STATISTICAL FINGERPRINT-BASED IDS IN SDN ARCHITECTURE

Francesco Bigotto

University of Genoa
Via Opera Pia 13
16145, Genoa, Italy
francesco.bigotto@gmail.com

Luca Boero

University of Genoa
Via Opera Pia 13
16145, Genoa, Italy
luca.boero@edu.unige.it

Mario Marchese

University of Genoa
Via Opera Pia 13
16145, Genoa, Italy
mario.marchese@.unige.it

Sandro Zappatore

University of Genoa
Via Opera Pia 13
16145, Genoa, Italy
sandro.zappatore@.unige.it

ABSTRACT

The number of people accessing the Internet is growing rapidly leading, on one hand, to new possible attacks used by cyber criminals and, on the other hand, to an increased complexity in the network management. It is crucial designing systems able to prevent cyber attacks. At the same time, many efforts are provided in order to get tools that can make easier network management. The Software Defined Networking (SDN) paradigm has been designed with this aim allowing network administrators to manage networks easily. This paper deals with an original Intrusion Detection System that exploits an SDN architecture to get the information needed to feed a statistical-fingerprint based IDS. Specifically the proposed system collects traffic data suitable to detect the possible presence of malware inside the network, and describes the design and implementation of an application developed upon a SDN controller (Ryu) and its role in the malware detection process.

Keywords: Software Defined Networking, Intrusion Detection System, Malware Detection, Flow Identification

1 INTRODUCTION

An Intrusion Detection System (IDS) is a hardware/software component or group of devices and components designed to monitor a network or a system to detect malicious activity. IDSs (Sabahi and Movaghar 2008) may be classified depending on: data source (host based, network based, and hybrid); detection time (on and off line); environment (wireless, wired, and heterogeneous); architecture type (centralized/distributed); reaction (active/passive); and processing (Misuse Detection and Anomaly Detection). We focus the attention on reactive network based systems, possibly operating online over heterogeneous networks. In parallel with the evolution of IDSs, the need of simplifying network management has brought to the development of the Software Defined Networking (SDN) (Stallings 2013, Nunes et al. 2014) paradigm. SDN is a networking

architecture that decouples data and control actions. Data forwarding functions are located within devices (switches, routers, gateways) called SDN switches, control functions are concentrated in SDN controllers. The communication between an SDN controller and the SDN switches under its domain is implemented through the OpenFlow signaling protocol.

Combining a malware detector IDS and SDN may represent a step forward in the service provided by SDN and may allow simplifying the IDS design by means of SDN functions.

The remainder of this paper is organized as follows. Section 2 briefly reviews the state of the art. In Section 3 a description of traffic flow is given and the app's code and the main functional blocks of the proposed architecture are illustrated. Section 4 shortly describes the malware tackled in the paper. Section 5 presents the emulated scenario. Section 6 shows the results of the proposed approach. Section 7 contains the conclusions.

2 STATE OF THE ART

Although the solution proposed in (Zhang 2013) is not directly linked to malware detection, it introduces a possible approach for collecting flow statistics in SDN. The author explains that the use of SDN is essential to allow deep accuracy and granularity without introducing too much communication overhead inside the network. SDN is able to control both temporal (how often to collect data) and spatial (how deep should the inspection be inside the packet) granularity and to distribute flow counting tasks in a smart way among all the switches in the network.

In (Skowyra, Bahargam, and Bestavros 2013), the authors design an environment that exploits SDN to implement an IDS for a network of Embedded Mobile Devices, so avoiding the problems of standard IDSs within this kind of network such as the inability to cope with end-host mobility and the limited set of actions which can be taken in response to anomalies. Without specifying any particular anomaly detection algorithm, the authors classify what kind of anomalies can be observed: Stateless Flow, Stateful Flow, Volumetric Anomalies, and Physical Anomalies. Within this classification, the category closer to our approach is the volumetric anomaly, which is revealed from the statistics sent by the switches to the controller. However, the reference (Skowyra, Bahargam, and Bestavros 2013) focuses on the changes of the overall traffic volume, whereas we consider more specific statistical features of flows.

