection. Several intrusion prevention systems (IPSs) and intrusion detection systems (IDSs) have been proposed in the literature so far for SDNs.

A common approach is to extend the functionality of OpenFlow switches to include a programmable middlebox with a IDS and/or IPS [17]. A collaborative intrusion prevention architecture for SDNs is proposed in [2], where programmable switches of the network play the role of neurons in a neural network (NN) that leverages their parallel and mathematical manipulation. An *unsupervised* NN using the Kohonen algorithm trained to recognize unauthorized activities in a SDN managed from OpenDayLight (ODL) controller in combination with Network Function Virtualization (NFV) has been also proposed in [5]. It is based on a Self-Organized Map (SOM) learning method for the classification problem phase to build an IPS for DDoS attacks mitigation. Schehlmann, et al. [13] introduces a botnet detector and mitigation framework for large-scale networks using the NetFlow technology.

The fundamental problem in NN-based botnet detection is the definition of suitable discriminators[1] to filter *malicious* traffic against *regular* traffic in the network. The comprehensive analysis of network

[1]Discriminators are referred to as *features* in Machine Learning jargon.

traffic presented in [9] determined two hundred discriminators as the set of the most significant ones. A botnet classification strategy based on the centralised collection of network flow counters in SDNs, and the use of a supervised C4.5 decision tree classification algorithm, is proposed in [15]. Their experiments show that OpenFlow counters represent suitable features to distinguish bot-malware patterns among general network traffic flows. Network-based intrusion detection and prevention systems relying on the SDN ODL controller to evaluate performance against Denial of Service (DoS) is proposed in [8], based on a C4.5 decision tree model built on the Darpa 99 dataset, and a pre-processing phase to store network packet via packet sniffers.

In our work we propose a simplified technique which is proved to provide high accuracy in the detection of Botnets in SDNs. The presented results demonstrate that it is possible to detect malicious traffic effectively by analyzing only the OpenFlow 1.3 messages exchanged by network nodes and the controller. Our approach relies on the traffic classification by an artificial neural network (ANN) to identify Botnets in a SDN as playground. Such ANN is trained with a data training set (TS) obtained by a Botnet attack running in conventional networks (CN), i.e. non SDN. The paper is organized as follows. Sec. 2 introduces the SDN architecture, and methods for choosing the appropriate features to train the ANN for Botnet detection. Sec. 2.3 presents our proposed machine learning technique. Sec. 3 introduces the case study of the Botnet Mirai, emulated on our virtual network, and summarizes the results obtained in the experimentations, showing the performance of our ANN approach in detecting Mirai traffic and discussing its limitations. Finally, conclusions and possible future work are presented in Sec. 4.

2 Botnet Detection Technique

2.1 System Architecture

The proposed approach relies on the passive monitoring of network traffic in SDNs, resulting in a more lightweight approach than Deep Packet Inspection (DPI) methods. The schematic of the system architecture is shown in the principle diagram in Fig. 1.

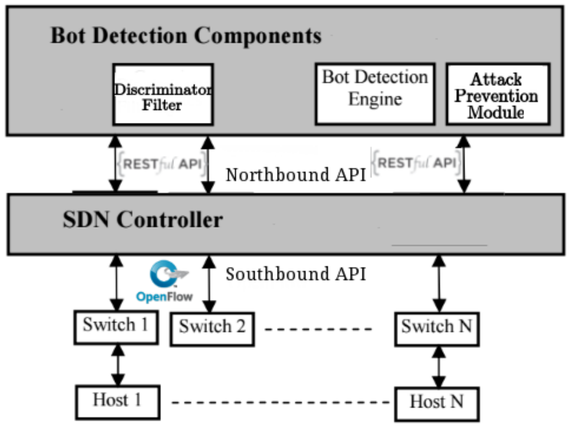

Figure 1: Principle diagram of the system architecture.

A SDN allows a single controller to orchestrate an entire network of switches. The SDN architecture is split in three layers: the lowest layer (*Data Layer*) is composed by hardware and virtual switches and is interfaced to the upper layer (*Control Layer*) by the Southbound APIs; the Control Layer (indicated as *SDN Controller* in Fig. 1) defines the entire network performance by monitoring and configuring the network entities; the top layer is the *Application Layer*, interfaced to the controller by the Northbound

APIs, that allows different applications to be developed on top of the control layer, including relevant network functionalities such as network management, security, middlebox and so on.

