ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martinez-del-Rincon, J., Siracusa, D. (2020). LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, 17(2), 876-889 [10.1109/TNSM.2020.2971776].

Availability:
This version is available at: https://hdl.handle.net/11585/722638 since: 2020-11-11

Published:
DOI: http://doi.org/10.1109/TNSM.2020.2971776

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

12 October 2024

This is the final peer-reviewed accepted manuscript of:

R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón and D. Siracusa, "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection," in *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876-889, June 2020.

The final published version is available online at DOI: 10.1109/TNSM.2020.2971776

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection

R. Doriguzzi-Corin[α], S. Millar[β], S. Scott-Hayward[β], J. Martínez-del-Rincón[β], D. Siracusa[α]
[α]ICT, Fondazione Bruno Kessler - Italy
[β]CSIT, Queen's University Belfast - Northern Ireland

***Abstract*—Distributed Denial of Service (DDoS) attacks are one of the most harmful threats in today's Internet, disrupting the availability of essential services. The challenge of DDoS detection is the combination of attack approaches coupled with the volume of live traffic to be analysed. In this paper, we present a practical, lightweight deep learning DDoS detection system called LUCID, which exploits the properties of Convolutional Neural Networks (CNNs) to classify traffic flows as either malicious or benign. We make four main contributions; (1) an innovative application of a CNN to detect DDoS traffic with low processing overhead, (2) a dataset-agnostic preprocessing mechanism to produce traffic observations for online attack detection, (3) an activation analysis to explain LUCID's DDoS classification, and (4) an empirical validation of the solution on a resource-constrained hardware platform. Using the latest datasets, LUCID matches existing state-of-the-art detection accuracy whilst presenting a 40x reduction in processing time, as compared to the state-of-the-art. With our evaluation results, we prove that the proposed approach is suitable for effective DDoS detection in resource-constrained operational environments.**

***Index Terms*—Distributed Denial of Service, Deep Learning, Convolutional Neural Networks, Edge Computing**

I. INTRODUCTION

DDoS attacks are one of the most harmful threats in today's Internet, disrupting the availability of essential services in production systems and everyday life. Although DDoS attacks have been known to the network research community since the early 1980s, our network defences against these attacks still prove inadequate.

In late 2016, the attack on the Domain Name Server (DNS) provider, Dyn, provided a worrying demonstration of the potential disruption from targeted DDoS attacks [1]. This particular attack leveraged a botnet (Mirai) of unsecured IoT (Internet of Things) devices affecting more than 60 services. At the time, this was the largest DDoS attack recorded, at 600 Gbps. This was exceeded in February 2018 with a major DDoS attack towards Github [2]. At its peak, the victim saw incoming traffic at a rate of 1.3 Tbps. The attackers leveraged a vulnerability present in memcached, a popular database caching tool. In this case, an amplification attack was executed using a spoofed source IP address (the victim IP address). If globally implemented, BCP38 "Network Ingress Filtering" [3] could mitigate such an attack by blocking packets with spoofed IP addresses from progressing through the network. However, these two examples illustrate that scale rather than sophistication enables the DDoS to succeed.

In recent years, DDoS attacks have become more difficult to detect due to the many combinations of attack approaches. For example, multi-vector attacks where an attacker uses a combination of multiple protocols for the DDoS are common. In order to combat the diversity of attack techniques, more nuanced and more robust defence techniques are required. Traditional signature-based intrusion detection systems cannot react to new attacks. Existing statistical anomaly-based detection systems are constrained by the requirement to define thresholds for detection. Network Intrusion Detection Systems (NIDSs) using machine learning techniques are being explored to address the limitations of existing solutions. In this category, deep learning (DL) systems have been shown to be very effective in discriminating DDoS traffic from benign traffic by deriving high-level feature representations of the traffic from low-level, granular features of packets [4], [5]. However, many existing DL-based approaches described in the scientific literature are too resource-intensive from the training perspective, and lack the pragmatism for real-world deployment. Specifically, current solutions are not designed for online attack detection within the constraints of a live network where detection algorithms must process traffic flows that can be split across multiple capture time windows.

Convolutional Neural Networks (CNNs), a specific DL technique, have grown in popularity in recent times leading to major innovations in computer vision [6]–[8] and Natural Language Processing [9], as well as various niche areas such as protein binding prediction [10], [11], machine vibration analysis [12] and medical signal processing [13]. Whilst their use is still under-researched in cybersecurity generally, the application of CNNs has advanced the state-of-the-art in certain specific scenarios such as malware detection [14]–[17], code analysis [18], network traffic analysis [4], [19]–[21] and intrusion detection in industrial control systems [22]. These successes, combined with the benefits of CNN with respect to reduced feature engineering and high detection accuracy, motivate us to employ CNNs in our work.

While large CNN architectures have been proven to provide state-of-the-art detection rates, less attention has been given to minimise their size while maintaining competent performance in limited resource environments. As observed with the Dyn attack and the Mirai botnet, the opportunity for launching DDoS attacks from unsecured IoT devices is increasing as we deploy more IoT devices on our networks. This leads to consideration of the placement of the defence mechanism. Mitigation of attacks such as the Mirai and Memcached examples include the use of high-powered appliances with the capacity to absorb volumetric DDoS attacks. These appliances are located locally at the enterprise or in the Cloud. With the

drive towards edge computing to improve service provision, it becomes relevant to consider the ability to both protect against attacks closer to the edge and on resource-constrained devices. Indeed, even without resource restrictions, it is valuable to minimize resource usage for maximum system output.

Combining the requirements for advanced DDoS detection with the capability of deployment on resource-constrained devices, this paper makes the following contributions:

- A DL-based DDoS detection architecture suitable for online resource-constrained environments, which leverages CNNs to learn the behaviour of DDoS and benign traffic flows with both low processing overhead and attack detection time. We call our model LUCID (Lightweight, Usable CNN in DDoS Detection).
- A dataset-agnostic preprocessing mechanism that produces traffic observations consistent with those collected in existing online systems, where the detection algorithms must cope with segments of traffic flows collected over pre-defined time windows.
- A kernel activation analysis to interpret and explain to which features LUCID attaches importance when making a DDoS classification.
- An empirical validation of LUCID on a resource-constrained hardware platform to demonstrate the applicability of the approach in edge computing scenarios, where devices possess limited computing capabilities.

The remainder of this paper is structured as follows: Sec. II reviews and discusses the related work. Sec. III details the methodology with respect to the network traffic processing and the LUCID CNN model architecture. Sec. IV describes the experimental setup detailing the datasets and the development of LUCID with the hyper-parameter tuning process. In Sec. V, LUCID is evaluated and compared with the state-of-the-art approaches. Sec. VI introduces our kernel activation analysis for explainability of LUCID's classification process. Sec. VII presents the experiment and results for the DDoS detection at the edge. Finally, the conclusions are provided in Sec. VIII.

II. Related work

DDoS detection and mitigation techniques have been explored by the network research community since the first reported DDoS attack incident in 1999 [23]. In this section, we review and discuss anomaly-based DDoS detection techniques categorised by statistical approaches and machine learning approaches, with a specific focus on deep learning techniques.

A. Statistical approaches to DDoS detection

Measuring statistical properties of network traffic attributes is a common approach to DDoS detection, and generally involves monitoring the entropy variations of specific packet header fields. By definition, the entropy is a measure of the diversity or the randomness in a data set. Entropy-based DDoS detection approaches have been proposed in the scientific literature since the early 2000s, based on the assumption that during a volumetric DDoS attack, the randomness of traffic features is subject to sudden variations. The rationale is that volumetric DDoS attacks are typically characterised by a huge number of attackers (in the order of hundreds of thousands [24]), often utilising compromised devices that send a high volume of traffic to one or more end hosts (the victims). As a result, these attacks usually cause a drop in the distribution of some of the traffic attributes, such as the destination IP address, or an increase in the distribution of other attributes, such as the source IP address. The identification of a DDoS attack is usually determined by means of thresholds on these distribution indicators.

