IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON CYBER-PHYSICAL-SOCIAL COMPUTING AND NETWORKING

Received December 18, 2017, accepted January 23, 2018, date of publication February 7, 2018, date of current version March 12, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2803446

A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks

LORENZO FERNÁNDEZ MAIMÓ[1], ÁNGEL LUIS PERALES GÓMEZ[1], FÉLIX J. GARCÍA CLEMENTE[1], MANUEL GIL PÉREZ[2], AND GREGORIO MARTÍNEZ PÉREZ[2], (Member, IEEE)

[1]Departamento de Ingeniería y Tecnología de Computadores, University of Murcia, 30100 Murcia, Spain
[2]Departamento de Ingeniería de la Información y las Comunicaciones, University of Murcia, 30100 Murcia, Spain

Corresponding author: Lorenzo Fernández Maimó (lfmaimo@um.es)

This work was supported in part by the European Commission Horizon 2020 Programme under Grant H2020-ICT-2014-2/671672-SELFNET (Framework for Self-Organized Network Management in Virtualized and Software-Defined Networks), in part by the Spanish MICINN (Project DHARMA, Dynamic Heterogeneous Threats Risk Management and Assessment) under Grant TIN2014-59023-C2-1-R, and in part by the European Commission (FEDER/ERDF).

ABSTRACT The upcoming fifth-generation (5G) mobile technology, which includes advanced communication features, is posing new challenges on cybersecurity defense systems. Although innovative approaches have evolved in the last few years, 5G will make existing intrusion detection and defense procedures become obsolete, in case they are not adapted accordingly. In this sense, this paper proposes a novel 5G-oriented cyberdefense architecture to identify cyberthreats in 5G mobile networks efficient and quickly enough. For this, our architecture uses deep learning techniques to analyze network traffic by extracting features from network flows. Moreover, our proposal allows adapting, automatically, the configuration of the cyberdefense architecture in order to manage traffic fluctuation, aiming both to optimize the computing resources needed in each particular moment and to fine tune the behavior and the performance of analysis and detection processes. Experiments using a well-known botnet data set depict how a neural network model reaches a sufficient classification accuracy in our anomaly detection system. Extended experiments using diverse deep learning solutions analyze and determine their suitability and performance for different network traffic loads. The experimental results show how our architecture can self-adapt the anomaly detection system based on the volume of network flows gathered from 5G subscribers' user equipments in real-time and optimizing the resource consumption.

INDEX TERMS 5G, anomaly detection, botnets, deep learning, performance evaluation.

I. INTRODUCTION AND MOTIVATION

Cybersecurity researchers and professionals have designed and developed over the years a number of cyberdefense systems to protect assets of organizations from malicious attackers. These systems address cybersecurity threats such as viruses, Trojans, worms, and botnets, among others [1]. Existing solutions based on Intrusion Detection Systems (IDS) include (pro-)active approaches to anticipate and remove vulnerabilities in computing systems with which to trigger reactive actions for mitigation.

Any protection mechanism needs to operate by integrating algorithms with good and precise detection capabilities, allowing quick processing of the data gathered by the information sources. Without these capabilities, IDSs cannot perform their monitoring and analysis functions in real time, making it almost impossible to detect potential cyberattacks when they are starting to happen. This problem is due to the fact that current networks provide increasingly high transmission rates. More specifically, the rates have increased from 100 Mbps a few years ago to the current data rate of 10+ Gbps in wired networks.

Large volumes of information flowing through networks make IDSs ineffective to gather and analyze every network packet. As an example, Deep Packet Inspection (DPI) tools like Snort [2] can work properly on wired networks up to 1 Gbps, starting to discard packets due to overhead from 1.5 Gbps [3]. A recent study [4] conducted intensive experiments to extract a thorough performance comparison by using Snort and the application of machine learning techniques on it, evaluating such IDS to process network traffic up to 10 Gbps network speed. These experiments demonstrate that the average packets drop when using Snort reaches 9.5% in

4 Gbps networks while the average packets drop with 10 Gbps networks rises to 20%. In order to improve performance, several studies have focused on advanced parallelization techniques based on hardware accelerators. Among them, techniques based on Field Programmable Gate Array (FPGA) support speeds of up to 4 Gbps without loss [5] while the ones based on Application-Specific Integrated Circuit (ASIC) reach speeds close to 7.2 Gbps [6].

However, and due to the increase of bandwidth, IDS-based solutions making use of deep analysis techniques were forced to evolve towards new ways of detection. They moved from inspecting raw network packets to analyzing traffic network flows with innovative AI-based techniques [7]. For example, a Block-Based Neural Network (BBNN) for an anomaly-based IDS achieved around 22 Gbps throughput by using an FPGA architecture [8]. A complete survey dealing with solutions to quickly classify collected network flows and detect attacks, or malicious code, can be found in [9]. Nonetheless, these solutions appear to be insufficient for the Future Internet envisaged for the coming years, since networks with even higher transmission rates are expected. Currently, a huge R&D effort is being made to create and deploy the new upcoming fifth generation (5G) mobile technology. The new advanced features of 5G will make existing detection procedures become obsolete quite easily in case they are not adapted accordingly to the new requirements.

The European 5G-PPP consortium has identified a pool of Key Performance Indicators (KPI) [10] that have a high impact when analyzing and inspecting traffic network flows. It needs to determine certain characteristics of the incoming network flow in an efficient and quick way so that detection procedures can be successful. Among these KPIs, we highlight below four of them that make detection procedures an even greater challenge in 5G mobile networks:

- 1000 times higher mobile data volume per geographical area.
- 10 to 100 times more connected devices.
- 10 times to 100 times higher typical user data rate.
- End-to-End latency of $<$1ms.

The large number of User Equipments (UE) belonging to 5G subscribers, the large volumes of data traffic produced by them, and the reduced latency in connectivity make us face new challenges to be solved without losing detection accuracy in real-time scenarios.

To overcome this challenge, this article substantially extends the solution proposed in [11], in which a 5G-oriented architecture was presented to identify cyberthreats in 5G networks by making use of deep learning techniques. Preliminary experiments were conducted in [11] to evaluate runtime performance. In this extension, our main contribution consists in extending our architecture with the ability of self-adaptation to manage traffic fluctuation. When required, our system will decide to deploy more computing resources, replace the deep learning framework, or even the detection model, with a more suitable one to the given cyberdefense context. This is made by creating an abstraction layer that uses virtualized frameworks and decides which one will be used depending on the behavior of the traffic. This adaptation can be made seamlessly by means of the virtualized network functions present in our proposal. The decision making of our architecture is based on experimental results that determine the system throughput with the different deep learning frameworks considered. By way of example, this paper presents experiments using a well-known botnet dataset to illustrate how a neural network model reaches a sufficient classification accuracy in our anomaly detection system. Extended experiments with diverse deep-learning frameworks and configurations analyze and determine their suitability and performance for different network traffic load. Finally, using the experimental results, the paper depicts how many network flows gathered from the 5G subscribers' UEs we can inspect in real-time with this new proposed cyberdefense architecture. These experiments show that our proposal is suitable to cope with the evaluation of the traffic in a real 5G scenario, and promising in achieving a sufficient classification performance.

This paper is organized as follows. Section II presents some of the challenges of using machine learning for anomaly detection in networks and a selected set of deep learning models. In Section III, we outline the proposed self-adaptive architecture for inspecting network flows in 5G mobile networks. Section IV details the deep learning-based inspection techniques that we propose to use for anomaly detection. Section V shows the experiments conducted to demonstrate the feasibility of our proposal, while conclusions are drawn and future work is outlined in Section VI.