The communications between the switches in the data plane and the controller are regulated by the *OpenFlow protocol*, which provides the rules and the formats of messages (Southbound API). An OpenFlow-compatible forwarding device in data plane consists of three parts: i) a *flow table* that stores the action(s) to be applied to process the flow(s), one entry per each flow; ii) a *secure channel* for the message exchange with the controller; iii) the *OpenFlow protocol*.

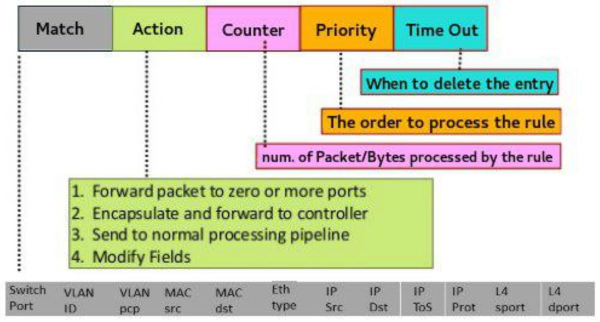
Match
Action
Counter
Priority
Time Out
When to delete the entry
The order to process the rule
num. of Packet/Bytes processed by the rule
1. Forward packet to zero or more ports
2. Encapsulate and forward to controller
3. Send to normal processing pipeline
4. Modify Fields
Switch Port
VLAN ID
VLAN pcp
MAC src
MAC dst
Eth type
IP Src
IP Dst
IP ToS
IP Prot
L4 sport
L4 dport

Figure 2: OpenFlow: Anatomy of a Flow Table Entry.

An entry in the *flow table* has five fields, as shown in Fig. 2: i) the *match*, with information about the packet header that defines the flow; ii) the *action*, which defines how the packets should be handled; iii) the *counter*, which keeps track of the amount of packets per flow; iv) the *priority*, which defines the order to process the rule, convenient to get statistics efficiently; v) the *time-out*, where the idle timeout and hard timeout control establishes the removal time of a flow entry from the OpenFlow table.

The components of the Botnet Detection technique proposed in this paper are shown in Fig. 1 and reside in an application running on top of the SDN controller, which uses the Northbound RESTful APIs to interact with the controller.

The *Attack Prevention Module* (APM) in Fig. 1 is responsible of creating an appropriate rule in the field *action* of the Flow Table (see the Fig. 2) of the virtual switches whenever a flow is detected by the *Bot Detection Engine* to be malicious and involved in a Botnet attack. Such rule blocks the infected host that is identified by its mac address. The installation phase of the blocking rule for a flow considered malicious proceeds in two steps: *i)* the mac-address of the infected host is recovered from the flow ID; *ii)* the APM communicates to the controller, through the Northbound API, that the *drop* rule has to be installed in the flow entry corresponding to the involved virtual switch in the detected malicious traffic. The Bot Detection Engine uses a Machine Learning (ML) technique to classify the traffic. In ML a system acquires knowledge through experience and is able to distinguish the specific patterns or anomalies for which it has been trained. This one is specifically trained to detect the presence of one or more components of a botnet in the network. The detection is typically based on the evaluation of appropriate *discriminators*, referred to as *features* in the ML field, which are application-specific. In the following, we describe the most common discriminators used for conventional data networks and SDNs, which are the ones we used in our experiments.

2.2 Discriminators Filter

The discriminator filter extracts a set of relevant parameters, referred to as *discriminators*, that are appropriately processed and given as *input features* to the ANN, core of the BotNet Detection Engine.

We selected a set of relevant features according to the definition of bidirectional flow within the use of time windows [1, 7, 18, 16]. The selected features are listed in Table 1 for CN and in Table 2 for the

SDNs, where also the conventional features **Data Rate** R and **Number of packets** N_p are employed as common practice.

Feature	**Relevance**
First packet size (Bytes)	It reveals typical characteristics of network protocol and refers to the entire flow
Number of small packets, i.e. with size lower than 320 Bytes, in the time window	Its use is widely known in P2P Botnet and in Bot – C&C server communications
Number of packets	Some Botnets try to keep the connection open by sending a large number of packets in a specific time window
Average payload size, i.e. the ratio between the size and the number of payloads in the time window	Typically, in legitimate traffic flow this value is higher than the one of the Botnet traffic flow
Data Rate, i.e. the ratio between the number of bytes exchanged within the time window and the time window size	Typically, normal traffic generates higher values
Variance of payloads size in the time window	Useful for finding DoS attacks or to detect evasion techniques employed in communication between Bots and C&C consisting in sending fake packets
Ratio between in-packets and out-packets	Bots are not managed by humans and are programmed to respond to a set of commands they receive.