In one of the first published works using this approach, Feinstein et al. [25] proposed a DDoS detection technique based on the computation of source IP address entropy and Chi-square distribution. The authors observed that the variation in source IP address entropy and chi-square statistics due to fluctuations in legitimate traffic was small, compared to the deviations caused by DDoS attacks. Similarly, [26] combined entropy and volume traffic characteristics to detect volumetric DDoS attacks, while the authors of [27] proposed an entropy-based scoring system based on the destination IP address entropy and dynamic combinations of IP and TCP layer attributes to detect and mitigate DDoS attacks.

A common drawback to these entropy-based techniques is the requirement to select an appropriate detection threshold. Given the variation in traffic type and volume across different networks, it is a challenge to identify the appropriate detection threshold that minimizes false positive and false negative rates in different attack scenarios. One solution is to dynamically adjust the thresholds to auto-adapt to the normal fluctuations of the network traffic, as proposed in [28], [29].

Importantly, monitoring the distribution of traffic attributes does not provide sufficient information to distinguish between benign and malicious traffic. To address this, some approaches apply a rudimentary threshold on the packet rate [30] or traceback techniques [31], [32].

An alternative statistical approach is adopted in [33], where Ahmed et al. use packet attributes and traffic flow-level statistics to distinguish between benign and DDoS traffic. However, this solution may not be suitable for online systems, since some of the flow-level statistics used for the detection e.g. total bytes, number of packets from source to destination and from destination to source, and flow duration, cannot be computed when the traffic features are collected within observation time windows. Approaches based on flow-level statistics have also been proposed in [34]–[39], among many others. In particular, [36]–[39] use flow-level statistics to feed CNNs and other DL models, as discussed in Sec. II-C. To overcome the limitations of statistical approaches to DDoS detection, machine learning techniques have been explored.

B. Machine Learning for DDoS detection

As identified by Sommer and Paxson in [40], there has been extensive research on the application of machine learning to network anomaly detection. The 2016 Buczak and Guven survey [41] cites the use of Support Vector Machine (SVM), k-Nearest Neighbour (k-NN), Random Forest, Naïve Bayes etc. achieving success for cyber security intrusion detection. However, due to the challenges particular to network intrusion

detection, such as high cost of errors, variability in traffic etc., adoption of these solutions in the "real-world" has been limited. Over recent years, there has been a gradual increase in availability of realistic network traffic data sets and an increased engagement between data scientists and network researchers to improve model explainability such that more practical Machine Learning (ML) solutions for network attack detection can be developed. Some of the first application of machine learning techniques specific to DDoS detection has been for traffic classification. Specifically, to distinguish between benign and malicious traffic, techniques such as extra-trees and multi-layer perceptrons have been applied [42], [43].

In consideration of the realistic operation of DDoS attacks from virtual machines, He et al. [44] evaluate nine ML algorithms to identify their capability to detect the DDoS from the source side in the cloud. The results are promising with high accuracy (99.7%) and low false positives ($< 0.07\%$) for the best performing algorithm; SVM linear kernel. Although there is no information provided regarding the detection time or the datasets used for the evaluation, the results illustrate the variability in accuracy and performance across the range of ML models. This is reflected across the literature e.g. [45], [46] with the algorithm performance highly dependent on the selected features (and datasets) evaluated. This has motivated the consideration of deep learning for DDoS detection, which reduces the emphasis on feature engineering.

C. Deep Learning for DDoS detection

There is a small body of work investigating the application of DL to DDoS detection. For example, in [47], the authors address the problem of threshold setting in entropy-based techniques by combining entropy features with DL-based classifiers. The evaluation demonstrates improved performance over the threshold-based approach with higher precision and recall. In [48], a Recurrent Neural Network (RNN)-Intrusion Detection System (IDS) is compared with a series of previously presented ML techniques (e.g. J48, Artificial Neural Network (ANN), Random Forest, and SVM) applied to the NSL-KDD [49] dataset. The RNN technique demonstrates a higher accuracy and detection rate.

Some CNN-based works [36]–[39], as identified in Sec. II-A, use flow-level statistics (total bytes, flow duration, total number of flags, etc.) as input to the proposed DL-based architectures. In addition, [36] and [37] combine the statistical features with packet payloads to train the proposed IDSs.

In [19], Kehe Wu et al. present an IDS based on CNN for multi-class traffic classification. The proposed neural network model has been validated with flow-level features from the NSL-KDD dataset encoded into 11x11 arrays. Evaluation results show that the proposed model performs well compared to complex models with 20 times more trainable parameters. A similar approach is taken by the authors of [20], where the CNN-based IDS is validated over datasets NSL-KDD and UNSW-NB-15 [50]. In [51], the authors study the application of CNNs to IDS by comparing a series of architectures (shallow, moderate, and deep, to reflect the number of convolution and pooling layers) across 3 traffic datasets; NSL-KDD, Kyoto Honeypot [52], and MAWILab [53]. In the results presented, the shallow CNN model with a single convolution layer and single max. pooling layer performed best. However, there is significant variance in the detection accuracy results across the datasets, which indicates instability in the model.

More specific to our DDoS problem, Ghanbari et al. propose a feature extraction algorithm based on the *discrete wavelet transform* and on the *variance fractal dimension trajectory* to maximize the sensitivity of the CNN in detecting DDoS attacks [5]. The evaluation results show that the proposed approach recognises DDoS attacks with 87.35% accuracy on the CAIDA DDoS attack dataset [54]. Although the authors state that their method allows real-time detection of DDoS attacks in a range of environments, no performance measurements are reported to support this claim.

DeepDefense [4] combines CNNs and RNNs to translate original traffic traces into arrays that contain packet features collected within sliding time windows. The results presented demonstrate high accuracy in DDoS attack detection within the selected ISCX2012 dataset [55]. However, it is not clear if these results were obtained on unseen test data, or are results from the training phase. Furthermore, the number of trainable parameters in the model is extremely large indicating a long and resource-intensive training phase. This would significantly challenge implementation in an online system with constrained resources, as will be discussed in Sec. V and VII.

Although deep learning offers the potential for an effective DDoS detection method, as described, existing approaches are limited by their suitability for online implementation in resource-constrained environments. In Sec. V, we compare our proposed solution, LUCID, with the state-of-the-art, specifically [4], [35], [36], [38], [47] and demonstrate the contributions of LUCID.

III. METHODOLOGY

In this paper we present LUCID, a CNN-based solution for DDoS detection that can be deployed in online resource-constrained environments. Our CNN encapsulates the learning of malicious activity from traffic to enable the identification of DDoS patterns regardless of their temporal positioning. This is a fundamental benefit of CNNs; to produce the same output regardless of where a pattern appears in the input. This encapsulation and learning of features whilst training the model removes the need for excessive feature engineering, ranking and selection. To support an online attack detection system, we use a novel preprocessing method for the network traffic that generates a spatial data representation used as input to the CNN. In this section, we introduce the network traffic preprocessing method, the CNN model architecture, and the learning procedure.

A. Network Traffic preprocessing

Network traffic is comprised of data flows between endpoints. Due to the shared nature of the communication link, packets from different data flows are multiplexed resulting in packets from the same flow being separated for transmission. This means that the processing for live presentation of traffic

TABLE I
GLOSSARY OF SYMBOLS.

α	Learning rate	n	Number of packets per sample
f	Number of features per packet	s	Batch size
h	Height of convolutional filters	t	Time window duration
id	5-tuple flow identifier	τ	Time window start time
k	Number of convolutional filters	$\mathcal{E}$	Array of labelled samples
m	Max pooling size	$\mathcal{L}$	Set of labels

to a NIDS is quite different to the processing of a static dataset comprising complete flows. For the same reason, the ability to generate flow-level statistics, as relied upon by many of the existing works described in Sec. II, is not feasible in an online system.

In order to develop our online NIDS, we created a tool that converts the traffic flows extracted from network traffic traces of a dataset into array-like data structures and splits them into sub-flows based on time windows. Shaping the input as packet flows in this manner creates a spatial data representation, which allows the CNN to learn the characteristics of DDoS attacks and benign traffic through the convolutional filters sliding over such input to identify salient patterns. This form of input is compatible with traffic captured in online deployments. The process is illustrated in Algorithm 1 and described next. The symbols are defined in Table I.