II. MACHINE LEARNING FOR NETWORK TRAFFIC ANOMALY DETECTION

An anomaly can be defined as a pattern that does not conform to the expected behavior, so it appears infrequently, and it is tightly linked to the concept of normality. Essentially, we can consider as anomalous all the network traffic that does not fit in the normal class. Consequently, anomaly detection systems should not be limited by any predefined set of anomalies; instead, they should be flexible enough to adapt themselves to any unknown event affecting the network. A first straightforward approach could be to define a region representing normal behavior and consider any sample that does not belong to this area as anomalous. However, several factors such as the imprecise boundary between normal and anomalous, or availability of labeled data for training/validation/testing can make this approach very challenging [12].

This section presents the main machine learning challenges in anomaly detection and, moreover, introduces the deep-learning approach towards the anomaly detection.

A. MACHINE LEARNING CHALLENGES IN NETWORK TRAFFIC ANOMALY DETECTION

Machine-learning anomaly detection techniques act as classifiers and operate in one of the following three modes:

- *Supervised*. A dataset with traffic labeled as normal or anomalous is available. These methods use this labeled dataset to find a boundary that separates normal and anomalous traffic.
- *Semi-supervised*. The training set contains only normal traffic, and anything that does not belong to this kind of traffic is considered anomalous.
- *Unsupervised*. No labeled training set is necessary. The goal in this technique may be to discover groups of similar examples within the data (clustering), or to determine the distribution of data within the input space –known as density estimation– or to project high-dimensional data to a lower-dimensional space.

In *supervised* detection, the main issue is how to build a really comprehensive training set with all the anomalous traffic properly labeled. This requires two sets containing anomalous and normal traces. These sets are not usually available, and they can be difficult to collect and maintain [13]. There are several reasons for this, such as the excessive effort needed to collect such data, the level of expert knowledge that would be necessary for the analysis, and even the issues with the users' privacy. In fact, there are different legal aspects in collecting all the traffic from the network of a company or institution.

For illustration purposes, let us take a botnet detection context [14]. The botnet traffic is usually obtained from synthetic environments, that is, by using honeypots, or by executing botnet binaries in a controlled environment. The normal traces are usually collected from an academic or institutional network, so there is a certain probability that these traces may contain unnoticed anomalous traffic, e.g., there might be an unknown previously active botnet in the network. The traces are then combined, and there is no perfect way of doing it. In our example, two possibilities could be either mapping the malicious traffic to hosts not present in the normal traces, or adding it to existing hosts. This may not match with any real traffic pattern. Furthermore, each trace typically contains traffic from a single botnet, so such traces might not represent real networks' traffic, which can involve concurrent infections of different malware on a single host.

When *unsupervised* methods are used for clustering, a question that arises is how to be certain that the identified classes correspond to the desired ones. Even if we have a large amount of data covering all the different scenarios and traffic patterns, we cannot blindly trust the results. However, when used for dimension reduction, these methods are well suited to extract discriminative higher level features that can improve the classification performance of a supervised or semi-supervised algorithm.

Finally, the *semi-supervised* machine learning approach tries to estimate the probability distribution of the normal traffic from a sufficient amount of collected samples. This defines a tight boundary around the region (not necessarily convex) where a sample is classified as normal. The difference with respect to the supervised method is that there is no information about the shape of the anomalous region in the samples space. The new traffic is classified as anomalous in case it exceeds a threshold distance.

B. DEEP LEARNING APPROACH

Deep learning algorithms have achieved state-of-the-art results in a range of difficult problem domains, involving supervised and unsupervised learning. They essentially use the well-known multilayer perceptron to obtain, in an unsupervised way, higher-level representations expressed in terms of other simpler ones. Several types of deep learning neural networks can be found, namely Convolutional Neural Networks (CNN), Deep Belief Networks (DBN), Stacked AutoEncoders (SAE), Long Short-Term Memory Recurrent Networks (LSTM), to name just a few. Each of them is particularly well suited for dealing with a different sort of classification problems. In computer networks, anomaly detection has been a subject of study for decades, and many approaches have been explored [12], [15] while the deep learning perspective has received special attention in recent years [16].

Among the unsupervised learning methods, DBN and SAE have proved to be effective in learning invariant features from complex and high-dimensional datasets. The Restricted Boltzmann Machine (RBM) is the building block of a DBN [17], where each layer is separately trained in a greedy way as a RBM that takes the input from the feature layer learned in the previous layer. SAE use the same idea to train stacked autoencoders in an unsupervised way, one at a time, to obtain a set of more descriptive low-dimension features. Both can be fine-tuned by means of backpropagation or Support Vector Machine (SVM) layers in a supervised way. They can also be configured as a semi-supervised one-class method, for example adding a one-class SVM as a last layer [18], [19]. These semi-supervised one-class algorithms are well suited in anomaly detection, where the set of anomalous traffic captured is usually much smaller than the set of normal traffic. They can also be used in a prior phase to detect background traffic outliers, which could give us some useful insights into the traffic characteristics.

LSTMs, in turn, are deep learning architectures especially well suited to detect/predict complex patterns in time series with time-lags of variable size between events. They can be used to detect anomalies from a temporal pattern of less reliable symptoms. Moreover, LSTM can be used with both supervised and unsupervised learning.

Training deep learning methods is a costly process because they need a great amount of data and iterations to converge. However, these methods usually outperform other classic algorithms. Additionally, they exhibit highly parallel computation patterns in prediction mode that take great advantage of GPU's computing power. As part of our research, we are interested in time-effective solutions that offer sufficient accuracy with a low evaluation runtime, to process the features coming from the large volumes of input information expected in 5G mobile networks.

III. SELF-ADAPTIVE ARCHITECTURE FOR NETWORK INSPECTION IN 5G

5G networks are conceived as extremely flexible infrastructures based on an architecture organized by different functional planes. These planes provide separation of duties to simplify and address a complete set of novel system requirements. Among these planes, we propose a high-level design of the management and orchestration plane, following the ETSI NFV architecture [20], which has also been used as the basis for other proposals [21], [22].

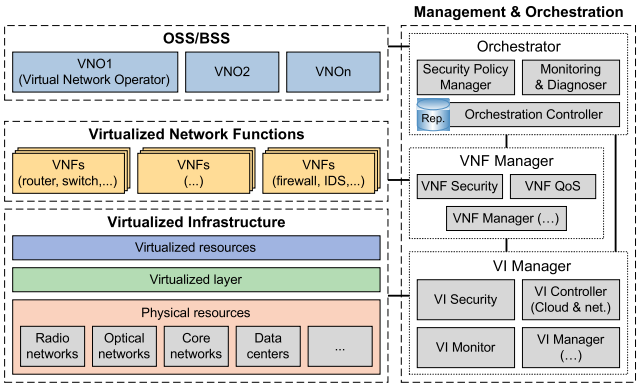

Fig. 1. High-level management and orchestration plane.

The proposed cyberdefense architecture, as shown in Fig. 1, consists of four groups of components: Virtualized Infrastructure (VI); Virtualized Network Functions (VNF); Management and Orchestration (MANO); and Operations and Business Support Systems (OSS/BSS). The VI group virtualizes the physical resources (computing, storage, and network) and exposes them for consumption by VNFs. The MANO group manages the combination of VNFs implementing network services, the full life cycle of VNFs, the deployment of VNFs within virtualized resources, and the network slicing for supporting multi-tenancy. Therefore, MANO controls the general behavior of the infrastructure by considering the policy set defined by Virtual Network Operators (VNO) of the OSS/BSS.

Among these policies, the VNO must define security policies to be applied into the network resources under its control. The Security Policy Manager module as defined in our architecture takes into account these security policies and decides the best actions after evaluating the input provided by the Monitoring and Diagnoser module. This module must process the current monitoring information generated by the network resources and identify the causes from any anomaly detected.