The authors of (Wang et al. 2016) propose a behavioral-based Security Monitoring System that exploits the flexibility of SDN to orchestrate the detection system. The used controller is Ryu but the collection of statistical data and the classification of flows are delegated to sFlow that represents an additional element in the network. The used classifier in (Wang et al. 2016) is SVM, a supervised classification algorithm in line with our design choice.

In (Braga, Mota, and Passito 2010) a DDoS detector is implemented by SDN to allow recognizing malicious flows without any deep packet inspection. The system architecture is similar to the one proposed in this paper, even though some differences arise, mainly concerning the classification phase. The reference (Braga, Mota, and Passito 2010) chooses NOX as a controller and develops an application that collects flows' statistics at predetermined time instants. 6 features are extracted from the collected statistics. Self Organizing Map (SOM), an unsupervised classifier, is employed for flow classification.

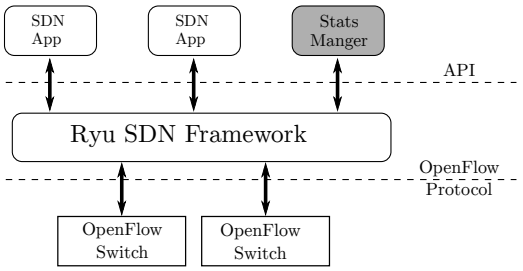

Figure 1: Ryu scheme

3 SDN CONTROLLER EMPLOYED WITHIN AN IDS

3.1 General Description of Ryu

The controller chosen to implement our IDS is Ryu (Ryu 2016), an open source, component-based, software defined networking framework written in python, which provides software components by using API that make easy for developers to create new network management and control applications. The general structure of the code is sketched in Figure 1that highlights how the main central framework, responsible for the whole system management, communicates with the switches through the OpenFlow protocol and with the different apps through the APIs.

The main contribution of this paper is the design and implementation of the application (app) called `stats_manager`, aimed at managing the flow tables in the used SDN switch, providing the information needed to classify the traffic, and making decisions about "malware/normal" on the examined traffic.

3.2 Flow Structure

A traffic flow is usually defined as a group of packets sharing some common characteristics. We use one of the most common conventions according to which a flow is a set of packets having the same 5-tuple: IP source address, IP destination address, TCP/UDP source port, TCP/UDP destination port, Protocol field of IP header.

For each traffic flow we store the data structure in Table 1 where the first 5 lines define the flow; the following 9 lines contain the features selected in our previous paper (Boero et al. 2016), which can be computed by the SDN controller directly from the information received by the SDN switch through the statistic reply message; last four lines are better explained in the following:

- **state:** describes the state of each flow with respect to the current *time window* (see section 3.3):
  - 'B' means "begun"
  - 'C' means "continued"
  - 'E' means "ended"
- **extra_p and extra_b:** some packets such as the packet-in, i.e. the first packet of a new flow, and corresponding bytes cannot be considered by the SDN switch counters. In other words they cannot be detected by using the statistic reply message. These fields allow the app not to lose this information. These aspects will be elaborated below.
- **label:** it is the ground truth about the nature of the flow: normal or affected by malware. During the training phase this information is used to give correct examples to the classifier. In the test phase this field is ignored by the classifier and it is used only to assess the system performance.

Table 1: Flow Dictionary Structure

Key	Description
IP_src	IP source address
port_src	TCP/UDP source port
IP_dst	IP destination address
port_dst	TCP/UDP destination port
protocol	Protocol field in the IP header
first_len	length of the first packet
pkt_count	number of packets in the flow
byte_count	number of bytes in the flow
dur_sec	duration of the flow(in seconds)
dur_nsec	nanoseconds exceeding dur_sec
byte_rate	byte rate of the flow
pkt_rate	packet rate of the flow
avg_iat	average inter-arrival time between packets
avg_pl	average packet length
state	current state of the flow
extra_p	number of packets not registered in the statistics
extra_b	number of bytes not registered in the statistics
label	class of the flow ('normal' or 'virus')

Data related to a single flow are stored in the application by two structures: `active_flows`, which contains all the flows currently active in the network, and `ended_flows`, which maintains the information about the flows recently terminated.