Algorithm 1 Network traffic preprocessing algorithm

Input: Network traffic trace (NTT), flow-level labels ($\mathcal{L}$), time window (t), max packets/sample (n)
Output: List of labelled samples ($\mathcal{E}$)

1: **procedure** PREPROCESSING(NTT, $\mathcal{L}$, t, n)
2: $\mathcal{E} \leftarrow \emptyset$ ▷ Initialise the set of samples
3: $\tau \leftarrow -1$ ▷ Initialise the time window start-time
4: **for all** $pkt \in NTT$ **do** ▷ Loop over the packets
5: $id \leftarrow pkt.tuple$ ▷ 5-tuple flow identifier
6: **if** $\tau == -1$ **or** $pkt.time > \tau + t$ **then**
7: $\tau \leftarrow pkt.time$ ▷ Time window start time
8: **end if**
9: **if** $|\mathcal{E}[\tau, id]| < n$ **then** ▷ Max n pkts/sample
10: $\mathcal{E}[\tau, id].pkts.append(pkt.features)$
11: **end if**
12: **end for**
13: $\mathcal{E} \leftarrow normalization_padding(\mathcal{E})$
14: **for all** $e \in \mathcal{E}$ **do** ▷ Labelling
15: $e.label \leftarrow \mathcal{L}[e.id]$ ▷ Apply the label to the sample
16: **end for**
17: **return** $\mathcal{E}$
18: **end procedure**

Feature extraction. Given a traffic trace file from the dataset and a pre-defined time window of length t seconds, the algorithm collects all the packets from the file with capture time between t_0, the capture time of the first packet, and time $t_0 + t$. From each packet, the algorithm extracts 11 attributes (see Table II). We intuitively exclude those attributes that would be detrimental to the generalization of the model, such as IP addresses and TCP/UDP ports (specific to the end-hosts and user applications), link layer encapsulation type (linked to the network interfaces) and application-layer attributes (e.g., IRC or HTTP protocol attributes).

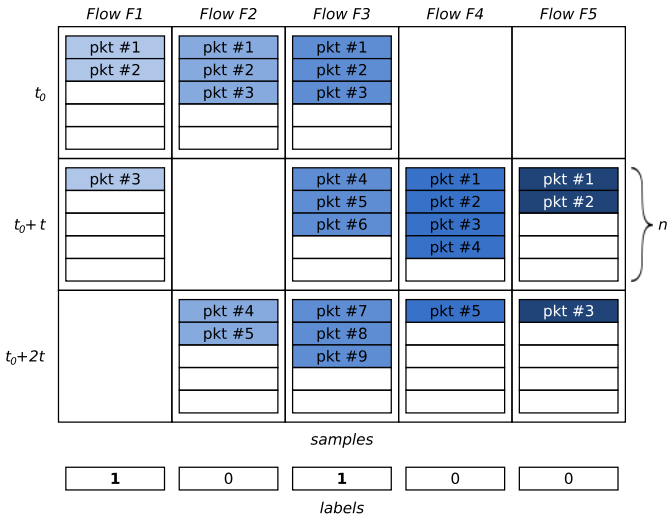

Fig. 1. Graphical representation of $\mathcal{E}$.

Data processing algorithm. This procedure, described in Algorithm 1 at lines 4-12, simulates the traffic capturing process of online IDSs, where the traffic is collected for a certain amount of time t before being sent to the anomaly detection algorithms. Hence, such algorithms must base their decisions on portions of traffic flows, without the knowledge of their whole life. To simulate this process, the attributes of the packets belonging to the same bi-directional traffic flow are grouped in chronological order to form an example of shape $[n, f]$ (as shown in Table II), where f is the number of features (11) and n is the maximum number of packets the parsing process collects for each flow within the time window. t and n are hyper-parameters for our CNN. Flows longer than n are truncated, while shorter flows are zero-padded at the end during the next stage after normalization. The same operations are repeated for the packets within time window $[t_0+t, t_0+2t]$ and so on, until the end of the file.

Logically, we hypothesize that short time windows enable the online systems to detect DDoS attacks within a very short time frame. Conversely, higher values of t and n offer more information on flows to the detection algorithms, which we expect to result in higher detection accuracy. The sensitivity of our CNN to the values of t and n is evaluated in Sec. IV.

The output of this process can be seen as a bi-dimensional array of samples ($\mathcal{E}[\tau, id]$ in Algorithm 1). A row of the array represents the samples whose packets have been captured in the same time window, whilst a column represents the samples whose packets belong to the same bi-directional flow. A graphical representation of array $\mathcal{E}$ is provided in Fig. 1.

Normalization and padding. Each attribute value is normalized to a $[0, 1]$ scale and the samples are zero-padded so that each sample is of fixed length n, since having samples of fixed length is a requirement for a CNN to be able to learn over a full sample set. In Fig. 1, each non-empty element of the array $\mathcal{E}$ is a compact graphical representation of a sample.

TABLE II
A TCP FLOW SAMPLE BEFORE NORMALIZATION.

	Pkt #	Time (sec)[1]	Packet Len	Highest Layer[2]	IP Flags	Protocols[3]	TCP Len	TCP Ack	TCP Flags	TCP Window Size	UDP Len	ICMP Type
Packets	0	0	151	99602525	0x4000	0011010001000b	85	336	0x018	1444	0	0
	1	0.092	135	99602525	0x4000	0011010001000b	69	453	0x018	510	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	j	0.513	66	78354535	0x4000	0010010001000b	0	405	0x010	1444	0	0
Padding	$j+1$	0	0	0	0	0000000000000b	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	n	0	0	0	0	0000000000000b	0	0	0	0	0	0

[1] Relative time from the first packet of the flow.
[2] Numerical representation of the highest layer recognised in the packet.
[3] Binary representation of the list of protocols recognised in the packet using the well-known Bag-of-Words (BoW) model. It includes protocols from Layer 2 (*arp*) to common clear text application layer protocols such as *http*, *telnet*, *ftp* and *dns*.

In each $\mathcal{E}$ element, coloured rows are the packets in the form of 11 normalized attributes (i.e., the upper part of Table II), while the white rows represent the zero-padding (i.e., the lower part of Table II). Please note that, empty elements in Fig. 1 are for visualization only and are not included in the dataset. An empty $\mathcal{E}[\tau, id]$ means that no packets of flow id have been captured in time window $[\tau, \tau + t]$ (e.g. $\mathcal{E}[t_0, F4]$).

Labelling. Each example $\mathcal{E}[\tau, id]$ is labelled by matching its flow identifier id with the labels provided with the original dataset (lines 14-16 in Algorithm 1). This also means that the value of the label is constant along each column of array $\mathcal{E}$, as represented in Fig. 1.

B. LUCID *Model Architecture*

We take the output from Algorithm 1 as input to our CNN model for the purposes of online attack detection. LUCID classifies traffic flows into one of two classes, either malicious (DDoS) or benign. Our objective is to minimise the complexity and performance time of this CNN model for feasible deployment on resource-constrained devices. To achieve this, the proposed approach is a lightweight, supervised detection system that incorporates a CNN, similar to that of [9] from the field of Natural Language Processing. CNNs have shared and reused parameters with regard to the weights of the kernels, whereas in a traditional neural network every weight is used only once. This reduces the storage and memory requirements of our model. The complete architecture is depicted in Fig. 2 and described in the next sections, with the hyper-parameter tuning and ablation studies being discussed in Sec. IV.

Input layer. Recall that each traffic flow has been reshaped into a 2-D matrix of packet features as per Sec. III-A, creating a novel spatial representation that enables the CNN to learn the correlation between packets of the same flow. Thus, this first layer takes as input a traffic flow represented by a matrix F of size n x f. F contains n individual packet vectors, such that $F = \{pkt_1, ..., pkt_n\}$ where pkt_n is the nth packet in a flow, and each packet vector has length f = 11 features.

CNN layer. As per Fig. 2, each input matrix F is operated on by a single convolutional layer with k filters of size h x f, with h being the length of each filter, and again f = 11. Each filter, also known as a kernel or sliding window, convolves over F with a step of 1 to extract and learn local features that contain useful information for detection of DDoS and benign flows. Each of the k filters generates an activation map a of size $(n - h + 1)$, such that $a_k = ReLU(Conv(F)W_k, b_k)$, where W_k and b_k are the weight and bias parameters of the kth filter that are learned during the training stage. To introduce non-linearity among the learned filters, we use the rectified linear activation function $ReLU(x) = max\{0, x\}$, as per convention for CNNs. All activation maps are stacked, creating an activation matrix A of size $(n - h + 1)$ x k, such that $A = [a_1|...|a_k]$.

