Security in the internet of things: botnet detection in software-defined networks by deep learning techniques

Ivan Letteri*, Giuseppe Della Penna and Giovanni De Gasperis

Department of Information Engineering,
Computer Science and Mathematics,
University of L'Aquila,
L'Aquila, Italy
Email: ivan.letteri@graduate.univaq.it
Email: giuseppe.dellapenna@univaq.it
Email: giovanni.degasperis@univaq.it
*Corresponding author

Abstract: The diffusion of the *internet of things* (IoT) is making cyber-physical *smart devices* an element of everyone's life, but also exposing them to malware designed for conventional web applications, such as botnets. Botnets are one of the most widespread and dangerous malware, so their detection is an important task. Many works in this context make use of general malware detection techniques and rely on old or biased traffic samples, making their results not completely reliable. Moreover, software-defined networking (SDN), which is increasingly replacing conventional networking especially in the IoT, limits the features that can be used to detect botnets. We propose a botnet detection methodology based on deep learning techniques, tested on a new, SDN-specific dataset with a high (up to 97%) classification accuracy. Our algorithms have been implemented on two state-of-the-art frameworks, i.e., Keras and TensorFlow, so we are confident that our results are reliable and easily reproducible.

Keywords: cyber-physical devices; internet of things; IoT; software-defined networking; SDN; botnet detection; machine learning; neural networks; deep learning; network security.

Reference to this paper should be made as follows: Letteri, I., Della Penna, G. and De Gasperis, G. (2019) 'Security in the internet of things: botnet detection in software-defined networks by deep learning techniques', *Int. J. High Performance Computing and Networking*, Vol. 15, Nos. 3/4, pp.170–182.

Biographical notes: Ivan Letteri received his Master degree in Computer Science in 2016 from the University of L'Aquila, Italy, where he is currently a PhD student. His current research focuses on cyber security and machine learning.

Giuseppe Della Penna received his Master degree in Computer Science in 1998 from the University of L'Aquila, Italy and PhD in Computer Science in 2002 from the University of Rome 'La Sapienza'. Currently, he is an Associate Professor at the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila, Italy. His research interests include formal methods applied to languages, security and verification. He developed several research tools like the CMuphi model checker and the UPMurphi universal planner.

Giovanni De Gasperis is Researcher and Assistant Professor at the Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Italy. He graduated in Electronics Engineering at University of L'Aquila in 1991 where also earned a Doctorate in Electronics Engineering in 1995. From 1995 to 1998, he was PostDoctoral Fellow at MD Anderson Cancer Center, Houston, TX, USA, doing research on computer vision applied to automation of electro-rotation measurements of living cells. His current research focus is on the application of computational intelligence and machine learning to the smart energy management problem, autonomous systems and cognitive robotics and cyber-security. He is an active developer and coordinator of several scientific computing tools, such as the DALI open-source logic multi agents systems framework based on Prolog and high performance TaLTaC V3 text mining engine.

This paper is a revised and expanded version of a paper entitled 'Botnet detection in software defined networks by deep learning techniques' presented at the 10th International Symposium on Cyberspace Safety and Security, CSS 2018, Amalfi, Italy, 29–31 October 2018.

1 Introduction

Cyber-physical systems (CPS) are built from the seamless integration of computation and physical components (National Science Foundation, 2018). The growing diffusion of the *internet of things* (IoT), made up of cyber-physical *smart devices* connected through virtualised cloud architectures, is making CPS a common element of everyone's life, but also exposing them to a number of *cyber attacks* that were initially designed to target conventional web applications and internet-enabled devices.

A *botnet* consists of a number of internet-connected devices, each of which runs one or more *bots*, i.e., systems compromised by some kind of malware controlled by an attacker (the *botmaster*). Botnets can be very difficult to detect and eradicate due to their size and distribution. Therefore, in the last twenty years, botnets have gained a lot of attention from security researchers (see, e.g., Abu Rajab et al., 2006; Bailey et al., 2009; Tanwar and Goar, 2014; Antonakakis et al., 2017).

In particular, IoT botnets have been designed to perform DDoS attacks on IoT networks in order to break smart devices of everyday use. As an example, the Mirai (Antonakakis et al., 2017) botnet scans the internet for IoT systems still making use default usernames and passwords such as 'admin'. Once found, these systems are infected by a malware which can be used to launch DDoS attacks.

On the other hand, most of the techniques used for the mitigation of DDoS attacks in traditional networks are impractical or useless in the IoT. Indeed, managing and securing large-scale, heterogeneous networks of devices like the IoT is a challenging task. To this aim, IoT networks are increasingly adopting (Yassein et al., 2017) the software-defined networking (SDN) approach (Kamal et al., 2016), which helps to solve these problems thanks to its flexibility and global network view (Flauzac et al., 2015).

Software-defined networking is an emerging networking approach that enables an easy and efficient (re)configuration in order to improve network performance. To this aim, SDN centralises the network intelligence in a single component, separating the packet forwarding process (*data plane*) from the routing (*control plane*). Elements on the control plane (the controllers) can communicate with elements on the data plane (e.g., switches) through the *OpenFlow* (Open Networking Foundation, 2012) protocol. In particular, SDN may help to effectively cope with DDoS attacks generated by a huge number of IoT devices since it provides network flow monitoring capability and centralised control of network devices (Özçelik et al., 2017).

Due to its particular structure, a SDN differs from a conventional network also for traffic monitoring-related aspects: indeed, there are traffic features, such as the packet ratio, that can be easily measured on both conventional and SDN networks, whereas some other features are hard to obtain from the data plane through OpenFlow requests. Obviously, talking directly with data plane such elements is sometimes possible through ad-hoc protocols, but this would break the SDN abstraction or make the approach very specific to the software running on the elements. Therefore, analysing network traffic in order to detect botnet attacks in a SDN requires a very different approach with respect to conventional networks. On the other hand, since SDN will increasingly replace conventional networking in the near future, it will also rapidly become the main diffusion field for botnets. Thus, SDN-based botnet analysis and detection is a very interesting field.

Current state-of-the-art approaches to botnet detection often make use of the so called *behavioural analysis*: this technique does not rely on the actual information transferred over the network (packet payload), but looks at some traffic features trying to detect specific patterns (Zhao et al., 2013) (for a general discussion on this technique and its applications see, e.g., D'Angelo et al., 2015, 2017). Indeed, botnets are continuously evolving and changing their behaviour, making quickly ineffective more established techniques such as, e.g., signature analysis, which is still widely adopted in malware detection engines, whereas behavioural analysis approaches have proved more appropriate to address the constant evolution of botnets (Miller and Busby-Earle, 2016). Behavioural analysis is often supported by machine learning techniques, to further improve its adaptability to the botnet evolution: indeed, by exploiting machine learning, it is possible to detect new attack schemes which have similarities with known attacks, thus preventing the so-called 'zero day' exploits. This dynamic adaptability of the security mechanisms, obtained through the employment of machine learning techniques, is especially useful when dealing with critical CPS, indeed some preliminary work already exists in this direction (see, e.g., Junejo and Goh, 2016).