3.3 Structure of the "Stats_Manager" Application

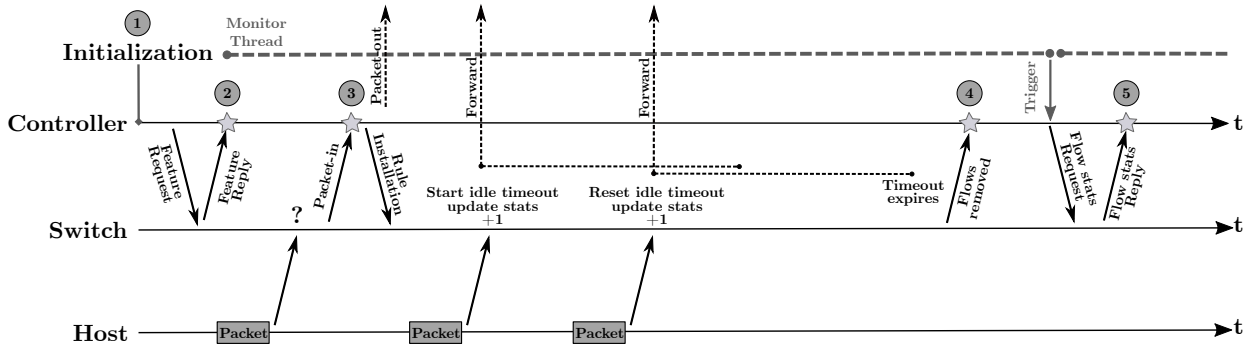

Figure 2: General picture of how the app interacts with Ryu

Being Ryu an event-based controller, developing an application for this framework means coding functions that will be executed when a particular event happens. Figure 2 depicts the most relevant events for our goal (denoted with a star) and their temporal action. In detail:

1) **Initialization**: this event happens only when the controller instance is created, all the internal variables are initialized, and the `monitor` thread, which will periodically trigger the statistics' request, is started. The controller periodically sends a flow statistic request message in order to gather information about the network traffic from all the switches in the network (just the used one in our case). The time period elapsed between two consecutive messages is called *time window* and, in this paper, it was set to 30 seconds empirically.
2) **Feature Reply**: shortly after the boot, the controller needs to collect some information about the network it has to control. To this purpose it sends a feature request message to the connected switch that, in turn, answer through a feature reply message, announcing what optional features it supports. Being the first communication between controller and switch, the initialization of flow tables occurs.
3) **Packet-in**: the first packet of a new flow doesn't match any rule in the flow tables, thus it is sent directly to the controller where two kinds of actions are performed: standard packet-in management and start of the IDS statistics collection. The former relates to the tasks that are usually done by every SDN controller: the path for the new flow is computed, the needed rules are installed in the switch, and the first packet is sent back to the sender switch encapsulated in a packet-out message. The latter is strictly related to "Stats_Manager" application: the unknown flow must be recorded in the active flows database and the length of the first packet (first_len in Table 1) is stored.
4) **Flow removed**: if no packet matches a given rule for a specified number of seconds, called *idle_timeout*, the flow is considered ended and the rule is removed from the table. When this occurs, the switch sends a packet to the controller, containing the statistics of the removed rule, to notify the event. This is the *asynchronous* way to collect statistics because it may happen at any time instant, as it is strictly dependent on the traffic. The controller extracts the flow identifiers and uses them as indexes to retrieve the specific flow in the active flows database. The measured features (pkt_count, byte_count, dur_sec, dur_nsec, in Table 1) are immediately saved and the derived features (byte_rate, pkt_rate, avg_iat, avg_pl) inferred from the previous ones through simple processing. Finally, the flow entry is removed from the active flow database and inserted in the ended flow database.
5) **Flow stats Reply**: as stated before, every 30 seconds the controller asks the switch for the statistics of the currently active flows. As a reply, the switch sends an OpenFlow packet containing the statistics of all the flows to the controller, as illustrated in Figure 3. This is the *synchronous* way to collect statistics since it is regularly scheduled by the controller. Similarly to the asynchronous case, every flow identifier is used as an index to look for the flow entry in the active flows database. Once found, the entry is updated with the new data, before being stored again.