There are two main benefits of including a CNN in our architecture. Firstly, it allows the model to benefit from efficiency gains compared to standard neural networks, since the weights in each filter are reused across the whole input. Sharing weights, instead of the full end-to-end connectivity with a standard neural net, makes the model more lightweight and reduces its memory footprint as the number of learnable parameters is greatly reduced. Secondly, during the training phase, the CNN automatically learns the weights and biases of each filter such that the learning of salient characteristics and features is encapsulated inside the resulting model during training. This reduces the time-consuming feature engineering and ranking involved in statistical and traditional machine learning methods, which relies on expert human knowledge. As a result, this model is more adaptable to new subtleties of DDoS attack, since the training stage can be simply repeated anytime with fresh training data without having to craft and rank new features.

Max pooling layer. For max pooling, we down-sample along the first dimension of A, which represents the temporal nature of the input. A pool size of m produces an output matrix m_o of size $(n - h + 1)/m$ x k, which contains the largest m activations of each learned filter, such that $m_o = [max(a_1)|...|max(a_k)]$. In this way, the model disregards the less useful information that produced smaller activations, instead paying attention to the larger activations. This also means that we dispose of the positional information of the activation, i.e. where it occurred in the original flow, giving a more compressed feature encoding, and, in turn, reducing the complexity of the network. m_o is then flattened to produce

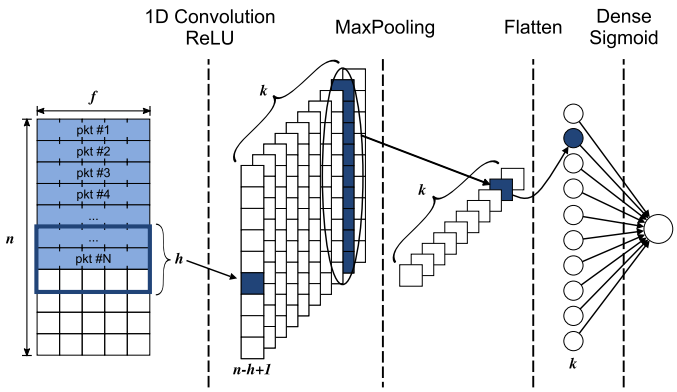

Fig. 2. LUCID architecture.

the final one-dimensional feature vector v to be input to the classification layer.

Classification layer. v is input to a fully-connected layer of the same size, and the output layer has a sole node. This output x is passed to the sigmoid activation function such that $\sigma(x) = 1/(1 + e^{-x})$. This constrains the activation to a value of between 0 and 1, hence returning the probability $p \in [0, 1]$ of a given flow being a malicious DDoS attack. The flow is classified as DDoS when $p > 0.5$, and benign otherwise.

C. The Learning Procedure

When training LUCID, the objective is to minimise its cost function through iteratively updating all the weights and biases contained within the model. These weights and biases are also known as trainable, or learnable, parameters. The cost function calculates the cost, also called the error or the loss, between the model's prediction, and the ground truth of the input. Hence by minimising this cost function, we reduce the prediction error. At each iteration in training, the input data is fed forward through the network, the error calculated, and then this error is back-propagated through the network. This continues until convergence is reached, when further updates don't reduce the error any further, or the training process reaches the set maximum number of epochs. With two classes in our problem the binary cross-entropy cost function is used. Formally this cost function c that calculates the error over a batch of s samples can be written as:

$$c = -\frac{1}{s}\sum_{j=1}^{s}(y_o \log p_o + (1 - y_o)\log(1 - p_o)) \qquad (1)$$

where y_o is the ground truth target label for each flow o in the batch of s samples, and p_o is the predicted probability flow o is malicious DDoS. This is supervised learning because each flow in our datasets is labelled with the ground truth, either DDoS or benign. To reduce bias in our learning procedure, we ensure that these datasets are balanced with equal numbers of malicious and benign flows, which gives a greater degree of confidence that the model is learning the correct feature representations from the patterns in the traffic flows. As previously highlighted, the learning is encapsulated inside the model by all the weights and biases, meaning that our approach does not require significant expert input to craft bespoke features and statistically assess their importance during preprocessing, unlike many existing methods, as outlined in Sec. II.

IV. Experimental Setup

A. Datasets

Our CNN model is validated with recent datasets ISCX2012 [55], CIC2017 [56] and CSECIC2018 [57] provided by the Canadian Institute for Cybersecurity of the University of New Brunswick (UNB), Canada. They consist of several days of network activity, normal and malicious, including DDoS attacks. The three datasets are publicly available in the form of traffic traces in *pcap* format including full packet payloads, plus supplementary text files containing the labels and statistical details for each traffic flow.

The UNB researchers have generated these datasets by using profiles to accurately represent the abstract properties of human and attack behaviours. One profile characterises the normal network activities and provides distribution models for applications and protocols (HTTP, SMTP, SSH, IMAP, POP3, and FTP) produced with the analysis of real traffic traces. Other profiles describe a variety of attack scenarios based on recent security reports. They are used to mimic the behaviour of the malicious attackers by means of custom botnets and well-known DDoS attacking tools such as High Orbit Ion Cannon (HOIC) [58] and its predecessor, the Low Orbit Ion Cannon (LOIC) [59]. HOIC and LOIC have been widely used by Anonymous and other hacker groups in some highly-publicized attacks against PayPal, Mastercard, Visa, Amazon, Megaupload, among others [60].

Table III shows the parts of the three datasets used in this work. In the table, the column *Traffic trace* specifies the name of the trace, according to [55], [56] and [57]. Specifically, the *ISCX2012-Tue15* trace contains a DDoS attack based on an IRC botnet. The *CIC2017-Fri7PM* trace contains a HTTP DDoS generated with LOIC, while the *CSECIC2018-Wed21* trace contains a HTTP DDoS generated with HOIC. With respect to the original file, the trace *CIC2017-Fri7PM* is reduced to timeslot 3.30PM-5.00PM to exclude malicious packets related to other cyber attacks (port scans and back-doors).

TABLE III
THE DATASETS FROM UNB [61].

Dataset	Traffic trace	#Flows	#Benign	#DDoS
ISCX2012	Tue15	571698	534320	37378
CIC2017	Fri7PM	225745	97718	128027
CSECIC2018	Wed21	1048575	360832	687743

In an initial design, the model was trained and validated on the ISCX2012 dataset producing high accuracy results. However, testing the model on the CIC2017 dataset confirmed the generally held observation that a model trained on one dataset will not necessarily perform well on a completely new dataset. In particular, we obtained a false negative rate of about 17%. This can be attributed to the different attacks

represented in the two datasets, as previously described. What we attempt in this work is to develop a model that when trained and validated across a mixed dataset can reproduce the high performance results on completely unseen test data. To achieve this, a combined training dataset is generated as described in Sec. IV-B.

B. Data preparation

We extract the 37378 DDoS flows from ISCX2012, plus randomly select 37378 benign flows from the same year to balance. We repeat this process with 97718/97718 benign/DDoS flows for CIC2017 and again with 360832/360832 benign/DDoS flows for CSECIC2018.

After the pre-preprocessing stage, where flows are translated into array-like data structures (Sec. III-A), each of the three datasets is split into training (90%) and test (10%) sets, with 10% of the training set used for validation. Please note that, the split operation is performed on a per-flow basis to ensure that samples obtained from the same traffic flow end up in the same split, hence avoiding the "contamination" of the validation and test splits with data used for the training. We finally combine the training splits from each year by balancing them with equal proportions from each year to produce a single training set. We do the same with the validation and test splits, to obtain a final dataset referred to as *UNB201X* in the rest of the paper. UNB201X training and validation sets are only used for training the model and tuning the hyper-parameters (Sec. IV-D), while the test set is used for the evaluation presented in Sec. V and VII, either as a whole combined test set, or as individual per-year test sets for state-of-the-art comparison.