To achieve effective network anomaly detection, we propose a system, depicted in Fig. 2, which is made up of two VNFs: Anomaly Symptom Detection (ASD) and Network Anomaly Detection (NAD). The former is located within the Radio Access Network (RAN) infrastructure. It focuses on the quick search of anomaly symptoms by using inspection of network-flow aggregations. Here, a symptom is any trace or sign of anomaly in the network traffic generated by UEs connected to the RAN. The latter is a collector of timestamped and RAN-associated symptoms, where a central process analyzes the timeline and the relationship among these symptoms to identify any network anomaly. Once an anomaly is produced, it is immediately communicated to the Monitoring and Diagnoser module.

On the other hand, VI monitors provides the Monitoring and Diagnoser module with additional monitoring information, related to the resource health and usage produced by RANs (e.g., CPU and memory usage), and also network flow measures about collecting and analyzing processes. We propose to input performance monitoring indicators and anomaly causes into the Security Policy Manager in order to take the corresponding actions according to VNO security policies automatically. Among the main actions that Security Policy Manager can decide, the following ones aim at the adaptation of the system configuration in order to optimize, when required, either the anomaly detection processes or the usage of network resources involved in the detection process:

- *Adapting RAN resources*. When the overload of any single resource is identified, the actions could deploy new virtualized resources, change current resource configurations, or balance load of flow collectors.
- *Optimizing ASD and NAD functions*. The network traffic fluctuation also implies variations in the volume of network flows. In order to optimize detection processes, maximize throughput, and minimize response time, the actions could replace the deep learning framework or the detection model with another, more suitable one.
- *Extending ASD and NAD functions*. In some cases the deployment of precise detection components may avoid false-positives. In particular, the actions could instantiate DPI mechanisms that permit a deep search into the L2/3 flows.

As DPI-based mechanisms are too resource consuming considering 5G mobile data volume, our system suggests their instantiation in particular cases for specific policy actions (e.g., after botnet detection in a specific RAN). In this sense, the use of flow analysis and DPI can be tightly combined into a two-stage detection process. The first process is devoted to detecting anomalous activities through flow analysis. However, since our architecture uses flow aggregations for detection, it cannot figure out who is responsible for the anomaly. For this reason we need a second stage, in which a virtual IDS is deployed into the RANs where the anomalous UEs' behavior has been observed. Both detection procedures can be performed even in case the UEs' IP addresses change. A migration process of the IDS to a new RAN is then required to follow the anomaly source, and thus continue providing the same detection capabilities in the UEs' path [23].

Moreover, our proposal is highly flexible because it allows dynamically deploying new virtualized resources in order to detect anomaly symptoms in a particular RAN when network traffic increases. The proposal is also extensible since the

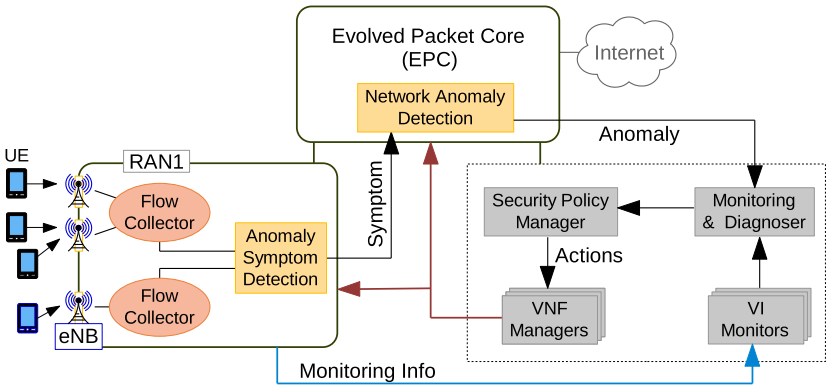

Fig. 2. Network Anomaly Detection System. (Red arrows represent control actions sent by VNF Managers; blue arrows do monitoring information and stats sent by RAN resources; black arrows do flow information in the detection process; orange boxes are VNFs –Anomaly Symptom Detection and Network Anomaly Detection; and grey boxes are MANO elements.)

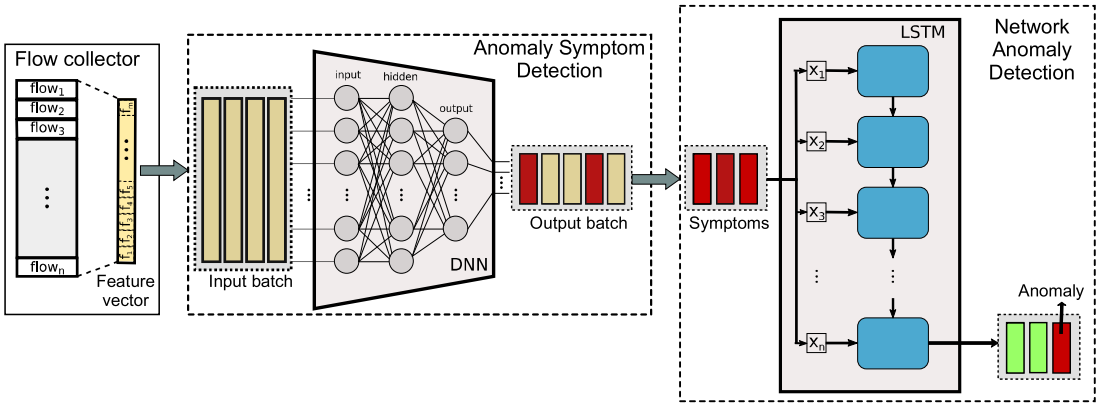

Fig. 3. Detail of the low-level (ASD) and high-level (NAD) modules.

symptom detection, which is the most expensive analysis process, is distributed among RANs while the anomaly detection is centralized in the core network, known as Evolved Packet Core (EPC), as it only requires symptoms as input.

Focusing on the implementation of both ASD and NAD functions, we propose to use machine learning techniques, particularly some related to deep learning, in order to analyze network flows. In this sense, a key aspect is to identify what network flow features should be analyzed so as to detect anomaly symptoms. In our proposal, flow accounting is a two-step process: flow export and flow collection. The flow exporter, also known as observation point, is responsible for the metering process; that is, the creation of flow records from observed traffic. The flow collector is in charge of retrieving and storing the flows created by the flow exporter as well as sending network flow features in a way suitable to the anomaly symptom detection module.

IV. DEEP LEARNING APPLIED TO THE ANOMALY DETECTION PROBLEM IN 5G NETWORKS

Regarding our proposal, the anomaly detection problem can be tackled from the machine learning perspective by using a wide range of methods. Moreover, most of them are suitable to be deployed according to the VNO security policies included in the proposed cyberdefense architecture. In particular, we propose to use deep learning because it has been shown to be a useful choice in other similar problems and because it has a set features (as the following sections detail) that are well aligned with the new requirements based on the KPIs identified for 5G networks (and described earlier).

In our architecture, the anomaly detection is arranged in two levels, as shown in Fig. 3. At the low level, the flow collector gathers all the different flows during a given period of time and calculates a vector of features that the ASD module will classify as anomalous or normal. This initial classification has to be made as quickly as possible, even at the expense of sacrificing accuracy for a lower response time. If an anomaly is suspected, a symptom packet composed of the feature vector involved, a time stamp, and the type of anomaly detected (when a multi-class method is being used), is sent to the next level, the NAD module.

The NAD receives several streams of symptoms from all the ASDs, sorts them by their time stamps and assembles a time sequence of symptoms. The task of deciding whether

this sequence belongs to any of a set of attack categories can be seen as a sequence-of-symptoms classification problem.