Table 1: Selected discriminators as input to the ANN core of the BotNet Detection Engine for conventional networks[1, 7, 16, 18].

Feature	**Description**
Protocol	Flow type at transport layer (UDP or TCP)
Bytes Sum	Amount of flow bytes in the time window
Average packet size	$\overline{S}_\mathrm{pk} = \sum S_\mathrm{pk}/N_\mathrm{pk}$, where S_pk and N_pk are the size and the number of packets in the time window, respectively.
Average Packet Rate	$R_\mathrm{pk} = N_\mathrm{pk}/S_\mathrm{TW}$, with S_TW the time window's size
Average inter-arrival time	$\overline{\tau}_i = S_\mathrm{TW}/N_\mathrm{TW}$, where N_TW is the number of the time windows.

Table 2: Selected discriminators as input to the ANN core of the BotNet Detection Engine for SDNs.

2.3 BotNet Detection with Multilayer Perceptrons

We approach the Botnet detection task as a supervised machine learning problem by using a Training Set (TS) and a Test Set with features computed from real traffic data with known malicious/non malicious label. As features are numerical, we thought it was appropriate to use a Multilayer Perceptron (MLP), which is a classical and easily manageable ANN architecture, trainable with the error back-propagation weight update rule [11]. Our aim is to identify a small MLP that, after training, can be used as a real-time detection engine for Botnets.

In order to gain some insights about the complexity of the learning problem, we tried at first to determine the minimum number of hidden neurons in a MLP with a single hidden layer that achieved an acceptable error rate. This preliminary study is particularly important for the Botnet detection task, as a large number of neurons in the MLP results in longer computation time that may negatively affect the real-time performance of the desired detection engine. We found out that just 5 hidden sigmoid neurons were sufficient to achieve an acceptable error with a small improvement by using a higher number of hidden neurons. Consequently we tried to increase the number of hidden layers, just to discover a small improvement in the performance of the Botnet detection.

Details about training and evaluating the MLPs are given in the following section 2.4.

Our Botnet detection engine is developed as an external software **kbDetector** that communicates via Northbound-API with the OpenDayLight controller, in order to capture statistical information about traffic with the OpenFlow protocol and turns it into input features to be given to the MLP. Our strategy differs from the typical intrusion detection schemes designed for SDN running on the controller [3] because we want to avoid the critical bottleneck challenges that occur during collection of a great quantity of traffic from the switch. Our choice was dictated by the need to make it work with multiple types of controllers. **kbDetector** is external to the controller, which it interfaces with via REST.

kbDetector interactions with the SDN controller can be sketched as follows :

1. the switch sends the OpenFlow packet *ofp_packet_in* to SDN controller;
2. the controller replies to the switch dictating to create a *Flow Entry*, and notifying, via WebSocket to **kbDetector**, that a new flow has started;
3. when the *hard-timeout* expires, the switch sends a *ofp_flow_removed* OpenFlow message with the statistics to the controller which then sends a notification via WebSocket to **kbDetector** that contains the statistics and the stream id;
4. **kbDetector** calculates the features from incoming information and activates the trained MLP;
5. if the MLP classifies the flow as malicious, **kbDetector** retrieves the MAC address of the internal hosts involved in the flow (starting from the flow rule id) and via REST messages add a block rule of flow in order to isolate the infected host in the appropriate SDN network switches flow entry.

Moreover we developed the application **kbTool** that manages host blocked by **kbDetector**. **kbTool** communicates via REST API with the controller in order to share configuration files with **kbDetector**.

We performed some experiments for some selected MLP architectures that are reported in Fig. 3 for both CNs and SDNs. All the MLP architectures were tested with different sizes of the time-window. The main result we obtained is that an MLP trained over data concerning non-SDNs has a very satisfactory performance also when tested for Botnet detection in SDNs. As expected, though, the test error in SDNs is higher than that for non-SDNs that they were trained for. We can then note that longer time windows bring about a better test error for most architectures, and that a four layers MLP is good enough to achieve an error rate of less than about 0.05%.