A summary of the final UNB201X splits is presented in Table IV, which reports the number of samples as a function of time window duration t. As illustrated in Table IV, low values of this hyper-parameter yield larger numbers of samples. Intuitively, using short time windows leads to splitting traffic flows into many small fragments (ultimately converted into samples), while long time windows produce the opposite result. In contrast, the value of n has a negligible impact on the final number of samples in the dataset.

TABLE IV
UNB201X DATASET SPLITS.

Time Window	Total Samples	Training	Validation	Test
t=1s	480519	389190	43272	48057
t=2s	353058	285963	31782	35313
t=3s	310590	251574	27957	31059
t=4s	289437	234438	26055	28944
t=5s	276024	223569	24852	27603
t=10s	265902	215379	23931	26592
t=20s	235593	190827	21204	23562
t=50s	227214	184041	20451	22722
t=100s	224154	181551	20187	22416

C. Evaluation methodology

As per convention in the literature, we report the metrics *Accuracy (ACC)*, *False Positive Rate (FPR)*, *Precision* (or *Positive Predictive Value (PPV)*), *Recall* (or *True Positive Rate (TPR)*) and *F1 Score (F1)*, with a focus on the latter. *Accuracy* is the percentage of correctly classified samples (both benign and DDoS). *FPR* represents the percentage of samples that are falsely classified as DDoS. *PPV* is the ratio between the correctly detected DDoS samples and all the detected DDoS samples (true and false). *TPR* represents the percentage of DDoS samples that are correctly classified as such. The *F1 Score* is an overall measure of a model's performance; that is the harmonic mean of the *PPV* and *TPR*. These metrics are formally defined as follows:

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \quad FPR = \frac{FP}{FP+TN}$$

$$PPV = \frac{TP}{TP+FP} \quad TPR = \frac{TP}{TP+FN} \quad F1 = 2 \cdot \frac{PPV \cdot TPR}{PPV+TPR}$$

where *TP=True Positives*, *TN=True Negatives*, *FP=False Positives*, *FN=False Negatives*.

The output of the training process is a combination of trainable and hyper parameters that maximizes the *F1 Score* on the validation set or, in other words, that minimizes the total number of False Positives and False Negatives.

Model training and validation have been performed on a server-class computer equipped with two 16-core Intel Xeon Silver 4110 @2.1 GHz CPUs and 64 GB of RAM. The models have been implemented in Python v3.6 using the Keras API v2.2.4 [62] on top of Tensorflow 1.13.1 [63].

D. Hyper-parameter tuning

Tuning the hyper-parameters is an important step to optimise the model's accuracy, as their values influence the model complexity and the learning process. Prior to our experiments, we empirically chose the hyper-parameter values based on the results of preliminary tuning and on the motivations described per parameter. We then adopted a grid search strategy to explore the set of hyper-parameters using *F1 score* as the performance metric. At each point in the grid, the training continues indefinitely and stops when the loss does not decrease for a consecutive 25 times. Then, the search process saves the *F1 score* and moves to the next point.

As per Sec. IV-B, UNB201X is split into training, validation and testing sets. For hyper-parameter tuning, we use only the validation set. It is important to highlight that we do not tune to the test set, as that may artificially improve performance. The test set is kept completely unseen, solely for use in generating our experimental results, which are reported in Sec. V.

Maximum number of packets/sample. n is important for the characterization of the traffic and for capturing the temporal patterns of traffic flows. The value of n indicates the maximum number of packets of a flow recorded in chronological order in a sample.

The resulting set of packets describes a portion of the life of the flow in a given time window, including the (relative) time information of packets. Repetition-based DDoS attacks use a small set of messages at approximately constant rates, therefore a small value of n is sufficient to spot the temporal patterns among the packet features, hence requiring a limited number of trainable parameters. On the other hand, more

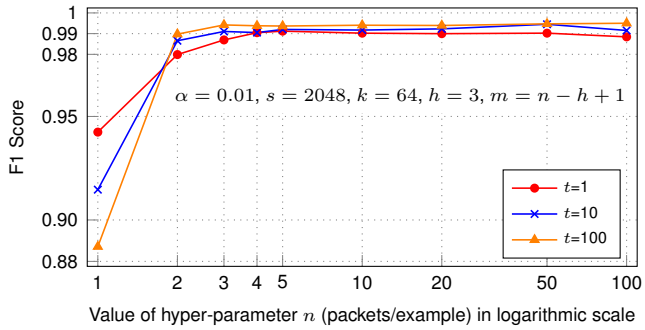

Fig. 3. Sensitivity of our model to hyper-parameter n.

complex attacks, such as the ones performed with the HOIC tool, which uses multiple HTTP headers to make the requests appear legitimate, might require a larger number of packets to achieve the desired degree of accuracy. Given the variety of DDoS tools used to simulate the attack traffic in the dataset (IRC-based bot, LOIC and HOIC), we experimented with n ranging between 1 and 100, and we compared the performance in terms of *F1 score*. The results are provided in Fig. 3 for different durations of time window t, but at fixed values of the other hyper-parameters for the sake of visualisation.

The *F1 score* steadily increases with the value of n when $n < 5$, and then stabilises when $n \geq 5$. However, an increase in *F1 score* is still observed up to $n = 100$. Although, a low value of n can be used to speed up the detection time (less convolutions) and to reduce the requirements in terms of storage and RAM (smaller sample size), which links to our objective of a lightweight implementation, we wish to balance high accuracy with low resource consumption. This will be demonstrated in Sec. VII.

Time Window. The time window t is used to simulate the capturing process of online systems (see Sec. III-A). We evaluated the *F1 score* for time windows ranging between 1 and 100 seconds (as in the related work e.g. [4]) at different values of n. The results are shown in Fig. 4.

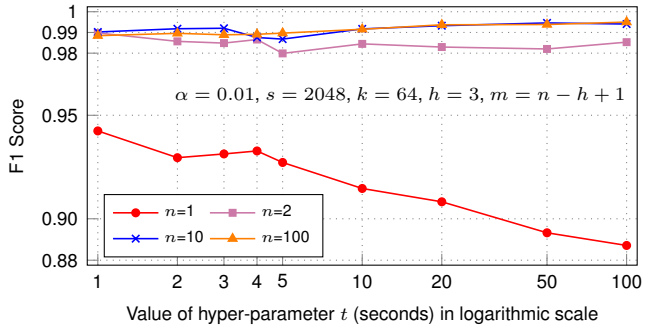

Fig. 4. Sensitivity of our model to hyper-parameter t.

Although the number of samples in the training set decreases when t increases (see Table IV), the CNN is relatively insensitive to this hyper-parameter for $n > 1$. With $n = 1$, the traffic flows are represented by samples of shape $[1, f]$, i.e. only one packet/sample, irrespective of the duration of the time window. In such a corner case, since the CNN cannot correlate the attributes of different packets within the same sample, the *F1 score* is more influenced by the number of samples in the training set (the more samples, the better).

Height of convolutional filters. h determines the height of the filters (the width is fixed to 11, the number of features), i.e. the number of packets to involve in each matrix operation. Testing with $h = 1, 2, 3, 4, 5$, we observed a small, but noticeable, difference in the *F1 score* between $h = 1$ (0.9934) and $h = 3$ (0.9950), with no major improvement beyond $h = 3$.

Number of convolutional filters. As per common practice, we experimented by increasing the number of convolutional filters k by powers of 2, from $k = 1$ to $k = 64$. We observed a steady increase in the *F1 score* with the value of k, which is a direct consequence of the increasing number of trainable parameters in the model.

Resulting hyper-parameter set. After conducting a comprehensive grid search on 2835 combinations of hyper-parameters, we have selected the CNN model configuration that maximises the *F1 score* on the UNB201X validation set (Table V). That is:

$$\mathbf{n} = 100, \ \mathbf{t} = 100, \ \mathbf{k} = 64, \ \mathbf{h} = 3, \ \mathbf{m} = 98$$

The resulting model, trained with batch size $s = 2048$ and using the Adam optimizer [64] with learning rate $\alpha = 0.01$, consists of 2241 trainable parameters, 2176 for the convolutional layer ($h \cdot f$ units for each filter plus bias, multiplied by the number of filters K) and 65 for the fully connected layer (64 units plus bias).