Upon flow stat replies are received at the end of a *time window* additional processing steps are done by the classification process, as shown in Figure 4.

All the data related to both active and ended flows are merged to form a unique dataset. The subsequent processing steps depend on the phase the controller is actually performing.

- *Training Phase*: received data are added to the training set of the classifier; when enough data are gathered, they are used to train the classifier. This process produces the classifier's model as a result, which is stored and used in the test phase.
- *Test Phase*: received data are directly addressed to a classifier's module for the actual classification. Even though the output of a single flow classification is just a binary label stating "malware" or "normal", the system produces two different text files as output.
  The first one contains the details of all the analyzed flows and reports all the fields of the flow structure, in Table 1, together with the result of the classification process. Consequently it is possible to check how the classifier performs for each single flow.

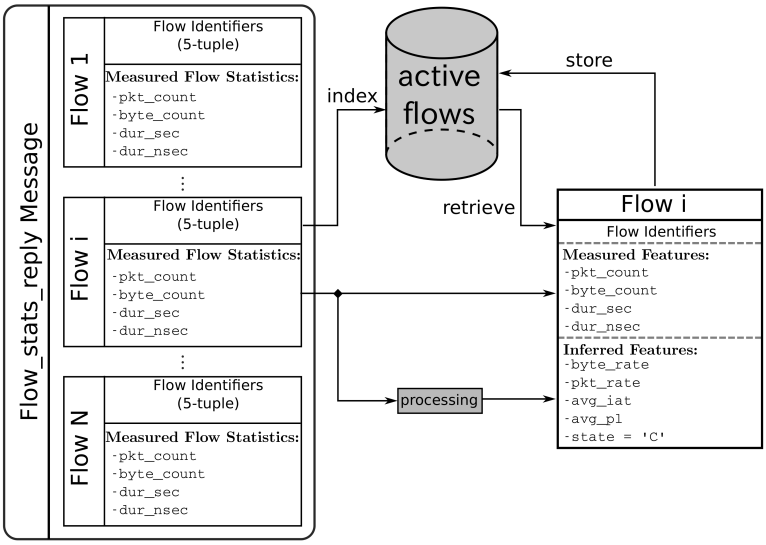

Figure 3: Flow Stats Reply

The second file contains a general report for every time window, containing the total number of flows, the number of normal and malware flows, and the usual classification metrics: true positives, true negatives, false positives, false negatives.

4 USED TRAFFIC

In this paper the traffic considered as malware is composed of a mix of traffic generated by different malwares briefly described in the following:

AlienspyRat: it belongs to the Remote Access Trojan family. Once activated, this software allows collecting system information, updating and downloading other malware. The malware sends captured information to the central server and waits to receive commands.

Asprox: is a spam botnet emerged in 2007. It is known for sending mass of phishing emails used in conjunction with social engineering lures (e.g. booking confirmations, postal-themed spam, etc.). This botnet arrives as an attachment to spammed messages disguised as notifications from postal companies, as well as airline booking confirmations.

Cutwail: is a botnet used to generate spam emails using the contacts in address books. The malware receives instructions from a command and control server about which and how many messages to send. Once it has completed the task, it sends a full report on the number of sent messages and on any found errors to the controller.

Darkness: also called Optima, it is a botnet specialized in DDoS attacks. It waits for commands from a Command and Control (C&C) server that sends encrypted control messages to the infected machines.

Kuluoz: is a botnet aimed at sending phishing emails that simulate messages sent by postal administrations, combined with the use of social engineering techniques. Furthermore, the control server is able to send commands to the infected machines to download and execute pay-per-install programs so to ensure gains to the botnet manager.