In our research we focus on analyzing the throughput of the ASD module from an execution time perspective, rather than from the classification accuracy point of view. The volumes of traffic that each RAN has to manage in a 5G network make it crucial to be able to process a sufficient number of flows per second. We do not require this level to reach a great detection performance because we rely on the second level (NAD) to refine the final detection results. Nonetheless, determining the ASD minimum accuracy level to achieve a good performance in the NAD module is out of the scope of this work.

A. SUITABLE DEEP LEARNING MODELS

There is a wide range of machine learning techniques that can accomplish the above requirements. In the case of the NAD module, it has no strict time restrictions if the ASDs have sufficient classification performance. Therefore, any deep learning architecture capable of dealing with complex patterns in symptom series can be an option. Bearing all this in mind, an LSTM network was the model selected to implement our NAD. Conversely, our ASD has to be able to evaluate a great amount of features per second and it does not need to analyze time series. In this paper we are mainly interested in the ASD module, as its run time performance is critical.

DBN and SAE models were chosen to be used in our ASD time measurement process. Two main reasons motivated this choice: they essentially share the same structure, (i.e. the prediction can be computed basically by using matrix operations followed by an activation function); and they can be used in both supervised and unsupervised learning.

If a labeled set is available, we can use a DBN or SAE followed by a classification layer. However, if no labeled set is available, we propose to convert a DBN into a semi-supervised method, simply training with the normal network traffic and using the DBN as a sort of Discriminative RBM without any further backpropagation layer [24]. Even in this case, the number of matrix operations needed to obtain the prediction is essentially the same.

Some authors propose using SVM as a final layer after a DBN or SAE [18]. This model can act as both a supervised two-class method and a semi-supervised one-class one. We initially discarded this approach because it is not well suited to large-scale high-dimensional datasets, and the number of support vectors obtained during the training phase is variable. This fact would make it difficult to obtain a good upper bound of the execution time.

Three different implementations of a supervised DBN were selected to be evaluated. They have one, three, and six hidden layers, respectively. These are followed by a classification layer which outputs a single binary label. The depth necessary is determined by the degree of variation in the network data. These models are assumed to be previously trained by using a training set labeled as *normal* or *anomalous*.

B. ASD CLASSIFICATION PERFORMANCE

Although the experimental component of this work is focused on prediction time evaluation, it is necessary to figure out whether good detection/classification results can be obtained with the restrictions that this proposal imposes. Any of the aforementioned models could have been selected; however, we decided to use one of the simplest classifiers, a DBN followed by a classification layer, because our tests revealed it is good enough to show successful experimental results.

An efficient machine learning mechanism also needs a set of highly discriminative features, so we have computed a variety of statistical measures and information metrics from a batch of network flows. Feature engineering cannot be completely avoided. Nevertheless, by using deep learning it is expected that the model will learn increasingly higher-level features by combining the original input components.

In the rest of this section we describe the botnet traffic dataset used in our experimentation for checking the classification performance, the feature vector with which our model was trained and the classification performance achieved which supports the suitability of our proposal.

1) THE CTU DATASET

The 5G scenario is really complex, and we need a publicly available dataset suitable to be used to test our proposal. Anomalies in communication networks are commonly related to malicious behavior of computers infected by malware forming botnets.

The complexity of this scenario and the availability of recent datasets with realistic traffic made us focus on botnets attack detection. CTU is a publicly available dataset suitable to be used in this context. It tries to accomplish all the good requirements to be considered as a good dataset: it has real botnets attacks and not simulations, unknown traffic from a large network, ground-truth labels for training and evaluating, as well as different types of botnets. The CTU dataset comprises thirteen scenarios with different numbers of infected computers and seven botnet families [25].

2) NETWORK FLOW FEATURES

Many artificial intelligence tasks can be solved by choosing the right representation or set of features, and then providing them to a simple machine learning algorithm. However, it is not easy to know which features should be extracted –or learning representation problem. It is generally admitted that great expertise is necessary to choose a proper feature vector in network security contexts. Conversely, deep learning has proved to be highly efficient in extracting high-level features from input vectors in a non-supervised way given a sufficient amount of data. This prevents the need of experts. Our research tries to automatically compute discriminative features from an input vector with many metrics.

Flow export requires a protocol, e.g., NetFlow, which defines how flow records are transported to the flow collector. NetFlow transports flow records containing source and

TABLE 1. Selected TCP/UDP features computed from the network flows for the classification test.

Total (TCP/UDP)	Features (t)
3/3	Number of flows, number of incoming flows, number of outgoing flows.
2/2	% of incoming and outgoing flows over total.
2/2	% of symmetric and asymmetric incoming flows over total.
15/15	Sum, maximum, minimum, mean, and variance of IP packets per incoming, outgoing, and total flows.
15/15	Sum, maximum, minimum, mean, and variance of bytes per incoming, outgoing, and total flows.
15/15	Sum, maximum, minimum, mean, and variance of source bytes per incoming, outgoing, and total flows.
2/2	Number of different source IPs for incoming flows and destination IPs for outgoing flows.
4/4	Number of different source and destination ports for incoming and outgoing flows.
2/2	Entropy of source IPs for incoming flows and destination IPs for outgoing flows.
4/4	Entropy of source and destination ports for incoming and outgoing flows.
4/4	% of source and destination ports >1024 for incoming and outgoing flows.
4/4 (*144 features*)	% of source and destination ports ≤ 1024 for incoming and outgoing flows.

destination IP addresses and ports, start and end timestamps, type of service, level 3 protocol, TCP flags, next hop router, input and output SNMP interfaces, source and destination autonomous systems, and network masks. Moreover, each flow carries aggregated information about the number of packets and bytes exchanged.

Each flow record could be directly sent from the flow collector to the symptom detection module. However, the flow record is too simple and only allows extracting a few features. A step forward is to obtain aggregated views from a window of flow records [19]. In this case, a timer and/or accounting function triggers the aggregation process after receiving a certain number of flows (offset). The aggregated view is computed taking one or more time periods in a multi-scale way, (e.g., three feature vectors could be obtained considering 5-minute, 1-minute, and 1-second periods).

The required features can be different depending on the network anomaly types to be detected; for example, finding TCP anomalies (e.g., TCP port scanning) only requires TCP features. In particular, we have used 288 features that are calculated from the features presented in TABLE 1 by making use of two subsets of flows: one whole set of TCP/UDP flows and another with TCP/UDP flows filtered so that their source/destination IPs belong to the flows within the offset.

3) FIRST CLASSIFICATION RESULTS

In order to obtain some classification results that support our decision of using a simple deep learning model for our ADS, we need some suitable models to be tested. First, feature vectors are created from the CTU dataset by taking aggregated views of 30 and 60 seconds after receiving every network flow; that is, using an offset of 1 flow. In this way, if the last received network flow is anomalous, then the corresponding feature vector will be labeled as anomalous as well. A different offset can be chosen for performance purposes. In Fig. 4 an example of the feature computation process is depicted for an offset value of two flows.