As previously noted, other configurations may present lower resource requirements at the cost of a minimal decrease in *F1 score*. For example, using $k = 32$ would reduce the number of convolutions by half, while $n = 10, 20, 50$ would also require fewer convolutions and a smaller memory footprint. However, setting $n = 100$ not only maximises the *F1 score*, but also enables a fair comparison with state-of-the-art approaches such as DeepDefense [4] (Sec. V), where the authors trained their neural networks using $n = 100$ (in [4], the hyper-parameter is denoted as T). Furthermore, the chosen configuration enables a worst-case analysis for resource-constrained scenarios such as that presented in Sec. VII.

These hyper-parameters are kept constant throughout our experiments presented in Sec. V and VII.

TABLE V
SCORES OBTAINED ON THE UNB201X VALIDATION SET.

Validation set	ACC	FPR	PPV	TPR	F1
UNB201X	0.9950	0.0083	0.9917	0.9983	0.9950

V. RESULTS

In this section, we present a detailed evaluation of the proposed approach with the datasets presented in Sec. IV-A. Evaluation metrics of *Accuracy (ACC)*, *False Positive Rate (FPR)*, *Precision* (*PPV*), *Recall* (*TPR*) and *F1 Score (F1)* have been used for performance measurement and for comparison with state-of-the-art models.

A. Detection accuracy

In order to validate our approach and the results obtained on the validation dataset, we measure the performance of LUCID in classifying unseen traffic flows as benign or malicious (DDoS). Table VI summarizes the results obtained on the various test sets produced through the procedure described in Sec. IV-B. As illustrated, the very high performance is maintained across the range of test datasets indicating the robustness of the LUCID design. These results are further discussed in Sec. V-B, where we compare our solution with state-of-the-art works reported in the scientific literature.

TABLE VI
LUCID DETECTION PERFORMANCE ON THE TEST SETS.

Test set	ACC	FPR	PPV	TPR	F1
ISCX2012	0.9888	0.0179	0.9827	0.9952	0.9889
CIC2017	0.9967	0.0059	0.9939	0.9994	0.9966
CSECIC2018	0.9987	0.0016	0.9984	0.9989	0.9987
UNB201X	0.9946	0.0087	0.9914	0.9979	0.9946

The results show that thanks to the properties of its CNN, LUCID learns to distinguish between patterns of malicious DDoS behaviour and benign flows. Given the properties of convolutional methods, these patterns are recognised regardless of the position they occupy in a flow, demonstrating that our spatial representation of a flow is robust. Irrespective of whether the DDoS event appears at the start or the end of the input, LUCID will produce the same representation in its output. Although the temporal dynamics in DDoS attacks might suggest that alternative DL architectures may seem more suitable (e.g. Long Short-Term Memory (LSTM)), our novel preprocessing method combined with the CNN removes the requirement for the model to maintain temporal context of each whole flow as the data is pushed through the network. In comparison, LSTMs are known to be very difficult to train, and their performance is inherently slower for long sequences compared to CNNs.

B. State-Of-The-Art Comparison

For a fair comparison between LUCID and the state-of-the-art, we focus our analysis on solutions that have validated the UNB datasets for DDoS attack detection.

We have paid particular attention to DeepDefense [4] as, similar to our approach, the model is trained with packet attributes rather than flow-level statistics used in other works. DeepDefense translates the *pcap* files of ISCX2012 into arrays that contain packet attributes collected within sliding time windows. The label assigned to a sample is the label of the last packet in the time window, according to the labels provided with the original dataset. The proposed data preprocessing technique is similar to LUCID's. However, in LUCID, a sample corresponds to a single traffic flow, whereas in DeepDefense a sample represents the traffic collected in a time window.

Of the four DL models presented in the DeepDefense paper, the one called 3LSTM produces the highest scores in the classification of DDoS traffic. Therefore, we have implemented 3LSTM for comparison purposes. The architecture of this model includes 6 LSTM layers of 64 neurons each, 2 fully connected layers of 128 neurons each, and 4 batch normalization layers. To directly compare the DL models, we have trained 3LSTM on the UNB201X training set with $n = 100$ and $t = 100$ as done with LUCID. We have compared our implementation of 3LSTM with LUCID on each of the four test sets, and present the F1 score results in Table VII.

TABLE VII
LUCID-DEEPDEFENSE COMPARISON (F1 SCORE).

Model	Trainable Parameters	ISCX 2012	CIC 2017	CSECIC 2018	UNB 201X
LUCID	2241	0.9889	0.9966	0.9987	0.9946
3LSTM	1004889	0.9880	0.9968	0.9987	0.9943

The results presented in Table VII show that LUCID and 3LSTM are comparable in terms of F1 score across the range of test datasets. However, in terms of computation time, LUCID outperforms 3LSTM in detection time. Specifically, as measured on the Intel Xeon server in these experiments, LUCID can classify more than 55000 samples/sec on average, while 3LSTM barely reaches 1300 samples/sec on average (i.e., more than 40 times slower). Indeed, LUCID's limited number of hidden units and trainable parameters contribute to a much lower computational complexity compared to 3LSTM.

As previously noted, there are a number of solutions in the literature that present performance results for the ISCX2012 and CIC2017 datasets. Notably, these works do not all specify whether the results presented are based on a validation dataset or a test dataset. For LUCID, we reiterate that the results presented in this section are based on a test set of completely unseen data.

TABLE VIII
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART APPROACHES USING THE ISCX2012 DATASET FOR DDOS DETECTION.

Model	ACC	FPR	PPV	TPR	F1
LUCID	0.9888	0.0179	0.9827	0.9952	0.9889
DeepDefense 3LSTM [4]	0.9841	N/A	0.9834	0.9847	0.9840
TR-IDS [36]	0.9809	0.0040	N/A	0.9593	N/A
E3ML [47]	N/A	N/A	N/A	0.9474	N/A

In Table VIII, we compare the performance of LUCID against state-of-the-art works validated on ISCX2012. Table VIII also includes the performance of 3LSTM as reported in the DeepDefense paper [4]. With respect to our version of 3LSTM, the scores are slightly lower, which we propose is due to the different *pcap* preprocessing mechanisms used in the two implementations. This indicates a performance benefit when using the LUCID preprocessing mechanism.

TR-IDS [36] is an IDS which adopts a text-CNN [9] to extract features from the payload of the network traffic. These

features, along with a combination of 25 packet and flow-level attributes, are used for traffic classification by means of a Random Forest algorithm. Accuracy and TPR of TR-IDS are above 0.99 for all the attack profiles available in ISCX2012 except the DDoS attack, for which the performance results are noticeably lower than LUCID.

E3ML [47] uses 20 entropy-based traffic features and three ML classifiers (a RNN, a Multilayer Perceptron and an Alternating Decision Tree) to classify the traffic as normal or DDoS. Despite the complex architecture, the TPR measured on ISCX2012 shows that E3ML is inclined to false negatives.

For the CIC2017 dataset, we present the performance comparison with state-of-the-art solutions in Table IX.

TABLE IX
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART APPROACHES USING THE CIC2017 DATASET FOR DDOS DETECTION.

Model	ACC	FPR	PPV	TPR	F1
LUCID	0.9967	0.0059	0.9939	0.9994	0.9966
DeepGFL [35]	N/A	N/A	0.7567	0.3024	0.4321
MLP [38]	0.8634	N/A	0.8847	0.8625	0.8735
1D-CNN [38]	0.9514	N/A	0.9814	0.9017	0.9399
LSTM [38]	0.9624	N/A	0.9844	0.8989	0.8959
1D-CNN + LSTM [38]	0.9716	N/A	0.9741	0.9910	0.9825

DeepGFL [35] is a framework designed to extract high-order traffic features from low-order features forming a hierarchical graph representation. To validate the proposed framework, the authors used the graph representation of the features to train two traffic classifiers, namely Decision Tree and Random Forest, and tested them on CIC2017. Although the precision scores on the several attack types are reasonably good (between 0.88 and 1 on any type of traffic profile except DDoS), the results presented in the paper reveal that the proposed approach is prone to false negatives, leading to very low F1 scores.