2.4 Training the MLPs

For our experiments, we compute features based on portions of packet traffic until the layer 4 (ISO/OSI stack), thus the flows have the same characteristics in SDNs and CNs, and are completely transparent to an observer. The difference is in the specific set of discriminators used to train the ANN in the two cases of CNs and SDNs, as explained in Sec. 2.2.

The MLPs in the experimentation were trained over a TS obtained by joining public datasets that are commonly used in the literature for Botnet malicious traffic identification. The first is the CTU-13 dataset [4], it was created in the Czech Republic at CTU University, and consists of 13 scenarios that come from different categories of Botnets. Each scenario contains the traffic of a particular Botnet using different protocols and performing several actions. Then we use the ISOT dataset [12] by taking out a small percentage of malicious traffic (about 1% of the total) for positive examples, while negative

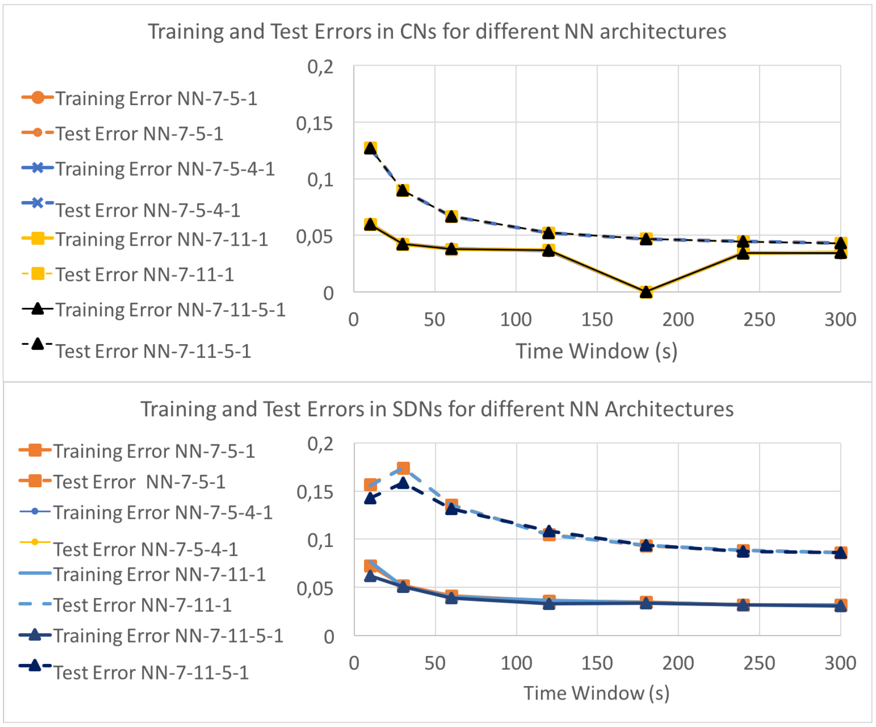

Figure 3: Training Errors for different time windows and multiple NN architectures for CNs and SDNs.

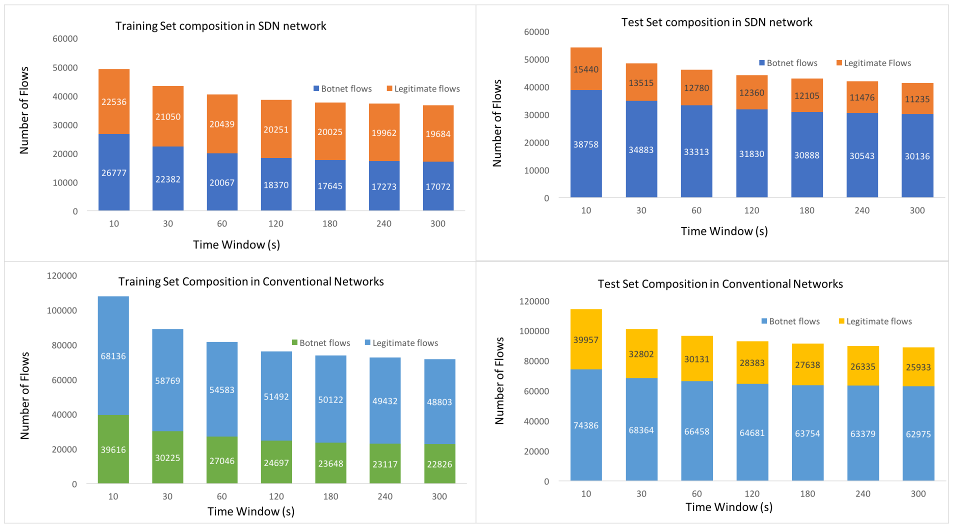

Figure 4: Training Set and Test Set compositions in CNs and SDNs.