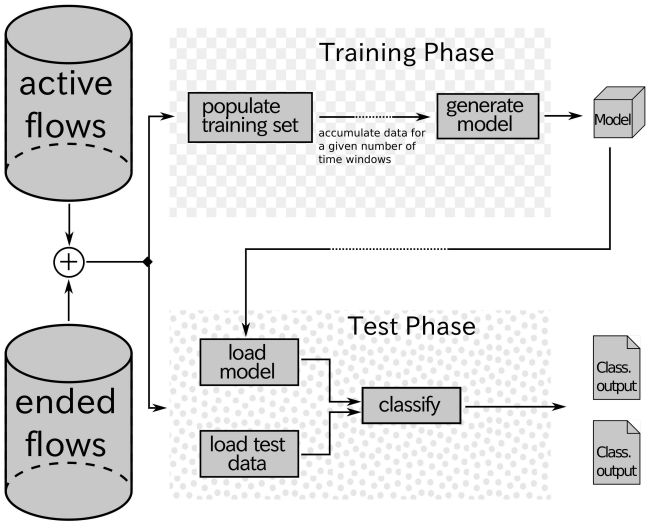

Figure 4: Classification Process

Madness: is a distributed denial of service botnet growing in size and popularity. It infects computers running Windows and communicates with its command and control server via HTTP by using a simple client-server model.

Neris: is a botnet that uses an http-based channel to communicate with the C&C server. The main aims of this malware, after establishing a communication with the C&C, are to send spam and perform click-fraud by using advertisement services.

Purplehaze: is a botnet targeted to take the control of machines with the aim of using them to generate many clicks on online advertising sites. It can generate a high volume of traffic on web sites containing advertisements or links in very short time.

Ramnit: this trojan primarily spreads through a contact with infected removable devices, mainly USB flash memory. Once installed, this program connects with a remote server via TCP port 443, sending all the obtained information on the infected machine.

Tbot: is a Trojan that targets older Windows versions in order to open a back door in the system and allow the attacker to use the machine without the owner's consent.

ZeroAccess: this Trojan has the main purpose of assuring money to the attacker via pay-per-click advertisement. This tool is able to create a hidden and encrypted file system where it can save its members in total freedom, as well as all other additional malware that can download.

Zeus: has the main purpose of stealing information related to the bank accounts of the targets by means of techniques such as man-in the-browser, keystroke logging and form grabbing. The spread of the virus occurs mainly through drive-by downloads, initiated by mistake from the user, or phishing schemes. There is a server that acts as a control center from which the commands start.

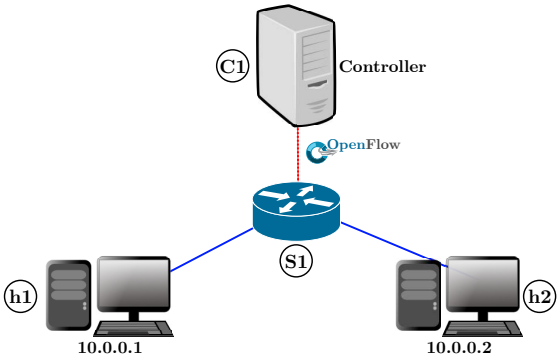

Figure 5: Network topology

5 TESTBED AND SCENARIO SETUP

5.1 The network

The network is emulated by means of mininet (Lantz, Heller, and McKeown 2010), an open source program which allows recreating a realistic virtual network inside a pc. More precisely, it is a *network emulation orchestration system* that runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network.

Mininet supports the SDN paradigm by implementing its main components: the switches and controller. Regarding switches, mininet leverages *Open v-Switch* (Pfaff et al. 2015), an open source, multilayer virtual switch, explicitly designed to enable network automation through programmatic extensions, while still supporting standard management interfaces and protocols.

Concerning the controller, mininet implements the basic OpenFlow reference controller by default but it is possible to specify which controller to use at launch time. In our work we set as SDN controller the instance of Ryu that contains the `stats_manager` app running on the same machine that emulates the network.

The chosen network topology to perform the simulations is sketched in Figure 5. It consists of two hosts ($h1$ and $h2$) connected to a single SDN switch ($S1$) that communicate with the SDN controller ($C1$) through a dedicated channel.