The authors of [38] propose four different DL models for DDoS attack detection in Internet of Things (IoT) networks. The models are built with combinations of LSTM, CNN and fully connected layers. The input layer of all the models consists of 82 units, one for each flow-level feature available in CIC2017, while the output layer returns the probability of a given flow being part of a DDoS attack. The model 1D-CNN+LSTM produces good classification scores, while the others seem to suffer from high false negatives rates.

To the best of our knowledge, no DDoS attack detection solutions validated on the CSECIC2018 dataset are available yet in the scientific literature.

C. Discussion

From the results presented and analysed in the previous sections, we can conclude that using packet-level attributes of network traffic is more effective, and results in higher classification accuracy, than using flow-level features or statistic information such as the entropy measure. This is not only proved by the evaluation results obtained with LUCID and our implementation of DeepDefense (both based on packet-level attributes), but also by the high classification accuracy of TR-IDS, which combines flow-level features with packet attributes, including part of the payload.

In contrast, E3ML, DeepGFL and most of the solutions proposed in [38], which all rely on flow-level features, seem to be more prone to false negatives, and hence to classify DDoS attacks as normal activity. The only exception is the model 1D-CNN+LSTM of [38], which produces a high TPR by combining CNN and RNN layers.

Furthermore, we highlight that LUCID has not been tuned to the individual datasets but rather to the validation portion of a combined dataset, and still outperforms the state-of-the-art on totally unseen test data.

VI. ANALYSIS

We now present interpretation and explanation of the internal operations of LUCID by way of proving that the model is learning the correct domain information. We do this by analysing the features used in the dataset and their activations in the model. To the best of our knowledge, this is the first application of a specific activation analysis to a CNN-based DDoS detection method.

A. Kernel activations

This approach is inspired by a similar study [65] to interpret CNNs in the rather different domain of natural language processing. However, the kernel activation analysis technique is transferable to our work. As each kernel has the same width as the input matrix, it is possible to remove the classifier, push the DDoS flows through the convolutional layer and capture the resulting activations per kernel. For each flow, we calculate the total activations per feature, which in the spatial input representation means per column, resulting in 11 values that map to the 11 features. This is then repeated for all kernels, across all DDoS flows, with the final output being the total column-wise activation of each feature. The intuition is that the higher a feature's activation when a positive sample i.e. a DDoS flow is seen, the more importance the CNN attaches to that particular feature. Conversely, the lower the activation, the lower the importance of the feature, and since our model uses the conventional rectified linear activation function, $ReLU(x) = max\{0, x\}$, this means that any negative activations become zero and hence have no impact on the Sigmoid classifier for detecting a DDoS attack.

Summing these activations over all kernels is possible since they are of the same size and operate over the same spatial representations. We analyse DDoS flows from the same UNB201X test set used in Sec. V-A.

Table X presents the ranking of the 11 features based on the post-$ReLU$ average column-wise feature activation sums, and highlights two features that activate our CNN the most, across all of its kernels.

Highest Layer. We assert that the CNN may be learning from the highest layer at which each DDoS flow operates.

TABLE X
RANKING OF THE TOTAL COLUMN-WISE FEATURE KERNEL ACTIVATIONS FOR THE UNB201X DATASET

Feature	Total Kernel Activation	Feature	Total Kernel Activation
Highest Layer	0.69540	Time	0.11108
IP Flags	0.30337	TCP Win Size	0.09596
TCP Flags	0.19693	TCP Ack	0.00061
TCP Len	0.16874	UDP Len	0.00000
Protocols	0.14897	ICMP Type	0.00000
Pkt Len	0.14392		

Recall that highest layer links to the type of DDoS attack e.g. network, transport, or application layer attack. We propose that this information could be used to extend LUCID to predict the specific type of DDoS attack taking place, and therefore, to contribute to selection of the appropriate protection mechanism. We would achieve the prediction by extending the dataset labeling, which we consider for future work.

IP Flags. In our design, this attribute is a 16-bit integer value which includes three bits representing the flags *Reserved Bit*, *Don't Fragment* and *More Fragments*, plus 13 bits for the *Fragment offset* value, which is non-zero only if bit *"Don't Fragment"* is unset. Unlike the *IP fragmented flood* DDoS attacks, in which the *IP flags* are manipulated to exploit the datagram fragmentation mechanisms, 99.99% of DDoS packets in the UNB datasets present an *IP flags* value of 0x4000, with only the *"Don't Fragment"* bit set to 1. A different distribution of *IP flags* is observed in the UNB benign traffic, with the *"Don't Fragment"* bit set to 1 in about 92% of the packets. Thus, the pattern of *IP flags* is slightly different between attack and benign traffic, and we are confident that LUCID is indeed learning their significance in DDoS classification, as evidenced by its 2nd place in our ranking.

B. Future Directions

However, even given this activation analysis, there is no definitive list of features that exist for detecting DDoS attacks with which we can directly compare our results. Analysing the related work, we identify a wide range of both stateless and stateful features highlighted for their influence in a given detection model, which is not unexpected as the features of use vary depending on the attack traffic. This is highlighted by the 2014 study [66], which concludes that different classes of attack have different properties, leading to the wide variance in features identified as salient for the attack detection. The authors also observe that the learning of patterns specific to the attack scenario would be more valuable than an effort to produce an attack-agnostic finite list of features. We, therefore, conclude from our analysis that LUCID appears to be learning the importance of relevant features for DDoS detection, which gives us confidence in the prediction performance.

Linked to this activation analysis, we highlight adversarial robustness as a key consideration for the deployment of ML-based IDSs. As detailed in [67], the two main attacks on IDSs are during training via a poisoning attack (i.e. corruption of the training data), or in testing, when an evasion attack attempts to cause incorrect classification by making small perturbations to observed features. Our activation analysis is a first step in the investigation of the model behaviour in adversarial cases with the feature ranking in Table X highlighting the features for perturbation for evasion attacks. Of course, the adversary model (goal, knowledge, and capability) dictates the potential for a successful attack. For example, the attacker would require full knowledge of the CNN and kernel activations, and have the ability to forge traffic within the network. The construction of defences robust to adversarial attacks is an open problem [68] and an aspect which we will further explore for LUCID.

VII. USE-CASE: DDOS DETECTION AT THE EDGE

Edge computing is an emerging paradigm adopted in a variety of contexts (e.g. fog computing [69], edge clouds [70]), with the aim of improving the performance of applications with low-latency and high-bandwidth requirements. Edge computing complements centralised data centres with a large number of distributed nodes that provide computation services close to the sources of the data.

The proliferation of attacks leveraging unsecured IoT devices (e.g., the Mirai botnet [71] and its variants) demonstrate the potential value in edge-based DDoS attack detection. Indeed, with edge nodes close to the IoT infrastructure, they can detect and block the DDoS traffic as soon as it leaves the compromised devices. However, in contrast to cloud high-performance servers, edge nodes cannot exploit sophisticated solutions against DDoS attacks, due to their limited computing and memory resources. Although recent research efforts have demonstrated that the mitigation of DDoS attacks is feasible even by means of commodity computers [72], [73], edge computing-based DDoS detection is still at an early stage.

In this section, we demonstrate that our DDoS detection solution can be deployed and effectively executed on resource-constrained devices, such as edge nodes or IoT gateways, by running LUCID on an NVIDIA Jetson TX2 development board [74], equipped with a quad-core ARM Cortex-A57@2 GHz CPU, 8 GB of RAM and a 256-core Pascal@1300 MHz Graphics Processing Unit (GPU). For the experiments, we used Tensorflow 1.9.0 with GPU support enabled by cuDNN, a GPU-accelerated library for deep neural networks [75].

A. Detection

In the first experiment, we analyse the applicability of our approach to online edge computing environments by estimating the prediction performance in terms of samples processed per second. As we are aware that edge nodes do not necessarily mount a GPU device, we conduct the experiments with and without the GPU support on the UNB201X test set and discuss the results.

We note that in an online system, our preprocessing tool presented in Section III-A can be integrated into the server/edge device. The tool would process the live traffic collected from the NICs of the server/edge device, collecting the packet attributes, organising them into flows and, after a predefined time interval, T, pass the data structure to the CNN for

inference. We acknowledge that the speed of this process will influence the overall system performance. However, as we have not focused on optimising our preprocessing tool, rather on optimising detection, its evaluation is left as future work. Instead, in these experiments, we load the UNB datasets from the hard disk rather than processing live traffic.