Unfortunately, this sort of dataset is highly unbalanced as the percentage of anomalous traffic is usually smaller in a real network. This makes the learning process more challenging

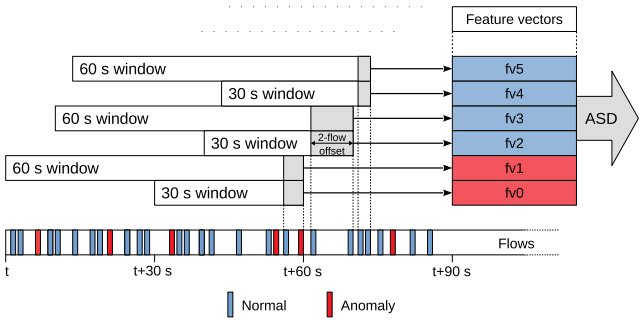

Fig. 4. Example of aggregated view computation for 30 and 60 second periods. In this case, an offset of 2 network flows controls the pace at which the aggregations are computed. The feature is labeled as anomalous only if one of the flows within the offset is anomalous.

because, in this case, neural networks tend to classify all traffic as normal and consider anomalies as noise, achieving an accuracy close to 100%. A variety of strategies can be adopted to prevent this behavior [26]. In this case, we are using a weighted loss function to compensate the different label frequencies. In addition, more appropriate error metrics have been selected as opposed to accuracy: the pair (*precision*, *recall*), or more concisely the *F1 score*. These metrics provide more information about the goodness of the model in unbalanced contexts. The entries in the confusion matrix are denoted as:

- *True Positive* (TP). This entry refers to the number of positive examples which are correctly predicted as positives.
- *True Negative* (TN). It denotes the number of negative examples correctly classified as negatives.
- *False Positive* (FP). This entry is defined as the number of negative examples incorrectly classified as positives.
- *False Negative* (FN). It is the number of positive examples incorrectly assigned as negatives.

And the proposed error metrics are:

- *Precision*, which indicates what percent of positive predictions were correct.

$$Precision = \frac{TP}{TP + FP}$$

- *Recall* or *sensitivity*, which defines what percent of positive cases did a classifier catch.

$$Recall = \frac{TP}{TP + FN}$$

- *F1 score*, which shows the trade-off between the precision and recall regarding the positive class.

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

In our study, two different training/test partitions were made. On one hand, our first dataset is built by the whole dataset split into training, validation, and test, while the models are evaluated with the same botnets they learned. On the other hand, our second dataset is built using the same partition suggested by the authors of the CTU dataset [25]. They split the datasets into training and testing, meeting the following constraints: the training and cross-validation datasets should be approximately 80% of the dataset; the testing dataset 20% of the dataset; and none of the botnet families used in the training and cross-validation dataset should be used in the testing dataset. This ensures generalization and detection of new behaviors, a critical aspect when dealing with cyberdefense scenarios in 5G networks.

Bearing all this in mind, we wanted to find a simple deep learning model with sufficient classification performance. We only needed a good result which gave us some evidence on the feasibility of our proposal. Therefore, we restricted our search domain to DBNs with up to three hidden layers having 16, 8, or 4 elements each combined in decreasing order, (e.g., 288-16-8-4-1, 288-16-8-1, or 288-16-1).[1] The set of hyper-parameters and the respective ranges are listed in TABLE 2.

TABLE 2. Configurations and hyper-parameters for our test.

Hyper-parameter	Values tested
Hidden layers	1, 2, 3
Sizes of the hidden layers	4, 8, 16
Learning rate	0.1, 0.01, 0.001
Weight-decay L2-regularization	0.0, 0.01
Activation function	Rectified Linear Unit (ReLU)
Dropout	0.0, 0.2
Batch normalization	On, Off

Deeper configurations were not tried because it would have been necessary to apply more advanced strategies to prevent model overfitting. The determination of the best model among the ones described in II-B, as well as the optimum feature set when the search space is less constrained and the models more expressive are out of scope of this article.

The training procedure was carried out following the two aforementioned datasets. The best results were obtained by the configuration 288-16-4-1. Evaluating the test dataset one, our winner model reached a precision of 0.8126 and a recall of 0.9934, thus giving an F1 score of 0.8940. This is not an impressive result in precision because 18.74% of the features classified as anomalous were actually normal. However, it obtained a good performance in recall because 99.34% of the actual anomalies were correctly labeled.

However, we were more interested in the classification capacity of new botnets, so the second dataset was used to test it. Using the test dataset containing only unknown botnets, the evaluation of the trained model reached a precision of 0.6863 and a recall of 0.7095 on average; that is, an F1 score of 0.6977. Approximately 71% of the anomalous traffic was correctly detected, even though the model had never seen those botnets. The complete list of scores for the test scenarios is provided in TABLE 3.

TABLE 3. Classification results in the unseen CTU botnet scenarios [25]

Test set	Precision	Recall	F1 score
Scenario 1	0.73	0.91	0.81
Scenario 2	0.65	0.55	0.59
Scenario 6	0.70	0.95	0.80
Scenario 8	0.40	0.76	0.53
Scenario 9	0.95	0.38	0.54

Our expectations are that these results will become reasonably improved by the second level detector in the NAD module. This module will receive anomaly information from all the RANs, detect complex patterns and, expectedly, filter the misclassified anomalies.

V. EXPERIMENTAL RESULTS

There is a significant number of libraries and frameworks related to deep learning that can be used to train the model proposed in this article. They have to be capable of managing LSTM Recurrent Networks (this is the model we have planned to use in our NAD) and DBNs. They should also have support for multiple GPUs and, optionally, allow multi-node parallel execution for future extensions. The ones selected from those which meet the requirements were: TensorFlow 1.4 [27], Caffe2 0.8.1 [28], [29], Theano 1.0.0rc1 [30] (using Lasagne 0.2.dev1 [31]), PyTorch 0.2.0_4 [32], MXNet 0.11.0 [33], and CNTK 2.2 [34]. All of them have been widely used in different kinds of R&D and commercial projects and in different scenarios. Caffe2 and PyTorch are receiving good acceptance by the community despite their recent appearance.

As a general rule, researchers are not only interested in the performance of the model, but also in the speed of the training process because that is the most time-consuming part when using deep learning techniques. Although the evaluation performance is not generally considered critical, in our case we need a framework that also evaluates the trained model as quick as possible due to the time restrictions inherent in our problem of cyberdefense in 5G networks. It can be argued that such a framework is not strictly necessary to make predictions, but by using it we can take advantage of the great support offered by the open source community.

In this experimental work, the performance of the above frameworks has been evaluated by using a DBN. We have

[1]The first number is the dimension of the input feature vector, and the last is the dimension of the output label.