5.2 Traffic emulation

In order to emulate a real scenario, we merged the flows produced by the malwares described in Section 4 with normal, malware-free traffic. We have captured normal flows by setting a network switch in our laboratory in port monitoring mode and sniffing all the traffic coming to the switch. Before starting the actual transmission, some manipulations on the original captured flows are performed with the help of the Tcpreplay (Turner and Bing 2011) and Wireshark (Combs et al. 2007) command line tools:

- IP addresses was rewritten in such a way that all the packets appear to be exchanged between only two IP addresses: 10.0.0.1 and 10.0.0.2.
- In order to keep the simulation time reasonable, we cut the longest traces in order to have the same duration for normal and malware traffic. The duration of our experiments was approximately 2 hours.

- We divided the 12 malware's captures in 2 groups of 6 and added one malware-free capture to each group. In particular, **group** 1 contained Asprox, Cutwail, Darkness, Madness, Purplehaze, and Zeus; while **group** 2 contained Alienspy, Kuluoz, Neris, Ramnit, Tbot, and Zeroaccess.
- All the captured flows of both groups were temporally shifted in order to begin at the same time instant. Finally, they were merged together in two final pcap files to be used in the emulations.

5.3 The Classifier

Ensemble learning algorithms (e.g. random forest, bagging and boosting) have received an increasing interest because they are more accurate and robust to noise and outliers than single classifiers (Dietterich 2000). The philosophy behind ensembles classifiers is that a set of classifiers performs better than an individual classifier. (Breiman 2001) introduced a new and promising classifier in 2001 called Random Forest that consists of a collection of tree-structured classifiers, each one initialized with an independent identically distributed random vector x. Random Forest presents many advantages: it runs efficiently on large databases, it is able to handle thousands of input variables without variable deletion, it is computationally lighter than other tree ensemble methods. Moreover, Random Forest estimates which variables are more important in the classification.

We performed a classifier performance evaluation in (Boero et al. 2016) and Random Forest provided the best performance. It has been used also in the following tests.

When a sample is given as input to the classifier every tree independently decides the class of the samples $\hat{C}_b(x)$ and casts a unit vote for it. The most voted class is the final output of the classifier $\hat{C}^B_{rf} = majorityvote\{\hat{C}_b(x)\}^B_1$.

The implementation of the classifier in python language is taken from the scikit-learn library (Pedregosa et al. 2011), whose APIs are inserted into Ryu's code through a wrapper class we wrote for this purpose.

6 RESULTS

Table 2 presents the overall results related to the two groups of malware described in Section 4. When group 2 is tested, group 1 is used in the training phase and viceversa, as shown in Table 2. The indicated values refer to the percentages of the parameters True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN), and Accuracy (ACC, computed as $\frac{TP+TN}{TF}$, where TF is the total number of flows).

Table 2: Results of the Simulations

Sim	Train	Test	TP	TN	FP	FN	ACC
1	group 1	group 2	0.845	0.987	0.013	0.155	0.887
2	group 2	group 1	0.978	0.964	0.036	0.022	0.972

In this particular application, since the result of the flows' classification is available for each time window, we computed an overall value for the aforementioned quantities. For each of them, a weighted average was

computed as follows:

$$TP = \frac{\sum_{t=1}^{T} TP_t}{\sum_{t=1}^{T} F_t^m}, \quad TN = \frac{\sum_{t=1}^{T} TN_t}{\sum_{t=1}^{T} F_t^n}$$
$$FP = \frac{\sum_{t=1}^{T} FP_t}{\sum_{t=1}^{T} F_t^n}, \quad FN = \frac{\sum_{t=1}^{T} FN_t}{\sum_{t=1}^{T} F_t^m}$$
$$ACC = \frac{\sum_{t=1}^{T} TP_t + TN_t}{\sum_{t=1}^{T} F_t^T}$$

where:

- t is a counter of the *time window* when the classification takes place.
- T is the last time instant of the simulation.
- TP_t is the number of TP at instant t.
- F_t^m is the number of malware flows at instant t.
- F_t^n is the number of normal flows at instant t.
- F_t^T is the total number flows at instant t.