examples (non-malicious traffic) were obtained by merging two different datasets from the Traffic Lab at Ericsson Research in Hungary [14], and from the Lawrence Berkeley National Lab (LBNL) [2]. The dataset that we obtain contains a large number of traffic flows collected from a variety of applications like web browsing, p2p, and online games during the period from 2001 to 2012 and all this traffic is structured in the *pcap*(packet capture) format that has become the common playground for network capture files in the open source world.

Due to the large size of these two datasets, we considered appropriate a subdivision into small *pcap* files performed via the opensource tool SplitCap [3] in order to create a division of flows.

The ISOT dataset dimension is 9.9 GB and contains 914812 different flows.

Different training sets and test sets of discriminators are created by applying the software **pcap2ml** we developed on purpose and using the following time windows: 10, 30, 60, 120, 180, 240, and 300 seconds. This choice allows a quick comparison with other studies [1] [7] [18]. For each time window, 70% of the files were used for the creation of the training set and the remaining 30% for the test set. New scenarios from the **Stratosphere IPS** [4] were added to the test set. The training and test files have been splintered by duplicates. Their composition is shown in Fig. 4.

MLPs training was stopped either when the error was considered acceptable (0%) or when a maximun number of 1500 training epochs was reached. The synaptic plasticity of the hidden sigmoid neurons was fixed to 0.3 and the momentum term to 0.1.

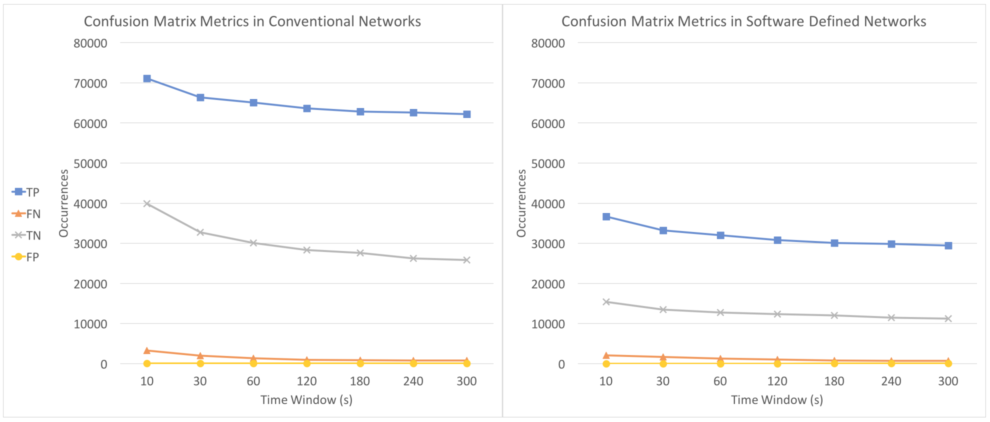

Figure 5: Confusion Matrix metrics in CNs and SDNs for the selected ANN architecture.

2.5 Performance Metrics

Performance of the trained MLPs is measured by means of the confusion matrix with its four quadrants that represent true positive TP, false positives FP, true negatives TN and false negatives FN, where obviously the actual positives result in total P = TP+FN and the negatives N = TN+FP. The confusion matrix metrics resulting from our experiments are presented in Fig. 5 for the selected ANN's architecture $(7-11-5-1)$[5]. The performance metrics for a supervised NN are computed from the four quadrants of the confusion matrix: the model *accuracy* is defined as A=(TP+TN)/(P+N), its *precision* as S=TP/(TP+FP), and the *recall* as Re=TP/P=TP/(TP+FN). The values of these performance metrics obtained in our tests are shown vs. the time window's duration in Fig. 6 for both CNs and SDNs with protocol OpenFlow 1.3, for the selected ANN architecture.

[2]Enterprise Tracing Project: www.icir.org/enterprise-tracing/

[3]Splitcap: www.netresec.com/?page= SplitCap

[4]https://stratosphereips.org/category/dataset.html project

[5]We made experiments for all architectures and obtained almost identical confusion matrices. Since we selected the architecture 7-11-5-1 as the one that gives the minimum errors (Ref. to Fig. 3), to simplify the Fig. 5 we reported only the confusion matrix associated with this architecture.