In a previous paper (Letteri et al., 2018), we started working in this context with a very basic neural network classifier for such a task, and obtained encouraging results from tests carried out on a small dataset with simple features. In this paper we propose a novel botnet-specific detection methodology that exploits machine learning techniques, in particular multilayered neural networks (which fall in the field of *deep learning* techniques (LeCun et al., 2015)), and has been experimented on a new SDN-specific dataset, obtaining a very high (up to 97%) accuracy. In particular, the employed dataset derives from the *HogZilla* dataset (Resende and Drummond, 2018), which combines selected parts of the well-known CTU-13 (García et al., 2014) and ISCX 2012 IDS (Shiravi et al., 2012) datasets in order to obtain a large, complex set of more than 990,000 samples. From this dataset, we performed a further feature selection and filtering in order to make it a realistic dataset for a SDN. Therefore, our experiments have been carried out using real, recent botnet traffic samples that could be observed in a SDN.

Finally, our neural networks have been implemented and experimented relying on two state-of-the-art frameworks, i.e., Keras and TensorFlow, so we are confident that our experimentation results are reliable and easily reproducible.

To the best of our knowledge, there is no work in the literature that combines these three components, i.e., botnet-specific detection mechanisms, SDN-specific traffic

features and behavioural analysis through multi-layered neural networks.

The paper is organised as follows. Section 2 lists some related works relative to our areas of interest, i.e., SDN, machine learning and botnet detection. Section 3 introduces some background notions on botnets and software defined networking, useful to understand the rest of the paper. Then, in Section 4 we present the dataset used in this paper and describe the SDN traffic features it contains, whereas in Section 5 we describe the architecture and technical details of the neural network used in the experiments. Finally, Section 6 shows our experimental results and Section 7 comments them, describing also our future work in this field.

2 Related work

As discussed in the introduction, to the best of our knowledge there is no work in the literature that focuses on our exact combination of technologies and aims. However, there are works in the recent literature focusing on botnet detection or, more in general, malware analysis in SDN networks and machine learning-based malware detection. Here we give an overview of the most recent and interesting ones.

A few works address the malware detection problem in SDN, possibly exploiting machine learning techniques. Among these, in Tang et al. (2016), the authors propose a deep learning-based approach for general network intrusion detection in SDN using self-taught learning (STL) and the NSL-KDD dataset. In particular, they use a simple five-layer deep neural network with an input dimension of six and an output dimension of two, which is instructed trough six features: *duration*, *protocol type*, *source bytes*, *destination bytes*, *count*, and *service count*. Even if the approach is interesting and similar to ours, the current results are not particularly good, probably due to the insufficient number of features addressed.

In Tariq and Baig (2017), the authors present a SDN-oriented botnet grading strategy based on the collection of OpenFlow *flow counters* and on the use of a monitored C4.5 tree grading algorithm. The experiments show that such features are suitable for distinguishing between malware and general network traffic flows.

In Jankowski and Amanowicz (2015), the authors employ self-organising maps as classifiers in order to build an intrusion prevention system for DDoS attacks mitigation in SDN architectures.

With respect to the approaches above, in this paper we make use of a multilayer perceptron as classifier and a different set of selected SDN traffic features to perform botnet detection.

Finally, in Wijesinghe et al. (2015), the authors propose a machine learning technique to detect botnets by analysing traffic flows directly from SDN switches through the IPFIX protocol (Quittek et al., 2004), thus bypassing the data plane isolation and the OpenFlow protocol. This approach, however, introduces a significant delay in the botnet detection, since measurements extracted with the IPFIX protocol are available only when the corresponding connection is closed. Moreover, the authors apply five different machine learning algorithms to classify the traffic but do not take in consideration their performance. To avoid these problems, in the present work we used only features that can be read from the SDN controller and collected through the OpenFlow protocol.

On the other hand, many works exploiting machine learning methods for malware detection (or, more in general, network traffic classification) can be cited. Kalaivani and Vijaya (2016) compare several different machine learning algorithms such as SVM, naive Bayes, decision trees and neural networks in the context of network traffic classification. They rely on the CTU-13 dataset and evaluate the model performance through 10-fold cross-validation. Surprisingly, they conclude that neural networks are the worst classifiers in this context, whereas the results presented in this paper show that neural networks can be employed in botnet detection with very good results. Our feeling is that Kalaivani and Vijaya (2016), being aimed to a comparison rather than to the optimisation of a specific approach, lacks an accurate feature selection, which is indeed at the heart of a good neural network classifier.

In Wang et al. (2017), the authors encode traffic data as images and then use a convolutional neural network for traffic classification. Their experiments, carried out in two different scenarios with three classifiers, reach an accuracy rate of roughly 99%.

In Van et al. (2017), the authors apply deep learning techniques in an intrusion detection system and experiment it on the KDDCup99 dataset (Dheeru and Karra Taniskidou, 2018) with good accuracy results.

Finally, in Winter et al. (2011), the authors propose a novel *flow-based* inductive intrusion detection system using a one-class support vector machines and reaching an accuracy rate of 98%.

However, for these works, the addressed problem is much more general (not botnet-specific), not SDN-specific, or relies on outdated datasets.

3 Background

In this section, we introduce some background notions useful to better understand the technique presented in this paper.

3.1 Botnets

As recalled in the introduction, botnets are nowadays the most common and harmful attack that may affect CPS in the IoT.

Generally speaking, the term *botnet* refers to a network infected and remotely controlled by a hacker. Botnets are created by a malware infecting individual devices that become ‘bots’ or ‘zombies’: a small botnet may be made up of hundreds or thousands of bots, while a large botnet can reach several millions of bots.

The most common way to use a botnet is to perform DDoS attacks. These attacks use the devices computational resources and available bandwidth to direct a large amount of traffic to a specific site with the intention of overloading it. Therefore, there are different types of DDoS attacks, but the goal is always the same: to collapse a site. In particular, DDoS attacks targeting *critical systems* can be extremely dangerous: as an example, back in 2016, the Mirai botnet attack on a widely used DNS provider made it difficult or impossible to access the internet from many US regions for several hours, and experts suppose that attacks of this kind could be used to bring down an entire nation's power or transportation grid. Moreover, botnets are also used to perform many other illegal or malicious actions. As an example, spammers use botnets to send e-mails.

Botnets have existed on the internet for many years, and their typical targets in this scenario are vulnerable PCs or servers. Examples of recently discovered botnets of this kind include Conficker (Shin and Gu, 2010), Zeus (Gañán et al., 2015), Waledac, Mariposa and Kelihos. Actually, the malware used to drive these botnets is often made public (or sold) by its creator, thus there are dozens of separate botnets that use the same malware and operate at the same time.