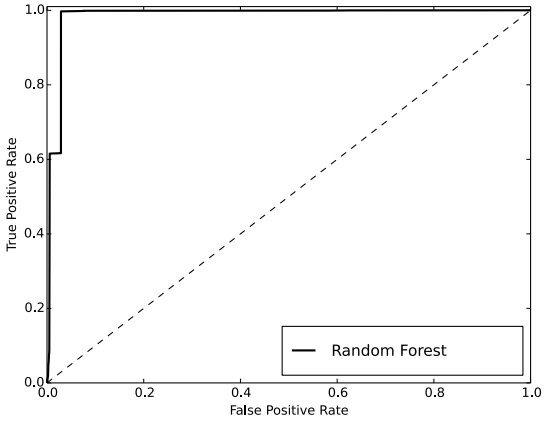

Figure 6: ROC Diagram of Simulation 1

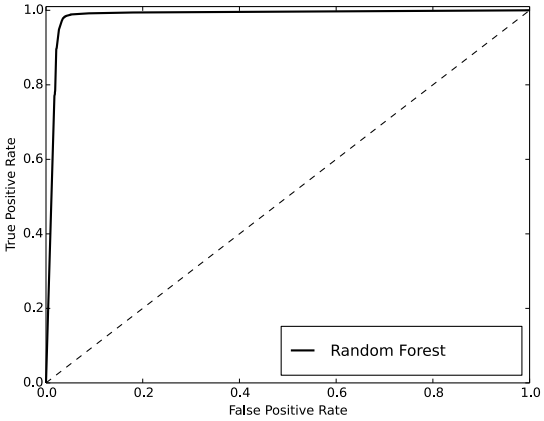

Figure 7: ROC Diagram of Simulation 2

Figures 6 and 7 show the Receiver Operating Characteristic (ROC) curves of the two simulations in Table 2. The dashed line is the "line of no discrimination" given by a random guess. The graphs show a performance very close to the ideal one (point $(0,1)$) in both figures. It means that our designed system reaches a very high value of correct detections (TP) with a very low probability of false alarms (FP). It is worth remarking that, in both cases, the classifier is trained with a set of malware flows which is different from the one employed in the testing phase, i.e. the malware used to test the classifier is not part of the dataset used to build the model for the Random Forest algorithm during the training phase.

7 CONCLUSION

The implementation of a combined malware detector IDS and SDN system allows simplifying the IDS design also improving the service offered by an SDN architecture. The paper has presented a possible implementation of an integrated SDN-IDS Ryu-based controller application devoted to detect the possible presence of malwares traversing the network. The results obtained through a large simulation campaign

have demonstrated the effectiveness and robustness of the proposed system, which has reached an accuracy level ranging from 88 and 97%.

This is a little step forward in the use of an SDN approach that can lead to innovative solutions to manage and secure networks.

REFERENCES

Boero, L., M. Cello, M. Marchese, E. Mariconti, T. Naqash, and S. Zappatore. 2016. "Statistical fingerprint-based intrusion detection system (SF-IDS)". *International Journal of Communication Systems* vol. 30 (10).

Braga, R., E. Mota, and A. Passito. 2010. "Lightweight DDoS flooding attack detection using NOX/OpenFlow". In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 408–415. IEEE.

Breiman, L. 2001. "Random Forests". *Machine Learning* vol. 45 (1), pp. 5–32.

Combs, G. et al. 2007. "Wireshark". *Web page: http://www. wireshark. org/last modified*, pp. 12–02.

Dietterich, T. G. 2000. "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization". *Machine learning* vol. 40 (2), pp. 139–157.

Lantz, B., B. Heller, and N. McKeown. 2010. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pp. 19:1–19:6. New York, NY, USA, ACM.

Nunes, B. A. A., M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. 2014. "A survey of software-defined networking: Past, present, and future of programmable networks". *IEEE Communications Surveys & Tutorials* vol. 16 (3), pp. 1617–1634.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* vol. 12, pp. 2825–2830.