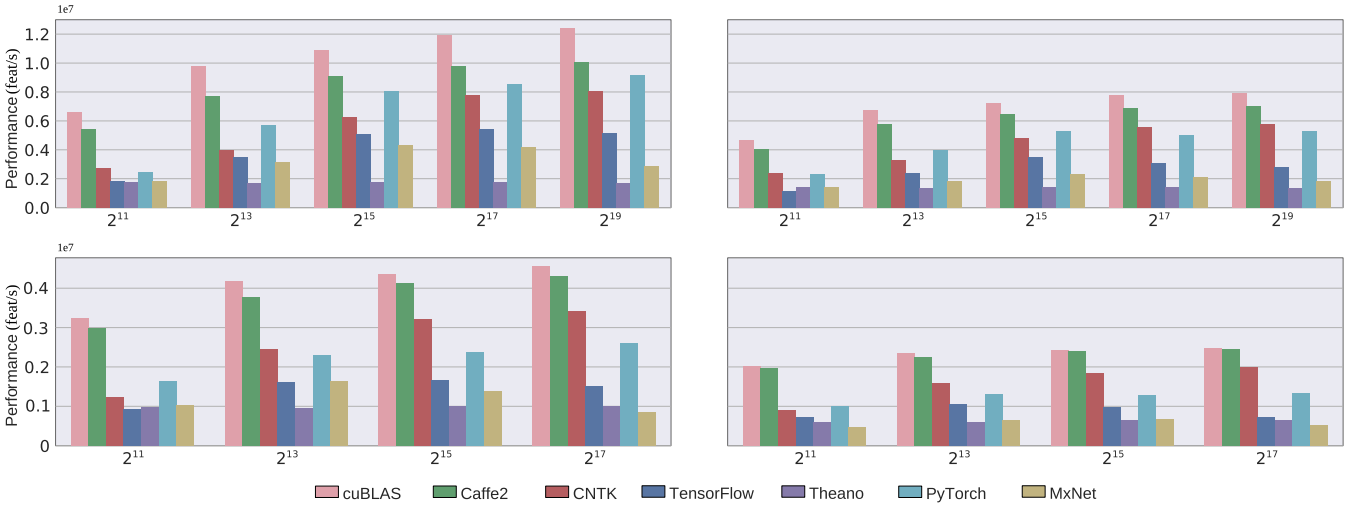

Fig. 5. Six hidden layer neural network (GPU) performance for the most common batch and feature vector sizes when running on the GPU. The largest batch size for each feature vector size is limited by the GPU memory (8 GB). (Top-left) 64-feature vector. (Top-right) 128-feature vector. (Bottom-left) 256-feature vector. (Bottom-right) 512-feature vector. One and three hidden layer neural networks show similar behavior except scale.

included in the evaluation our own implementation on cuBLAS [35] as a baseline solution with minimal overhead. The DBN model keeps the same input and output size for each experiment, (i.e., the input layer has the same size as the feature vector, and the output layer size is 2). Softmax is then used to estimate the likelihood of belonging to the normal or anomalous class. In further work, the number of outputs can be extended to match the different detected attacks, acting as a symptom label. This can be made by simply setting the output size to the number of classes and adjusting the size of each hidden layer, if necessary.

The chosen architectures are:

- One hidden full layer of 128 floats.
- Three hidden full layers of 128, 64, and 32 floats.
- Six hidden full layers of 128, 128, 64, 64, 32, and 32 floats.

The number of layers were set up to work with a range of feasible depths. Their sizes were selected to serve as an upper-bound estimate of the ones potentially needed. We assume that the batched feature vectors arrive at maximum speed. Then the CPU uses them to either feed the DBN contained in the GPU (normally by means of a memory copy), or directly evaluate it.

Several factors have to be taken into account in order to reach the best performance. The batch size has a great influence on the execution time, regardless of which processor is running the model. If the model is evaluated by the CPU, a big batch takes advantage of the Advanced Vector Extensions. However, above a certain batch size there will probably be more cache/TLB misses and page faults, resulting in a worse throughput. Conversely, if the model is evaluated by the GPU, increasing the batch size can decrease the execution time as a greater percentage of the available processing units can be used simultaneously. However, in this latter case, the time spent on transferring the batch to the GPU's memory also increases. In this experimental work we used only one GPU, so the batch size is limited by the GPU's memory size.

Every layer in the model has an associated matrix-vector product. For example, in the three hidden layers model, the sizes of the matrices when the input is a 256-element feature vector and the output layer has 2 neurons are 256×128, 128×64, 64×32, and 32×2 float elements. The input vector size only affects the size of the first matrix leaving the rest unchanged.

We carried out the performance evaluation using a workstation with 32GB of RAM, a six-core Intel i7-5930K at 3.5GHz with hyper-threading running Linux, and one NVIDIA GeForce GTX 1080 with 8GB RAM.

A. DETERMINING THE OPTIMUM BATCH SIZE AND FRAMEWORK

Most deep learning frameworks are software libraries for numerical computation based on data flow graphs, where the nodes represent operations and the edges represent the data communicated between them. Its flexible architecture allows deploying computation to one or more CPUs or GPUs in a variety of hardware platforms with a single API. We have decided to take it as a black box and determine the more suitable framework by using a benchmark executed during the installation procedure of our software on each RAN. This benchmark runs several tests to find out the best batch size for a given framework. Performance is measured for all the feasible combinations of the tuple (*model*, *framework*, *vector size*, *batch size*). In this context, *model* can be any of the different deep learning models trained to detect the anomalies and *framework* represents any of the suitable ones for our hardware. The benchmark might also obtain any additional performance metric needed to assist the Security Policy Manager to make more accurate decisions.

For the shake of clarity, Fig. 5 only shows the resulting performance for the six hidden layer neural network running on the GPU. The performances of the one and three hidden layer neural networks have no significant behavior change except scale. The figure shows that the speedup, as a function of the vector size when the batch size is fixed, is not linear. There might be several reasons for this fact: memory transfer time, computing time, number of GPU threads or CUDA's grid and block dimensions, among others. However, our goal is not to provide an insightful description of this behavior, but tuning our software to maximize its performance. Specifically, Fig. 5 presents the marks obtained by every framework for an input vector of 64, 128, 256, and 512 features, and a variety of batch sizes.

As seen in Fig. 5, the best performing for all vectors and batches sizes is our own implementation using cuBLAS, followed closely by Caffe2. Given that all the analyzed frameworks use cuBLAS for matrix operation, our ad hoc implementation of the models can be considered as an upper bound of the achievable throughput. Our version does not suffer from the overhead imposed by the abstraction layers that the frameworks need in order to provide flexibility and extended functionality. However, its limiting fact is precisely its lack of flexibility because we cannot benefit from all the improvements that the open source community is continuously making to the frameworks (e.g. multi-GPU support, function optimizations or bug fixing).

Analyzing the individual behavior or each framework, cuBLAS's best marks are 12, 7.9, 4.5, and 2.48 million feature vectors per second for 64, 128, 256, and 512 vector size, respectively. It is followed closely by Caffe2, with a small performance difference that becomes even smaller when batch size is increased. In contrast, MxNet and Theano obtain the worst marks; Theano achieves its best result (1.77 million feature vectors per second) with a batch size of 2^{11} and a vector size of 64; while MxNet obtains 4.3 million feature vectors per second as its best mark for a batch size of 2^{15} and a vector size of 64. It should be noticed that Theano is a special case because the performance remains almost unchanged for all batch sizes with the vector size fixed. In general, the implementation and internal optimizations of each framework determine its final performance; therefore, the performance offered by all frameworks varies substantially among batch sizes.

Although the number of features can be potentially unlimited, we set our feature vector to 256 elements. We chose this number because it is a representative value considering our previous ASD classification performance (see Section IV-B.2). With this input vector size and running the models on our workstation, the best batch sizes in the three DBN architectures are shown in TABLE 4.

We can consider our cuBLAS implementation as the maximum practical throughput. However, fixing the vector size to 256 and taking into account only commercial frameworks, the best performance is offered by Caffe2 for all batch sizes. For one-layer architecture and a batch size of 2^{17} it reaches 5.1 million feature vectors per second, while the second best option is CNTK offering 3.8 million feature vectors per second. On the other hand, MxNet and Theano compete for the worst position in the ranking, providing poor results in almost every configuration when compared with Caffe2 and PyTorch. Particularly, in this configuration MxNet reaches its best mark of 1.54 million feature vectors per second for the batch size 2^{13}. Theano obtains 1.64 million feature vectors per second for a batch size of 2^{17}.

TABLE 4. Optimum batch size (see Fig. 5).