In the IoT, smart devices infected by a malware that makes them part of a botnet are also known as *thingbots*. In the last years, many thingbot botnets have been created, sometimes with disastrous effects. The most recent examples are the already mentioned Mirai, Persirai, Bashlite, and Carna (Kolias et al., 2017).

Botnets, especially when they employ a large number of cyber-physical devices as thingbots, which usually cannot be easily disconnected or updated as a computer, can be very hard to detect and eradicate. Furthermore, a botnet can be used in combination with other attacks to build the so-called advanced persistent threats.

3.2 Software-defined networking

SDN is being widely adopted in IoT networks and, in general, it is considered the next-generation networking paradigm. In a SDN, network control is decoupled from forwarding and centralised in software-based SDN controllers, which have a global view of the network. This enables the network infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity. Moreover, SDN architectures have APIs that make a quick and easy setup for common network services, as well manage the entire network through intelligent orchestration (Open Networking Foundation, 2013).

The SDN controller summarises the network state for applications and translates application requirements to low-level rules. Therefore, control decisions are made on a global view of the network, rather than distributed across isolated network appliances. The main communication channel between the SDN control plane and the data plane elements is the OpenFlow protocol (Open Networking Foundation, 2012), although other lower-level, proprietary protocols can be also deployed in specific SDN. In particular, the controller uses OpenFlow calls to manipulate the *flow tables* of the SDN data plane elements (i.e., switches and routers), in order to route each data flow (i.e., a group of packets belonging to the same time interval and sharing specific features such as the source and destination addresses) submitted to the SDN. Moreover, the controller can send a *flow statistics* request to data plane elements in order to read the number of packets/bytes contained in the flow (*received packages/bytes*) and the elapsed time since the flow was installed in the flow table (*flow duration*) (Open Networking Foundation, 2012).

From a cybersecurity perspective, traditional security mechanisms used on the internet, like firewalls and intrusion detection systems, are no longer enough to guarantee the security of the IoT architecture (Olivier et al., 2015). On the other hand, SDN provides new, powerful tools to handle security in this context:

- *Dynamic flow control:* in a SDN, the network data flow can be dynamically reconfigured. In his way, once malicious network traffic is detected, it is possible to quickly and easily split it from authorised traffic.
- *Integrated control:* in a SDN, the controller has an overall visibility on the status of the network and its traffic, making security monitoring easier.
- *Network programmability:* the SDN control plane can be programmed through standard application programming interfaces, making network security development (both detection and blocking) easier to setup.
- *Simple data packets:* being divided from the control plane, the SDN data plane, and therefore the data packets, are simpler to extend, e.g., in order to add security extensions.

4 Dataset and feature selection

Ideally, a good dataset should provide a variety of realistic samples. However, this often implies a knowledge extraction process from complex, inhomogeneous devices, which is not easy to accomplish, especially in the IoT (see, e.g., Colace et al., 2018; Clarizia et al., 2018). Moreover, the resulting datasets are often *biased* due to the specific experiment they were initially aimed at, and therefore they do not constitute a good source of knowledge for a machine learning process. In particular, finding a publicly available dataset which includes realistic samples of both botnet attacks and legitimate traffic is a challenging task and a true concern in the network security community with respect to reproducibility and comparability.

Table 1 SDN traffic features extracted from the controller

#	*Feature*	*SDN feature*	*HogZilla feature*
1	Length of the connection	Duration	Flow duration
2	Maximum expire time of a flow	Hard time out	Flow use time
3	Protocol type	Protocol	Protocol
4	Data bytes in bidirectional flow	Bytes count	Bytes
5	Packets in bidirectional flow	Packets	Packets
6	Data bytes from source to destination	Tx packets	src2dst packets
7	Data bytes from destination to source	Rx packets	dst2src packets
8	Flow permanence time	Idle time out	flow idle time

Table 2 Derived SDN traffic features

#	*Feature*	*HogZilla feature*
9	Packet per second from source to destination	Packet rate (src2dst)
10	Packet per second from destination to source	Packet rate (dst2src)
11, 12, 13, 14	Inter arrival time (min, avg, max, std)	Inter time
15, 16, 17, 18	Inter arrival time from source to destination (min, avg, max, std)	Inter time (src2dst)
19, 20, 21, 22	Inter arrival time from destination to source (min, avg, max, std)	Inter time (dst2src)

4.1 The fair dataset

The *HogZilla* dataset (Resende and Drummond, 2018) is a recently published dataset obtained by merging a fragment of the well-known CTU-13 dataset (García et al., 2014), which contains 13 network traces of seven distinct botnet malwares captured in the CTU University network, Czech Republic, and a large set of normal/background traffic traces taken from the ISCX 2012 IDS dataset (Shiravi et al., 2012), also captured in a real network. This traffic is further pre-processed and classified, resulting in $990,000$ samples, each associated with 192 behavioural features.

To support our experiments, we refine the dataset above, that we shall call the *complete dataset*, by extracting from it a *fair dataset*, i.e., a balanced dataset containing exactly the same number (180,000) of normal and botnet samples. This dataset is used for the neural network training phase, since the complete dataset exhibits much more normal traffic than malicious traffic, and this could bias the neural network making it *learn* only the normal traffic (Kotsiantis et al., 2005).

Note that the traffic samples above are not strictly related to CPS, but are much more general since they were captured in a traditional network. Thus, we are confident that this choice will allow us to build an effective botnet detection system that could also be exploited in more general scenarios.

4.2 SDN traffic features

Feature selection aims to devise the minimal subset of flow features that are enough to detect the behaviour of a botnet. This is especially useful for machine learning-based behavioural analysis, where too many (useless) features to take into account may confuse the classifier. In this paper, we focused on 22 features, taking into account the ones more suitable for botnet detection that can be actually collected in a SDN. Indeed, in a SDN, only statistics like the *number of packets*, the *number of bytes*, and the *flow duration* can be read from the controller, which obtains it through the OpenFlow calls, as discussed in Section 3. However, other useful features can be manually computed from the statistics exposed by the controller.

The eight features from the HogZilla dataset shown in Table 1 can be directly obtained from the SDN controller through appropriate calls. In particular, all the features are read from the *flow statistics* obtained by an OpenFlow call, except for the *protocol*, which is the result of an OVSDB (Tang et al., 2016) call executed by the controller. In the table, we show both the SDN feature name and the corresponding HogZilla feature. For instance, the *protocol* feature (e.g., tcp, udp, icmp, etc.) is useful for botnet detection since bots usually employ different protocols in different phases of their life-cycle (setup, attack, update, etc.).