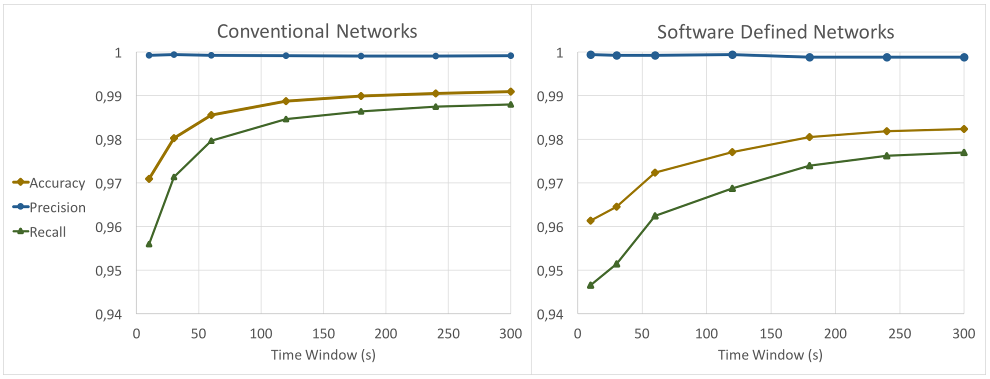

Figure 6: Comparison of the performance metrics in CNs (left plot) and SDNs (right plot) for the $7-11-5-1$ selected ANN architecture.

We may notice that, as a general trend, the precision appears independent from the duration of the time window in both cases of CNs and SDNs, showing a very slight decrease for larger time window. The accuracy and the recall, instead, grow with the increasing time window in all cases. Furthermore, comparing the two types of networks, the accuracy and the recall are higher in CNs than in SDNs.

3 Test case study: Mirai Botnet Traffic

The experience of the last October, when Mirai Botnet DDoSed 17 Dyn **Data Centers** (DCs), represents a case study from which we have to learn. Today DC networks manage high volumes of dynamic traffic and there is no doubt that the most dangerous threat for the performance of a DC is represented by a DDoS attack. Hence, as a test case for our experiments we consider a DC network. The spread of the malware on Internet has been strongly boosted by the online availability of the code for Mirai, which was posted by a hacker known as *Anna-Senpai*[6], then many cyber criminals started using the tool to assemble their own botnet armies. We simulate the same event in a SDN environment, by implementing a typical Fat-tree topology to create a Software-Defined Data Center (SDDC) with *4 pods* within the Mininet framework [6]. In the simulator we have infected with the Mirai malware six hosts in pods #1, #2 and #3, as shown in Fig. 7, by listening on TCP ports 23 and 2323, in order to simulate vulnerable IoT webcams. One of the infected hosts plays the role of the Mirai C&C server with a MySQL DBMS support, for *scan receiver* function used by Mirai. The victim host is located in pod#0, and undergoes two different DoS attacks: SYN Flood and UDP Flood as shown in Fig. 7.

num. of packets	Average of Bytes	num. of Bytes	Arrival rate	Byte/time
57	68.473	3903.0	0.0316	21.68
33	68.363	2256.0	0.183	12.53
9	67.333	606.0	0.05	3.366

Table 3: Traffic between C&C and Bot in Mirai botnet

Our experimentation relies on the evaluation of the traffic flows in the network in order to generate the set of relevant features to be given in input to the ANN for the Botnet detection and evaluate the performance of our approach in terms of confusion matrix metrics accuracy, recall

[6]https://github.com/jgamblin/Mirai-Source-Code

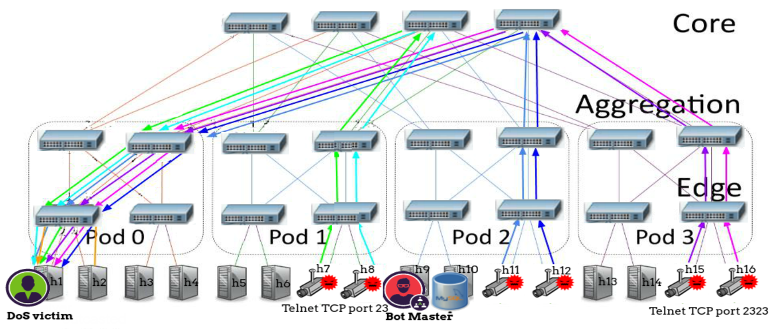

Figure 7: Diagram of the test case: the Mirai bots attack on the SDDC Fat-Tree topology.