Pfaff, B., J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. 2015. "The Design and Implementation of Open vSwitch". In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pp. 117–130. Berkeley, CA, USA, USENIX Association.

Ryu 2016. "Framework Community: Ryu sdn controller". https://osrg.github.io/ryu/.

Sabahi, F., and A. Movaghar. 2008. "Intrusion detection: A survey". In *Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on*, pp. 23–26. IEEE.

Skowyra, R., S. Bahargam, and A. Bestavros. 2013. "Software-defined ids for securing embedded mobile devices". In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pp. 1–7. IEEE.

Stallings, W. 2013. "Software-defined networks and openflow". *The internet protocol Journal* vol. 16 (1), pp. 2–14.

Turner, Aaron and Bing, Matt 2011. "Tcpreplay". https://tcpreplay.appneta.com/.

Wang, P., K.-M. Chao, H.-C. Lin, W.-H. Lin, and C.-C. Lo. 2016. "An Efficient Flow Control Approach for SDN-Based Network Threat Detection and Migration Using Support Vector Machine". In *e-Business Engineering (ICEBE), 2016 IEEE 13th International Conference on*, pp. 56–63. IEEE.

Zhang, Y. 2013. "An adaptive flow counting method for anomaly detection in SDN". In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 25–30. ACM.

AUTHOR BIOGRAPHIES

FRANCESCO BIGOTTO got his B.Sc. degree in Electronic and Telecommunication engineering in 2015, at the university of Genoa, with a thesis on a load balancing algorithm for multi-controller SDN. He then focuses on telecommunication domain, getting the M.Sc. degree in Multimedia Signal Processing and Telecommunication Networks, in 2017, in the same University. His work for the master thesis, from which this article comes from, is related to malware detection in SDN using a Machine Learning engine to analyze the statistical fingerprint of the traffic flows.

LUCA BOERO got his B.Sc. in Telecommunication Engineering in 2012 and his M.Sc. in Multimedia Signal Processing and Telecommunication Networks at the University of Genoa with a thesis on Software Defined Networking (SDN), developed at the Satellite Communications and Networking Laboratory (SCNL) in March 2015. He is currently a Ph.D. Student at the SCNL, and his main research activity concerns SDN, in particular Routing and Resource Allocation algorithms and Network Security for SDN Networks.

MARIO MARCHESE was born in Genoa, Italy in 1967. He got his "Laurea" degree cum laude at the University of Genoa, Italy in 1992, and his Ph.D. (Italian "Dottorato di Ricerca") degree in "Telecommunications" at the University of Genoa in 1997. From 1999 to January 2005, he worked with the Italian Consortium of Telecommunications (CNIT), by the University of Genoa Research Unit, where he was Head of Research. From February 2005 to January 2016 he was Associate Professor at the University of Genoa. Since February 2016 he has been Full Professor at the University of Genoa. He authored/co-authored more than 300 scientific works, including international magazines, international conferences and book chapters. His main research activity concerns: Networking, Quality of Service over Heterogeneous Networks, Software Defined Networking, Satellite DTN and Nanosatellite Networks, Network Security.

SANDRO ZAPPATORE was born in Savona, Italy. He received the "Laurea" and Ph. D. degrees in Electronic Engineering from the University of Genoa, Italy, in 1995 and 1990, respectively. In 1990 and 1991, he was awarded two scholarships from the Italian National Council of Research (CNR) within a National Project on Telecommunications. From 1992, he is a Researcher at the Department of Communication, Computer and Systems Science of the University of Genoa, where he is currently an Associate Professor of telecommunications and teaches the course "Digital Communications". His interests are both in the area of signal processing, especially audio and video coding, and computer networks. His current research is devoted to Multimedia Network Applications. In this field, he is the technical manager of some national projects, funded by the Italian Ministry of the Education, University and Scientific Research (MIUR), concerning the networked access and management of remote and complex laboratories. Currently, he is interested in grid-based platform for the control of remote laboratories, in wireless sensor networks, and network security. He is author of many papers, appeared on International Journals and on proceedings of International conferences.