On the other hand, the other fourteen HogZilla features used in our experimentation and described in Table 2 represent different statistics based on the *packet per second* (PPs) and *inter arrival time* (IAT) features, which can be suitably derived in a SDN environment as follows:

- $PPs = \dfrac{number\ of\ packets}{duration}$
- $IAT = \dfrac{Start\ time\ of\ flow\ (n)}{Start\ time\ of\ flow\ (n-1)}$

In particular, we use the PPs and IAT features relative to the packets exchanged in both directions as well as those travelling in a specific direction (source to destination and destination to source). As an example, the bidirectional variant of the *packet per second* is indeed the *packets* feature in Table 1. Having direction-specific features is useful to detect particular botnets since, for example, most of spam botnets generate unidirectional flows.

Moreover, we consider the minimum, maximum, average and standard deviation of such values. Indeed, bot traffic tends to be uniformly distributed over time but, as a bot progresses through its life-cycle and accomplishes different tasks, the characteristics of the generated traffic vary. The use of such statistic measures helps to capture the behavioural differences between these tasks, so that the overall bot behaviour can be detected.

5 Neural network setup

Neural networks have been widely used in classification tasks, since they have a high tolerance of noisy data and can be used when there is a little knowledge of the relationships between features and classes. This is very important in the botnet detection scenario, since bots are continuously evolving and thus their behaviour quickly changes. This makes deterministic classifiers (e.g., signature scanners) not suitable for such a task.

In our experiments, we used the multilayer perceptron sketched in Figure 1 with the following characteristics:

- The *input layer* consists of 22 neurons, as the number of features selected.
- There are seven *hidden layers* with a specific distribution of neurons: 44, 88, 176, 88, 44, 22, 11. Such neurons are activated using the *rectified linear unit* (ReLU) (Glorot et al., 2011) function, whereas the more common *sigmoid* function has not been employed due to its known *vanishing gradient* issue, which proved to considerably slow down the training in our first experiments.
- Hidden layers are interleaved with *dropout layers*. This kind of layer does not actually contribute to the network capabilities, but is used during training to randomly set a given percentage of its inputs (which are then directly fed to the subsequent hidden layer) to zero. We will discuss the effect of this dropout later in the experiments.
- The *output layer* consists of two neurons with a *softmax* activation function. Indeed, softmax is commonly used in the last layer of neural networks in order to transform the results into *classes*. In our case, the possible output values are 10 for botnet traffic and 01 for normal traffic, whereas the two other values are not considered.

Note that the shape of the neural network, which initially grows by duplicating the number of neurons in each subsequent hidden layer up to the half, and then reduces by halving them, is quite usual (Vincent et al., 2010), whereas the number hidden layers required by our application has been devised with a set or experiments that, for sake of brevity, we do not report here.

We used *cross entropy* as the loss function, e.g., the mean squared error between the predictions and the true values. This choice makes more evident the situations where the predictions are far from the true values. In each experiment, weights were randomly initialised according to a *Gaussian distribution* with zero mean and small (0.05) standard deviation (Bengio, 2012).

5.1 Implementation platform

To run our neural network we used TensorFlow 1.7.0 (Abadi et al., 2015), an open source software library, originally developed by the Google Brain team, which offers a state-of-the-art support for machine learning algorithms. Its main advantages are being cross-platform and able to transparently exploit both GPUs and CPUs to achieve the best computation performances. TensorFlow exposes a low-level Python frontend, upon which sits the so-called Layers API, a mid-level API providing a simpler interface to define the most common neural network layers, i.e., dense and convolutional. In our setup, TensorFlow was installed in an Ubuntu Server 16.04.3 machine with 16 GB DDR4 RAM and an Intel Core I7 CPU.

Figure 1 Structure of the employed perceptron

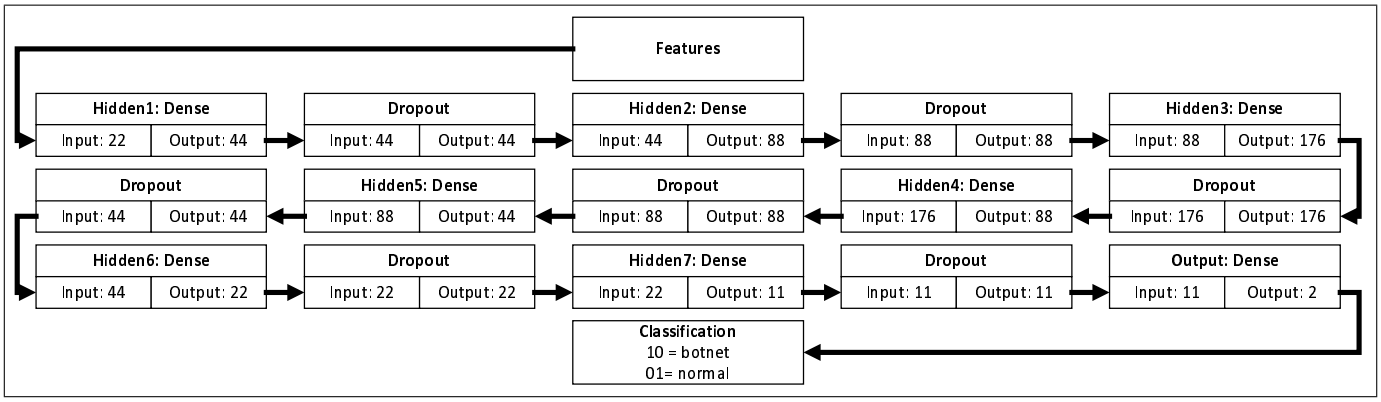


Figure 2 Perceptron definition with Keras

```
model = Sequential()
model.add(Dense(44, input_dim=22, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(88, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(176, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(88, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(44, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(22, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(11, kernel_initialiser='normal', activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2, kernel_initialiser='random_normal', activation='softmax'))

model.compile(loss='binary_crossentropy',
optimiser= keras.optimisers.Adam(lr=0.001),
metrics=['accuracy'])
```

However, to quickly prototype the neural network we did not rely directly on TensorFlow, but on the neural networks API of Keras (Chollet et al., 2018), which is also written in Python and capable of running on top of TensorFlow (as well as on CNTK (Seide and Agarwal, 2016) and Theano (Theano Development Team, 2016)). This allowed us to abstract from the actual underlying engine and rapidly develop our network with a very compact code, which is shown in Figure 2. In the code it is possible to read, for each layer, the number of neurons, the weight initialiser function (`kernel_initialiser`, except for the input layer) and the `activation` function (*relu* or *softmax*). All the layers are *dense* and concatenated with a further *dropout* layer with a dropout percentage of 30%. The chosen `loss` function and `optimiser` are specified in the last code line.

Finally, we used the *scikit-learn* (Pedregosa et al., 2011) and *numpy* (Oliphant, 2018) libraries to support the calculations needed to derive the neural network performance metrics. Thanks to the chosen technology setup above, we are confident that our experimentation results are reliable and easily reproducible.

6 Experimentation

In this section, we report the results obtained by applying the neural network defined in Section 5 to the fair dataset described in Section 4 to detect botnet traffic. We also discuss the actual neural network configuration used, and show the impact of alternative settings.