and precision for Mirai detection. A sample of traffic between C&C and bots detected in a time window of 180 seconds is shown in Table 3. In particular, the packet set (in both directions) is represented by a quintuple consisting of source address, destination address, source port, destination port, and protocol at the transport level. We analyse the efficiency of the trained ANN as a Botnet detection tool by simulating UDP flood attacks by bots replicated from the Mirai malware. We obtained the best performance in the attack detection for time windows of large duration, as expected, and for time windows of 240 or 300 seconds the performance is roughly the same. In practical implementations, the only possible time window with the current version 1.3 of OpenFlow protocol is 300 seconds. However, a real-time system for botnet detection would benefit significantly from the use of time-window sizes of few seconds, especially if measures, like, e.g., isolation from the rest of the network, should be taken. The main reason is that the current protocol is not provided with a method for extracting flow statistics at arbitrary time intervals, and the only way is by removing a flow entry from the switch. Furthermore, in a high traffic environment, such as DCs, that manage a large number of streams, it is mandatory to reduce the number of rule removals per second. So, there is no choice but setting the time window at the highest possible duration.

4 Conclusion

In this paper we have shown that the statistical analysis and classification by a supervised neural network is an effective method for detecting the malicious traffic produced from bots during the attacks, for individuating the communication flow between bots and C&C and for preventing the attacks. In addition, we have shown that it is possible to block the attacks at the source, not simply a mitigation strategy. During the testing phase we noticed, like a serendipity, that DDoS attacks were automatically recognized by our neural network despite the work is explicitly focused on Botnet detection and not on the effects of their application. In particular, the use of neural networks has been as effective as other Machine Learning methods with 99% accuracy. The use of the OpenFlow protocol version 1.3 imposes some limits, like e.g., the impossibility of performing real-time detection, due to the constrained time window duration. It is expected that the OpenFLow 1.5.1 protocol, which is not implemented yet in ODL and in some switches open-source, will allow a more flexible customisation of the type of statistics that can be extracted from a stream. Future refinements of the proposed work include the analysis of performance of the detection application by implementing the *kbDetector* with Big-Data techniques and the implementation of a malicious traffic generator to improve the neural network training.

References

[1] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 247–255. IEEE, 2014.

[2] Xiao-Fan Chen and Shun-Zheng Yu. Cipa: A collaborative intrusion prevention architecture for programmable network and sdn. *Computers & Security*, 58:1–19, 2016.

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

[4] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

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

[6] Eric Jo, Deng Pan, Jason Liu, and Linda Butler. A simulation and emulation study of sdn-based multipath routing for fat-tree data center networks. In *Proceedings of the 2014 Winter Simulation Conference*, Piscataway, NJ, USA, 2014. IEEE Press.

[7] G Kirubavathi and R Anitha. Botnet detection via mining of traffic flow characteristics. *Computers & Electrical Engineering*, 50:91–101, 2016.

[8] A. Le, P. Dinh, H. Le, and N. C. Tran. Flexible network-based intrusion detection and prevention system on software-defined networks. In *2015 International Conference on Advanced Computing and Applications (ACOMP)*, pages 106–111, Nov 2015.

[9] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[10] R. A Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro. Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys (CSUR)*, 45(4):45, 2013.

[11] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[12] S. Saad et al. Detecting p2p botnets through network behavior analysis and machine learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 174–180. IEEE, 2011.

[13] Lisa Schehlmann and Harald Baier. Coffee: a concept based on openflow to filter and erase events of botnet activity at high-speed nodes. In *GI-Jahrestagung*, pages 2225–2239, 2013.

[14] Géza Szabó, Dániel Orincsay, Szabolcs Malomsoky, and István Szabó. On the validation of traffic classification algorithms. *Passive and active network measurement*, pages 72–81, 2008.

[15] F. Tariq and S. Baig. Botnet classification using centralized collection of network flow counters in software defined networks. *International Journal of Computer Science and Information Security*, 14(8):1075, 2016.

[16] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 349–360. ACM, 2012.

[17] Wen Wang, Wenbo He, and Jinshu Su. Network intrusion detection and prevention middlebox management in sdn. In *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*, pages 1–8. IEEE, 2015.

[18] David Zhao et al. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, 2013.