Framework	One hidden layer	Three hidden layers	Six hidden layers
TensorFlow	2^{15}	2^{17}	2^{15}
Caffe2	2^{17}	2^{17}	2^{17}
Theano	2^{17}	2^{17}	2^{17}
PyTorch	2^{11}	2^{17}	2^{17}
MxNet	2^{15}	2^{13}	2^{13}
CNTK	2^{17}	2^{17}	2^{17}
cuBLAS	2^{17}	2^{17}	2^{17}

We should bear in mind that these results have been obtained by running the benchmark on the specific workstation described above. Therefore, the marks could be rather different with other existing hardware configurations, where the virtualized computational resources available in each RAN can result in a different optimal combination of deep learning framework and batch size, for example. This makes it crucial to run the benchmark suite on each RAN at installation stage and determine the best framework that suits each particular RAN in the 5G network. It should also be noted that, although Caffe2 running on a GPU provides the best performance, it could not be the most suitable option. As a case in point, a system with low traffic can decide not to use a GPU due to power consumption policy. In this case, TensorFlow on a CPU would be the best option. Additionally, other system configurations could be considered such as frameworks using more than one GPU. In any case, our architecture allows the self-adaptation to every hardware context.

B. CPU VERSUS GPU PERFORMANCE

GPUs have become an essential tool for machine learning computation, because deep learning models are actually well suited to be executed on them. The more complex and independent the computations are, the more impressive the performance gain usually is.

The model proposed in this article cannot be considered too deep, and the matrices are rather small when compared with the ones typically used by other deep learning models (e.g., CNNs). In our particular case, the time spent in transferring the batch from main memory to the GPU cannot be disregarded when compared with the computing time. If the batch is small, the number of memory transfers increases, limiting the throughput. Conversely if the batch is large, the memory becomes the most common limitation. Fig. 6 illustrates how some of the frameworks improve as the batch size increases. In contrast, some others even get worse in

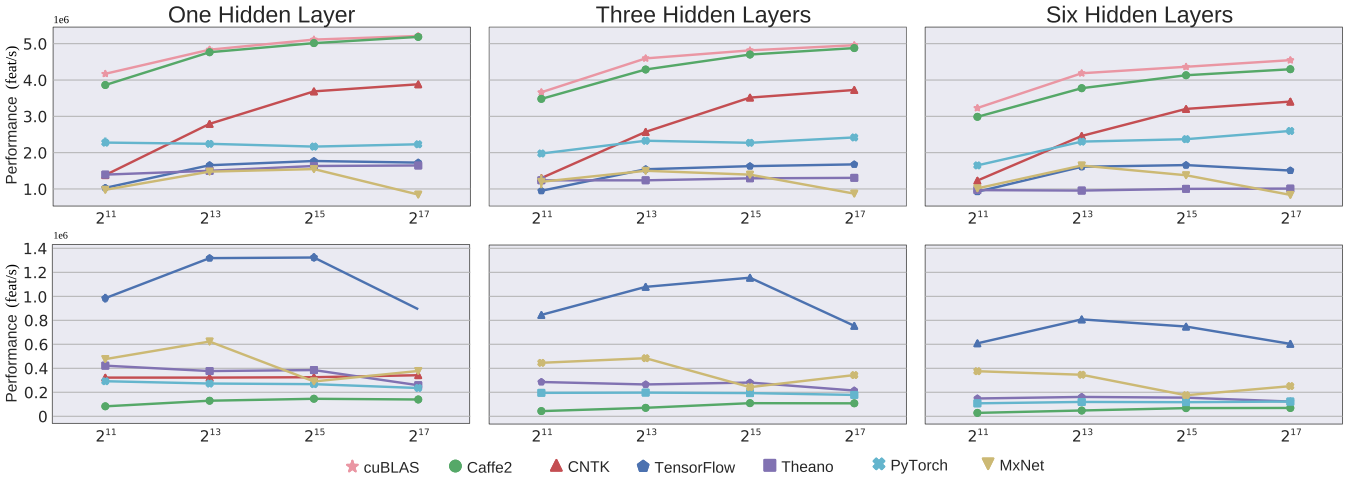

Fig. 6. Comparison between GPU and CPU performance with a vector of 256 features for each of the three DBN architectures. (Top) GPU performance. (Bottom) CPU performance.

the same case. GPU shows for every framework a significant performance improvement compared to CPU. It is important to note that cuBLAS is a library running on GPU, so we have excluded our own implementation based on cuBLAS from CPU benchmark.

It is worth mentioning that TensorFlow running on CPU gets a better mark than some frameworks running on GPU. In particular, for one hidden layer architecture with 2^{13} and 2^{15} batch sizes, TensorFlow behaves better than MxNet and similar to Theano. In contrast, Caffe2, which gets the best mark on GPU, obtains the worst one on CPU. In general, all the frameworks running on CPU show a performance lower than 1 million feature vectors per second, with the TensorFlow exception, which reaches up to 1.3 million feature vector per second in the one-layer architecture.

In models with more layers, the gap between CPU and GPU performance increases dramatically. In the six-layer architecture, all the GPU frameworks obtain a better result than any other CPU framework. Even CPU TensorFlow reaching a maximum of 0.8 million feature vectors per second with 2^{13} batch size, gets only close to the worse GPU framework (MxNet) with 0.9 million feature vectors per second.

The GPU performances for the three models are rather similar. This proves that the GPU's 2 560 cores have a huge computing power on high parallelizable computations like matrix-vector product. This leads us to conclude that a great amount of time is spent on data transfers instead of calculations. There can be an opportunity to improve the global throughput when several GPUs are used, if we manage to overlap computations and DMA transfers.

Connecting these experimental results with the cyberdefense problem targeted in this study, we can consider a 5G scenario with a city of 1 million of habitants, each of them having only one UE. These UEs will be connected to the eNBs (elements of the LTE Radio Access Network for connectivity purposes), demanding services and each generating 10 flows per second (fps) on average. In order to offer proper wireless connectivity, that city maintains a single RAN with 50 deployed eNBs to which all UEs are connected. There is a perfect balancing of 20 000 UEs per eNB and there is one flow collector per eNB; that is, each flow collector will be managing 200 000 fps. If we assume an ASD per flow collector and an offset of 1 flow, this fact will result in having 200 000 feature vectors per second inputting to each ASD. As seen in the experiments, this number could be perfectly managed by each ASD separately, even by a CPU using the TensorFlow framework with a batch size of 2^{15}. Instead, when taking 5 eNBs (from 50) sending data to the ASD through the corresponding flow collectors, we should obtain 1 million of feature vectors per second. This will overload an ASD using a single CPU. However, a single GPU can manage this situation easily even considering the TensorFlow framework, as shown in Fig. 6.

In a worse situation where taking 15 eNBs sending data to the ASD at the same time, 3 million feature vectors per second would be generated. In this case, TensorFlow on a single GPU will not be able to adequately process this volume of feature vectors. Therefore, the system would switch to another deep learning framework that can handle 3 or more million feature vectors per second. In this specific case, CNTK, Caffe2, or cuBLAS are the best options. By choosing 2^{17} as batch size the system would be able to handle 3.9, 5.1, and 5.1 million feature vectors per second with CNTK, Caffe2, and cuBLAS, respectively.

Finally, focusing on a worst case where 10 million feature vectors per second would be generated, with all the eNBs simultaneously active in the RAN and sending data to a unique ASD, this could not be assumed with any of the studied frameworks. As a solution, we suggest two mechanisms: a) dynamically deploy more computational resources, through new virtualized components as VNFs;

and b) adapting the flow aggregation frequency in order to reduce the rate of collected feature vectors.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

A self-adaptive system for anomaly detection and cyberdefense has been proposed in the context of a 5G mobile network architecture in which unpredictable traffic fluctuation is expected. This fact will make the system to automatically adapt the computational resources and inspection elements in order to optimize and ensure the detection processes. This adaptation can be made seamlessly by means of the virtualized network functions present in the designed system.