6.1 Experiment setup

The performance of a neural network can be strongly influenced by the right choice of its configuration parameters. Indeed, starting from the base architecture reported in Section 5, we tried several different parameters, i.e., learning rates, training epochs and batch sizes. We also tried different *optimisation algorithms* for *error backpropagation*. For the sake of simplicity, in this section we report the best results obtained with the Adam optimiser (Kingma and Ba, 2014), setting the *learning rate* to 0.001, the *batch size* to 100 and the number of *epochs* to 10. At the end of this section we will report further experiments to show that other parameter choices cannot reach better results.

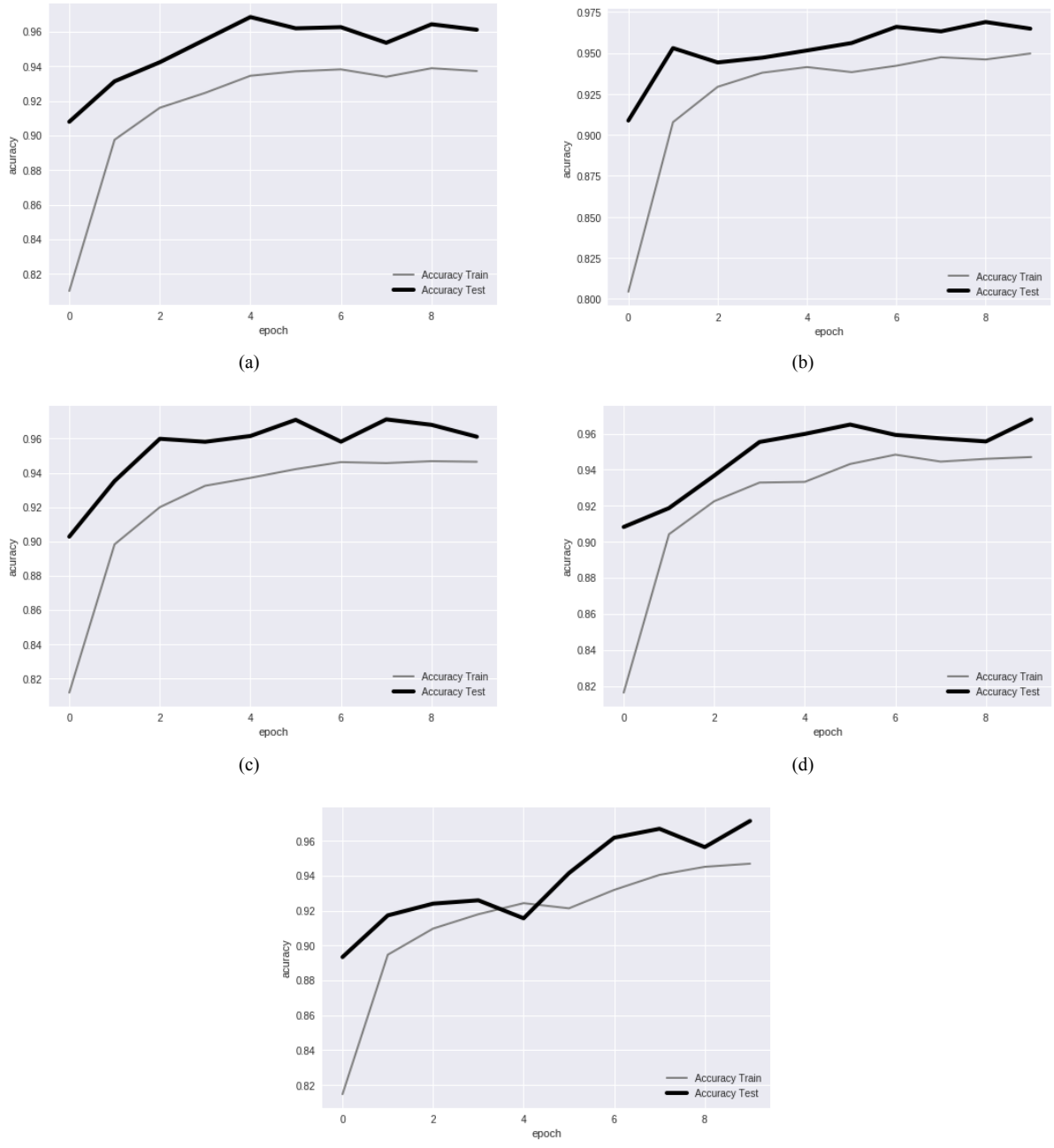
To test our approach we perform five experiments, extracting five different training and test sets from the *fair* dataset (see Section 4). To this aim, we shuffle the dataset and then partition it in 50%/30%/20% parts. The first and second parts will be used for *training* and *testing*, respectively, whereas the remaining 20% is discarded.

6.2 Experimental results

In each of our five experiments, we first run the *fit* Keras method to train the network with the corresponding training set. Normally, after each training epoch, Keras reports the prediction accuracy reached on the training set, but we instructed the library to calculate the accuracy also against the corresponding testing set, which contains completely unknown samples.

The light lines in Figure 3 show the accuracy increase during the training epochs for each experiment, as reported by Keras. We can see that the accuracy quickly increases in all the experiments, reaching or exceeding 93% in the last epoch. Note that, in all the five experiments, the training took less than one minute to conclude. Therefore, our classifier could be synthesised and updated almost in real time.

Figure 3 Neural network accuracy trends during the ten training epochs, (a) experiment #1 (b) experiment #2 (c) experiment #3 (d) experiment #4 (e) experiment #5



(a) (b) (c) (d) (e)

On the other hand, the bold lines in Figure 3 show the accuracy trends for the testing set prediction. In the 5th experiment, the accuracy exceeds 96%. It is worth noting the testing accuracy almost always exceeds the training one: this is a desired effect, obtained by using a suitable dropout (30%) in the network during the training phase. Roughly speaking, the dropout makes the network randomly ignore some neurons during the training: this *regularises* the network by lowering the neuron inter-dependency and shaping the network as a collection of nearly independent classifiers which have a low accuracy on their own, but get an higher accuracy when merged [in some sense they become an ensemble of classifiers (Rokach, 2010)]. The overall effect is to make the network more robust and avoid *overfitting* (Srivastava et al., 2014), i.e., a too tight adaptation to the training patterns, which would make it less general and therefore less accurate on unknown patterns (i.e., during the prediction phase). Overfitting can be visually detected, e.g., from the accuracy trend: when a network is overfitting the training accuracy increases and the test one decreases, whereas in our case they are almost parallel, except in an anomalous intermediate epoch of experiment 5.

Table 3 Neural network training and testing metrics after 10 epochs on different sets extracted from the *fair* dataset

Experiment #		*Loss*	*Accuracy*	*Recall*	*Precision*	*M. correlation*
1	Training	12.40	96.10	93.86	98.27	0.9229
	Testing	12.27	96.10	93.87	98.24	0.9229
2	Training	9.50	96.61	96.39	96.81	0.9322
	Testing	9.79	96.48	96.26	96.66	0.9296
3	Training	14.50	93.93	89.45	98.27	0.8823
	Testing	14.44	94.04	89.65	98.26	0.8841
4	Training	10.14	96.84	94.68	98.94	0.9376
	Testing	10.26	96.79	94.69	98.86	0.9367
5	Training	11.43	97.15	95.37	98.90	0.9436
	Testing	11.91	97.14	95.43	98.80	0.9434

Figure 4 Performance metrics calculation on the trained model

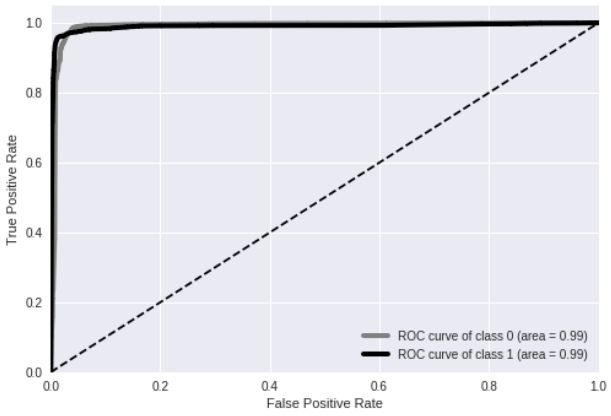
```
predictScore = model.predict_proba(X_Set, batch_size=100)
conf_mat = confusion_matrix(Y_Set, predictScore)
tn = conf_mat[0][0]
fp = conf_mat[0][1]
fn = conf_mat[1][0]
tp = conf_mat[1][1]

accuracy = (tp + tn)/ (tp + tn + fp + fn)
recall = tp / (tp + fn)
precision = tp / (tp + fp)
matthewsCorrelation = ((tp * tn) - (fp * fn)) / math.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
```

However, the Keras history object (used to build the graphs in Figure 3) does not contain the actual loss/accuracy values, but just a mean over all the training batches, as the weights change with each batch (Valdenegro, 2017), thus the values reported in Figure 3 may not be completely correct, but only useful to show the accuracy trend during the training epochs.

Therefore, after the last training epoch, we performed a further evaluation of the final network against the training and testing sets, and Table 3 reports the final performance metrics for the five experiments. In particular, we report the most common metrics used to evaluate neural networks: *loss*, *accuracy*, *recall* and *precision*. We also report the *Matthews correlation coefficient* (Matthews, 1975) that, for binary classifications like ours, gives a clear estimate of the classifier quality. It is worth noting that such metrics (excluding the loss, which is taken from the output of the *evaluate* Keras method) have been manually calculated by comparing the output of the *predict_proba* Keras method with the expected classification results through the code shown in Figure 4. In this way, we have the full control over the calculations and their precision.

We can see that, in the testing phase, the fifth experiment actually gets an accuracy of 97.14% with a loss of 11.91%. The Matthews correlation and the recall are the highest among all the experiments, although the fourth experiment has a slightly higher precision. Thus, the fifth training set is the best one. This is confirmed also by the receiver operating characteristic (ROC) curves plotted in Figure 5 for this dataset: such kind of curve relates the true positive rate (i.e., the recall) with the false positive rate for increasing thresholds of the decision function (which maps the numerical network output to a traffic class). In particular, Figure 5 shows the ROC curve for both traffic classifications (normal and botnet). The curves first go from the bottom left of the plot to nearly the top left, and then continue straight to the right end: this represents a model with nearly perfect skill (Hanley and McNeil, 1982).

Figure 5 Receiver operating characteristic (ROC) curve for the fifth experiment



However, it is worth noting that all the other experiments also report very satisfying results, confirming that the chosen network shape and parameters as well as the selected traffic features are correct.

Table 4 Neural network testing on the *complete* dataset

Experiment #	*Accuracy*
1	92.24
2	82.55
3	93.18
4	92.41
5	92.63

Finally, we tested the accuracy of the five trained networks also against the *complete* HogZilla dataset obtaining an accuracy that ranges from 82.55% to 93.18%, as reported in Table 4. The lowest value belongs to the second training set, probably because the training samples had a low variance and thus contained less information. The best accuracy on the complete dataset comes from the third training set, followed by the fifth one. In general, we note an accuracy decrease, but the overall performance remains good considering that the network was trained on the 50% of the fair dataset, thus approximately $180,000$ samples, and then tested on the compete dataset of $990,000$ samples.

6.3 Impact of network parameters

As stated at the beginning of this section, network configuration parameters may strongly affect experimental results, and the values employed in the experiments above have been devised through further experimentation, where different choices were tested. To validate our final setup, and show how different choices actually impact the network accuracy, here we run further experiments with different parameters on the fifth training/testing set, which resulted to have the best performances. Note that, for sake of simplicity, here we report only the accuracy metric for each experiment.

Table 5 Accuracy metrics for different optimisers

Optimiser	*Training accuracy*	*Testing accuracy*
Adamax	93.41	93.43
AdaDelta	92.22	92.38
AdaGrad	68.80	68.25
Nadam	49.94	50.14
RMSprop	49.92	50.18
SGD	50.07	49.86
Adam	**96.83**	**97.00**

First we try to change the network optimiser: in particular, we use the *stochastic gradient descent (SGD)* optimisation algorithm as well as its variants *RMSprop* (Hinton, 2018), *Adadelta* (Zeiler, 2012), *AdaGrad* (Duchi et al., 2011) and *Adam* (Kingma and Ba, 2014) with the corresponding further variants *Adamax* and *Nadam*. Note that the stochastic nature of such algorithms helps to converge in a lower number of epochs on datasets with redundant data, as traffic flows are, and gives better possibility of escaping from local minima (Bottou, 1991). The results are shown in Table 5. As we can see, Adam remains the best choice in our case. Note that the values reported here (as well as in Tables 6–8) may be slightly different from the ones of Table 3 even when using the same parameters (the Adam optimiser in this case) due to the random weight initialisation that makes each training of the network behave differently, even if the train and test sets are the same.

Table 6 Accuracy metrics for different learning rates

Learning rate	*Training accuracy*	*Testing accuracy*
0.0001	89.48	89.63
0.001	**96.19**	**96.34**
0.01	49.92	50.14
0.1	50.08	50.86

Then we try to change the *learning rate* parameter (also known as the step size), which controls the size of the updates applied to the network weights. Table 6 shows that when the learning rate is higher or lower than 0.001, the accuracy decreases considerably, especially in the testing phase. Therefore, the value 0.001 seems to be the best choice to make the network suitably generalise the notions coming from the training set.