Our proposal is based on a novel two-level deep machine learning model where the first level is a supervised or semi-supervised learning method implementing a DBN or a SAE running on every RAN as quickly as possible. This sacrifices some accuracy due to the amount of network traffic that a RAN handles. Its task is to detect symptoms, that is, local anomalous traffic conditions happening during a configurable short time period. All the collected symptoms are sent to the NAD component, where they are assembled and used as input for a LSTM Recurrent Network, trained in a supervised way to recognize temporal patterns of cyberattacks.

A number of experiments were also conducted to realize a comprehensive comparative performance evaluation of deep learning frameworks. The aim was to determine the best suited one for reaching the highest processing performance. The experimental results show that each system in the 5G network will be able to use one framework or another depending on, for example, the availability of physical resources (GPU or CPU) in a given RAN or the network flow reception rate.

Several directions for future improvement are currently being considered. Firstly, extending the experimental work related to detection/classification so as to determine which of the deep learning models from a selection, their respective best hyper-parameters, and optimum feature set are more appropriate for each configuration and a given throughput requirement. Moreover, it is also necessary to train the two different levels using real data and to evaluate the accuracy of the anomaly detection architecture as a whole. This has been identified as the next step in a real 5G network, as the one being currently built in the SELFNET EU H2020 project.

REFERENCES

[1] S. Muthuramalingam, M. Thangavel, and S. Sridhar, "A review on digital sphere threats and vulnerabilities," in *Combating Security Breaches and Criminal Activity in the Digital Sphere*. Antwerp, Belgium: IGI Global, 2016, pp. 1–21.

[2] Sourcefire, Inc., "Snort: An open source network intrusion detection and prevention system," [Online]. Available: http://www.snort.org.

[3] V. Richariya, U. P. Singh, and R. Mishra, "Distributed approach of intrusion detection system: Survey," *Int. J. Adv. Comput. Res.*, vol. 2, no. 6, pp. 358–363, 2012.

[4] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to Snort system," *Future Generat. Comput. Syst.*, vol. 80, pp. 157–170, Mar. 2018.

[5] J. Yu, B. Yang, R. Sun, and Y. Chen, "FPGA-based parallel pattern matching algorithm for network intrusion detection system," in *Proc. Int. Conf. Multimedia Inf. Netw. Secur.*, Nov. 2009, pp. 458–461.

[6] Y.-M. Hsiao, M.-J. Chen, Y.-S. Chu, and C.-H. Huang, "High-throughput intrusion detection system with parallel pattern matching," *IEICE Electron. Exp.*, vol. 9, no. 18, pp. 1467–1472, Sep. 2012.

[7] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2nd Quart., 2016.

[8] Q. A. Tran, F. Jiang, and J. Hu, "A real-time NetFlow-based intrusion detection system with improved BBNN and high-frequency field programmable gate arrays," in *Proc. IEEE 11th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Jun. 2012, pp. 201–208.

[9] J. Gardiner and S. Nagaraja, "On the security of machine learning in malware C&C detection: A survey," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 59:1–59:39, Dec. 2016.

[10] The 5G Infraestructure Public Private Partnership. *Key Performance Indicators (KPI)*. [Online]. Available: http://5g-ppp.eu/kpis

[11] L. Fernández-Maimó, F. J. García-Clemente, M. Gil-Pérez, and G. Martínez-Pérez, "On the performance of a deep learning-based anomaly detection system for 5G networks," in *Proc. IEEE 3rd Smart World Congr.*, Aug. 2017, pp. 303–310.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[13] A. J. Aviv and A. Haeberlen, "Challenges in experimenting with botnet detection systems," in *Proc. 4th Conf. Cyber Secur. Experim. Test*, 2011, pp. 1–8.

[14] G. Vormayr, T. Zseby, and J. Fabini, "Botnet communication patterns," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2768–2796, 4th Quart., 2017.

[15] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, nos. 1–2, pp. 18–28, Feb. 2009.

[16] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson. (Jan. 2017). "Shallow and deep networks intrusion detection system: A taxonomy and survey." [Online]. Available: https://arxiv.org/abs/1701.02145

[17] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Proc. Artif. Intell. Stat.*, 2009, pp. 448–455.

[18] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016.

[19] Q. Niyaz, W. Sun, and A. Y. Javaid. (Nov. 2016). "A deep learning based DDoS detection system in software-defined networking (SDN)." [Online]. Available: https://arxiv.org/abs/1611.07400

[20] J. Erfanian *et al.*, "Network operator perspectives on NFV priorities for 5G," ETSI NFV Netw. Oper. Council, White Paper, Feb. 2017. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper_5g.pdf

[21] M. S. Siddiqui *et al.*, "Hierarchical, virtualised and distributed intelligence 5G architecture for low-latency and secure applications," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 9, pp. 1233–1241, Sep. 2016.

[22] P. Neves *et al.*, "Future mode of operations for 5G—The SELFNET approach enabled by SDN/NFV," *Comput. Standards Interfaces*, vol. 54, no. 4, pp. 229–246, Nov. 2017.

[23] M. Gil-Pérez *et al.*, "Dynamic reconfiguration in 5G mobile networks to proactively detect and mitigate botnets," *IEEE Internet Comput.*, vol. 21, no. 5, pp. 28–36, Sep. 2017.

[24] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the restricted Boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, Dec. 2013.

[25] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Secur.*, vol. 45, pp. 100–123, Sep. 2014.

[26] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: A review," *GESTS Int. Trans. Comput. Sci. Eng.*, vol. 30, no. 1, pp. 25–36, 2006.

[27] M. Abadi *et al.* (Mar. 2016). "TensorFlow: Large-scale machine learning on heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1603.04467

[28] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, Nov. 2014, pp. 675–678.

[29] Facebook Open Source. *Caffe2: A New Hightweight, Modular, and Scalable Deep Learning Framework*. Accessed: Jan. 2018. [Online]. Available: http://caffe2.ai

[30] Theano Development Team. (May 2016). "Theano: A Python framework for fast computation of mathematical expressions." [Online]. Available: https://arxiv.org/abs/1605.02688
[31] S. Dieleman *et al.* (Aug. 2015). *Lasagne: First Release*. [Online]. Available: http://doi.org/10.5281/zenodo.27878
[32] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like environment for machine learning," in *Proc. BigLearn, NIPS Workshop*, 2011, pp. 1–6.
[33] T. Chen *et al.* (Dec. 2015). "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1512.01274
[34] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, p. 2135.
[35] NVIDIA Corporation. *Dense Linear Algebra on GPUs*. Accessed: Jan. 2018. [Online]. Available: http://developer.nvidia.com/cublas

LORENZO FERNÁNDEZ MAIMÓ received the M.Sc. degree from the University of Murcia. He is currently an Assistant Professor with the Department of Computer Engineering, University of Murcia. His research interests include machine learning and deep learning.

ÁNGEL LUIS PERALES GÓMEZ received the M.Sc. degree in computer science from the University of Murcia, where he is currently pursuing the Ph.D. degree with the Department of Computer Engineering. His research interests include deep learning and the security of distributed communication networks.

FÉLIX J. GARCÍA CLEMENTE received the Ph.D. degree in computer science from the University of Murcia. He is currently an Associate Professor with the Department of Computer Engineering, University of Murcia. His research interests include the security and management of distributed communication networks.

MANUEL GIL PÉREZ received the Ph.D. degree (Hons.) in computer science from the University of Murcia, Spain. He is currently a Research Associate with the Department of Information and Communication Engineering, University of Murcia. His research interests include cybersecurity, including intrusion detection systems, trust management, privacy-preserving data sharing, and security operations in highly dynamic scenarios.

GREGORIO MARTÍNEZ PÉREZ received the Ph.D. degree in computer science from the University of Murcia. He is currently a Full Professor with the Department of Information and Communication Engineering, University of Murcia. His research interests include the security and management of distributed communication networks.