Table 7 Accuracy metrics for different number of epochs

Epochs	*Training accuracy*	*Test accuracy*
10	96.86	96.92
15	96.33	96.31
20	96.28	96.23
25	**96.87**	**96.96**
30	94.03	94.04

We may further try to change the number of epochs used for training, i.e., the number of times the network is exposed to the training set. Table 7 shows that with 25 epochs we get the higher testing accuracy (96.96%). On the other hand, the accuracy obtained with 10 epochs (96.92%) is only slightly less than the maximum and, in this case, the training is significantly faster (less than $1/3$ of the time required by 25 epochs). Thus, we may conclude that 10 epochs is an optimal value to get a good accuracy/performance compromise.

Table 8 Accuracy metrics for different batch sizes

Batch size	*Training accuracy*	*Testing accuracy*
10	88.38	88.63
100	**96.77**	**96.79**
1,000	94.42	94.64
10,000	86.16	86.25

Finally, we try the impact of changing the batch size, i.e., the number of training instances observed before the optimiser performs a weight update. Table 8 shows that the batch size impacts learning significantly. In general, larger batch sizes result in faster training, but do not always get an acceptable accuracy. Smaller batch sizes train slower, but get higher accuracies. Indeed, the right batch size will provide the optimiser with a stable estimate of the gradient to apply. In our case, a batch size of 100 gets the best results.

7 Conclusions

In the IoT, CPS are connected through networks and therefore subject to attacks that, for critical systems, may have disastrous results. In particular, IoT botnets have already proved to be a dangerous threat.

Many research works currently address botnet detection, but their results are often biased by unrealistic datasets. Indeed, public datasets are commonly published in a pre-processed format, where specific features have already been extracted, or in raw format, leaving the user with the hard task of correctly extracting a meaningful set of features.

In both cases, the samples come from standard internet traffic, whereas in some network architectures raw packet data is not available and therefore some features cannot be deduced. This is the case of SDN networks, which are tightly connected to the IoT, where the traffic statistics should be obtained only from the controller, and observing in real time the raw traffic through the data plane is, in general, not allowed.

In this paper we performed botnet detection experiments on a new dataset, containing a very large amount of normal and botnet traffic samples, from which we extracted a set of botnet-specific meaningful features that can be actually derived in a SDN environment. Starting from this realistic basis, we applied behavioural analysis techniques using a multilayer perceptron as the classifier, showing that, with a good selection of parameters, this simple kind of neural network can reach very good accuracy rates, up to 97%.

For sake of transparency, both our dataset and the definition of the employed perception are publicly available (Letteri and Della Penna, 2018), so our experiments can be easily reproduced by other researchers. In particular, in our implementation we used two state-of-the art public libraries, i.e., TensorFlow and Keras, to minimise any possible implementation problem that could invalidate the results and make our models easily executable in any environment.

Our future work on this field will include a more in-depth analysis of the SDN traffic features, to further reduce the amount of data needed to reach the current accuracy levels. To this aim, we are studying the correlations among the current set of features and also plan to employ deep leaning techniques to perform a further feature selection. Moreover, we plan to extend our analysis beyond the IoT, since SDN is increasingly replacing traditional networking in many other contexts.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015) *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* [online] http://www.tensorflow.org (accessed 20 November 2018).

Abu Rajab, M., Zarfoss, J., Monrose, F. and Terzis, A. (2006) 'A multifaceted approach to understanding the botnet phenomenon', *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC '06*, pp.41–52, ACM, New York, NY, USA.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K. and Zhou, Y. (2017) 'Understanding the Mirai botnet', *Proceedings of the 26th USENIX Conference on Security Symposium, SEC'17*, pp.1093–1110, USENIX Association, Berkeley, CA, USA.

Bailey, M., Cooke, E., Jahanian, F., Xu, Y. and Karir, M. (2009) 'A survey of botnet technology and defenses', *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security, CATCH '09*, pp.299–304, IEEE Computer Society, Washington, DC, USA.

Bengio, Y. (2012) 'Practical recommendations for gradient-based training of deep architectures', in Montavon, G., Orr, G.B. and Müller, K-R. (Eds.): *Neural Networks: Tricks of the Trade*, 2nd ed., Vol. 7700 of *Lecture Notes in Computer Science*, pp.437–478, Springer Berlin Heidelberg, Berlin, Heidelberg.

Bottou, L. (1991) 'Stochastic gradient learning in neural networks', *Proceedings of Neuro-Nimes 91*, Nimes, France, EC2.

Chollet, F. et al. (2018) *Keras: The Python Deep Learning Library* [online] http://keras.io (accessed 20 November 2018).

Clarizia, F., Colace, F., Lombardi, M., Pascale, F. and Santaniello, D. (2018) 'A multilevel graph approach for road accidents data interpretation', in Castiglione, A., Pop, F., Ficco, M. and Palmieri, F. (Eds.): *Cyberspace Safety and Security*, Vol. 11161 of *Lecture Notes in Computer Science*, pp.303–316, Springer.

Colace, F., Lombardi, M., Pascale, F., Santaniello, D., Tucker, A. and Villani, P. (2018) 'MuG: a multilevel graph representation for big data interpretation', *20th IEEE International Conference on High Performance Computing and Communications; 16th IEEE International Conference on Smart City; 4th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2018*, 28–30 June, Exeter, UK, pp.1408–1413, IEEE.

D'Angelo, G., Rampone, S. and Palmieri, F. (2015) 'An artificial intelligence-based trust model for pervasive computing', *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp.701–706.

D'Angelo, G., Rampone, S. and Palmieri, F. (2017) 'Developing a trust model for pervasive computing based on apriori association rules learning and Bayesian classification', *Soft Computing*, Vol. 21, No. 21, pp.6297–6315.

Dheeru, D. and Karra Taniskidou, E. (2018) *UCI Machine Learning Repository: KDD Cup 1999 Data Data Set* [online] http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data (accessed 20 November 2018).

Duchi, J., Hazan, E. and Singer, Y. (2011) 'Adaptive subgradient methods for online learning and stochastic optimization', *J. Mach. Learn. Res.*, Vol. 12, pp.2121–2159.

Flauzac, O., González, C., Hachani, A. and Nolot, F. (2015) 'SDN based architecture for IoT and improvement of the security', *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pp.688–693.

Gañán, C., Cetin, O. and van Eeten, M. (2015) 'An empirical analysis of ZeuS C&C lifetime', *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15*, pp.97–108, ACM, New York, NY, USA.

García, S., Grill, M., Stiborek, J. and Zunino, A. (2014) 'An empirical comparison of botnet detection methods', *Computers & Security*, Vol. 45, pp.100–123.

Glorot, X., Bordes, A. and Bengio, Y. (2011) 'Deep sparse rectifier neural networks', *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp.315–323.

Hanley, J.A. and McNeil, B.J. (1982) 'The meaning and use of the area under a receiver operating characteristic (ROC) curve', *Radiology*, Vol. 143, No. 1, pp.29–36.

Hinton, G. (2018) *RMSprop: Divide the Gradient by a Running Average of Its Recent Magnitude (Lecture 6e)* [online] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (accessed 20 November 2018).

Jankowski, D. and Amanowicz, M. (2015) 'Intrusion detection in software defined networks with self-organized maps', *Journal of Telecommunications and Information Technology*, Vol. 2015, No. 4, pp.3–9.

Junejo, K.N. and Goh, J. (2016) 'Behaviour-based attack detection and classification in cyber physical systems using machine learning', *Proceedings of the 2Nd ACM International Workshop on Cyber-Physical System Security, CPSS '16*, pp.34–43, ACM, New York, NY, USA.

Kalaivani, P. and Vijaya, M. (2016) 'Mining based detection of botnet traffic in network flow', *IRACST – International Journal of Computer Science and Information Technology and Security*, Vol. 6, No. 1, pp.535–541. .

Kamal, B., Abdeslam, E.F. and Abdelbaki, E.E. (2016) 'Software-defined networking (SDN): a survey', *Security and Communication Networks*, Vol. 9, No. 18, pp.5803–5833.

Kingma, D.P. and Ba, J. (2014) 'Adam: a method for stochastic optimization', *CoRR*, arxiv.org/abs/1412.6980.

Kolias, C., Kambourakis, G., Stavrou, A. and Voas, J. (2017) 'DDoS in the IoT: Mirai and other botnets', *Computer*, Vol. 50, No. 7, pp.80–84.

Kotsiantis, S., Kanellopoulos, D. and Pintelas, P. (2005) 'Handling imbalanced datasets: a review', *GESTS International Transactions on Computer Science and Engineering*, Vol. 30, pp.25–36.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', *Nature*, Vol. 521, No. 7553, p.436.

Letteri, I., Del Rosso, M., Caianiello, P. and Cassioli, D. (2018) 'Performance of botnet detection by neural networks in software-defined networks', *Proceedings of the Second Italian Conference on Cyber Security*, 6–9 February, Milan, Italy.

Letteri, I. and Della Penna, G. (2018) *Sources for Botnet Detection Experiments on SDN Networks through Machine Learning Techinques* [online] http://github.com/IvanLetteri/BotNet-SDN-ML.

Matthews, B. (1975) 'Comparison of the predicted and observed secondary structure of T4 phage lysozyme', *Biochimica et Biophysica Acta (BBA) – Protein Structure*, Vol. 405, No. 2, pp.442–451.

Miller, S. and Busby-Earle, C. (2016) 'The role of machine learning in botnet detection', *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*.

National Science Foundation (2018) *Cyber-Physical Systems (CPS)* [online] http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf18538&org=NSF (accessed 20 November 2018).

Oliphant, T. (2018) *Numpy* [online] http://www.numpy.org (accessed 20 November 2018).

Olivier, F., Carlos, G. and Florent, N. (2015) 'New security architecture for IoT network', *The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), The 5th International Conference on Sustainable Energy Information Technology (SEIT-2015)*, *Procedia Computer Science*, Vol. 52, pp.1028–1033.

Open Networking Foundation (2012) *Openflow Switch Specification*, version 1.3.0 [online] http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (accessed 20 November 2018).

Open Networking Foundation (2013) *SDN Architecture Overview*, version 1.0 [online] http://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf (accessed 20 November 2018).

Özçelik, M., Chalabianloo, N. and Gür, G. (2017) 'Software-defined edge defense against IoT-based DDoS', *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pp.308–313.

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

Quittek, J., Zseby, T., Claise, B. and Zander, S. (2004) *Requirements for IP Flow Information Export (IPFIX)* [online] http://tools.ietf.org/html/rfc3917 (accessed 20 November 2018).

Resende, P.A.A. and Drummond, A.C. (2018) *The Hogzilla Dataset* [online] http://ids-hogzilla.org/dataset (accessed 20 November 2018).

Rokach, L. (2010) 'Ensemble-based classifiers', *Artificial Intelligence Review*, Vol. 33, No. 1, pp.1–39.

Seide, F. and Agarwal, A. (2016) 'CNTK: Microsoft's open-source deep-learning toolkit', *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pp.2135–2135, ACM, New York, NY, USA.

Shin, S. and Gu, G. (2010) 'Conficker and beyond: a large-scale empirical study', *Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10*, pp.151–160, ACM, New York, NY, USA.

Shiravi, A., Shiravi, H., Tavallaee, M. and Ghorbani, A.A. (2012) 'Toward developing a systematic approach to generate benchmark datasets for intrusion detection', *Computers & Security*, Vol. 31, No. 3, pp.357–374.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) 'Dropout: a simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, Vol. 15, pp.1929–1958.

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', *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*.

Tanwar, G.S. and Goar, V. (2014) 'Tools, techniques & analysis of botnet', *Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies, ICTCS '14*, pp.92:1–92:5, ACM, New York, NY, USA.

Tariq, F. and Baig, S. (2017) 'Machine learning based botnet detection in software defined networks', *International Journal of Security and Its Applications*, Vol. 11, No. 11, pp.1–12.

Theano Development Team (2016) 'Theano: a Python framework for fast computation of mathematical expressions', *arXiv e-prints*, abs/1605.02688.

Valdenegro, M. (2017) *Stackoverflow Answer: Why is Accuracy Different between Keras model.fit and model.evaluate?* [online] http://stackoverflow.com/questions/48041867/why-is-accuracy-different-between-keras-model-fit-and-model-evaluate (accessed 20 November 2018).

Van, N.T., Thinh, T.N. and Sach, L.T. (2017) 'An anomaly-based network intrusion detection system using deep learning', *2017 International Conference on System Science and Engineering (ICSSE)*, pp.210–214.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P-A. (2010) 'Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion', *Journal of Machine Learning Research*, December, Vol. 11, pp.3371–3408.

Wang, W., Zhu, M., Zeng, X., Ye, X. and Sheng, Y. (2017) 'Malware traffic classification using convolutional neural network for representation learning', *2017 International Conference on Information Networking (ICOIN)*, pp.712–717.

Wijesinghe, U., Tupakula, U. and Varadharajan, V. (2015) 'Botnet detection using software defined networking', *2015 22nd International Conference on Telecommunications (ICT)*, pp.219–224.

Winter, P., Hermann, E. and Zeilinger, M. (2011) 'Inductive intrusion detection in flow-based network data using one-class support vector machines', *2011 4th IFIP International Conference on New Technologies, Mobility and Security*.

Yassein, M.B., Abuein, Q. and Alasal, S.A. (2017) 'Combining software-defined networking with internet of things: survey on security and performance aspects', *2017 International Conference on Engineering MIS (ICEMIS)*, pp.1–7.

Zeiler, M.D. (2012) 'ADADELTA: an adaptive learning rate method', *CoRR*, abs/1212.5701.

Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A. and Garant, D. (2013) 'Botnet detection based on traffic behavior analysis and flow intervals', *27th IFIP International Information Security Conference*, *Computers & Security*, Vol. 39, pp.2–16.