With respect to this, one relevant parameter is the batch size, which configures how many samples are processed by the CNN in parallel at each iteration. Such a parameter influences the speed of the detection, as it determines the number of iterations and, as a consequence, the number of memory reads required by the CNN to process all the samples in the test set (or the samples collected in a time window, in the case of online detection).

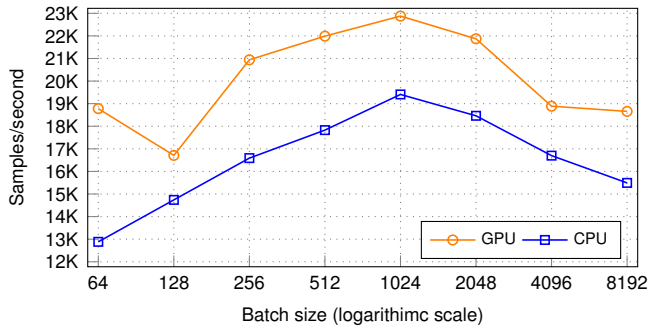

Fig. 5. Inference performance on the NVIDIA Jetson TX2 board.

Fig. 5 shows the performance of LUCID on the development board in terms of processed samples/second. As the shape of each sample is $[n, f] = [100, 11]$, i.e. each sample can contain the features of up to 100 packets, we can estimate that the maximum number of packets per second (pps) that the device can process without the GPU and using a batch size of 1024 samples is approximately 1.9 Mpps. As an example, the content of the UNB201X test set is 602,547 packets distributed over 22,416 samples, which represents a processing requirement of 500 Kpps without the GPU, and 600 Kpps when the GPU is enabled. This illustrates the ability to deploy LUCID on a resource-constrained platform.

The second measurement regarding resource-constrained systems is the memory requirement to store all the samples collected over a time window. The memory occupancy per sample is 8,800 bytes, i.e. $100 \cdot 11 = 1100$ floating point values of 8 bytes each. As per Fig. 5, the CNN can process around 23K samples/second with the help of the GPU and using a batch size of 1024. To cope with such a processing speed, the device would require approximately 20 GB RAM for a $t = 100$ time window. However, this value greatly exceeds the typical amount of memory available on edge nodes, in general (e.g., 1 GB on Raspberry Pi 3 [76], 2 GB on the ODROID-XU board [77]), and on our device, in particular. Indeed, the memory resources of nodes can represent the real bottleneck in an edge computing scenario.

Therefore, assuming that our edge node is equipped with 1 GB RAM, the maximum number of samples that can be stored in RAM is approximately 100K (without taking into account RAM used by the operating system and applications). We have calculated that this memory size would be sufficient for an attack such as the HTTP-based DDoS attack in the CSECIC2018 dataset, for which we measured approximately 30K samples on average over a 100 s time window. For more aggressive attacks, however, a strategy to overcome the memory limitation would be to configure the CNN model with lower values of t and n. For instance, setting the value of both parameters to 10 can reduce the memory requirement by a factor of 100, with a low cost in detection accuracy (F1 score 0.9928 on the UNB201X test set, compared to the highest score obtained with $t = n = 100$, i.e. 0.9946). The dynamic configuration of the model itself is out of scope of this work.

The measurements based on our test datasets demonstrate that the LUCID CNN is usable on a resource-constrained platform both with respect to processing and memory requirements. These results are promising for effective deployment of LUCID in a variety of edge computing scenarios, including those where the nodes execute latency-sensitive services. A major challenge in this regard is balancing between resource usage of LUCID (including traffic collection and preprocessing) and detection accuracy, i.e. ensuring the required level of protection against DDoS attacks without causing delays to the services. A deep study of this trade-off is out of scope of this paper and is reserved for future work.

B. Training time

In a real-world scenario, the CNN model will require re-training with new samples of benign and malicious traffic to update all the weights and biases. In edge computing environments, the traditional approach is to send large amounts of data from edge nodes to remote facilities such as private or commercial datacentres. However, this can result in high end-to-end latency and bandwidth usage. In addition, it may raise security concerns, as it requires trust in a third-party entity (in the case of commercial cloud services) regarding the preservation of data confidentiality and integrity.

A solution to this issue is to execute the re-training task locally on the edge nodes. In this case, the main challenge is to control the total training time, as this time determines how long the node remains exposed to new DDoS attacks before the detection model can leverage the updated parameters.

To demonstrate the suitability of our model for this situation, we have measured the convergence training time of LUCID on the development board using the UNB201X training and validation sets with and without the GPU support. We have experimented by following the learning procedure described in Sec. III-C, thus with a training termination criterion based on the loss value measured on the validation set. The results are presented in Table XI along with the performance obtained on the server used for the study in Sec. IV-D.

As shown in Table XI, the CNN training time on the development board without using the GPU is around 2 hours (184 epochs). This is approximately 4 times slower than training on the server, but clearly outperforms the training time of our implementation of DeepDefense 3LSTM, which we measured at more than 1000 sec/epoch *with* the GPU (i.e., 40 times slower than LUCID under the same testing conditions).

TABLE XI
TRAINING CONVERGENCE TIME.

Setup	Time/epoch (sec)	Convergence time (sec)
LUCID Server	10.2	1880
LUCID Dev. board (GPU)	25.8	4500
LUCID Dev. board (CPU)	40.5	7450
3LSTM Dev. board (GPU)	1070	>90000

In application scenarios where a faster convergence is required, the time can be further reduced by either terminating the training process early after a pre-defined number of epochs, or limiting the size of the training/validation sets. As adopting one or both of such strategies can result in a lower detection accuracy, the challenge in such scenarios is finding the trade-off between convergence time and detection accuracy that meets the application requirements.

VIII. CONCLUSIONS

The challenge of DDoS attacks continues to undermine the availability of networks globally. In this work, we have presented a CNN-based DDoS detection architecture. Our design has targeted a practical, lightweight implementation with low processing overhead and attack detection time. The benefit of the CNN model is to remove threshold configuration as required by statistical detection approaches, and reduce feature engineering and the reliance on human experts required by alternative ML techniques. This enables practical deployment.

In contrast to existing solutions, our unique traffic pre-processing mechanism acknowledges how traffic flows across network devices and is designed to present network traffic to the CNN model for online DDoS attack detection. Our evaluation results demonstrate that LUCID matches the existing state-of-the-art performance. However, distinct from existing work, we demonstrate consistent detection results across a range of datasets, demonstrating the stability of our solution. Furthermore, our evaluation on a resource-constrained device demonstrates the suitability of our model for deployment in resource-constrained environments. Specifically, we demonstrate a 40x improvement in processing time over similar state-of-the-art solutions. Finally, we have also presented an activation analysis to explain how LUCID learns to detect DDoS traffic, which is lacking in existing works.

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER project).

REFERENCES

[1] Krebs on Security, "DDoS on Dyn Impacts Twitter, Spotify, Reddit," https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit, 2016, [Accessed: 31-Oct-2019].

[2] Radware, "Memcached DDoS Attacks," https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/memcached-under-attack/, 2018, [Accessed: 31-Oct-2019].

[3] IETF Network Working Group, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," https://tools.ietf.org/html/bcp38, 2000, [Accessed: 31-Oct-2019].

[4] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," in *Proc. of SMARTCOMP*, 2017.

[5] M. Ghanbari and W. Kinsner, "Extracting Features from Both the Input and the Output of a Convolutional Neural Network to Detect Distributed Denial of Service Attacks," in *Proc. of ICCI*CC*, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[8] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed, and R. Klette, "Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes," *Computer Vision and Image Understanding*, vol. 172, pp. 88 – 97, 2018.

[9] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. of EMNLP*, 2014.

[10] B. Alipanahi, A. Delong, M. Weirauch, and B. J Frey, "Predicting the sequence specificities of dna- and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, 07 2015.

[11] D. Quang and X. Xie, "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences," *Nucleic Acids Research*, vol. 44, no. 11, pp. e107–e107, 2016.

[12] O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. V. de Walle, and S. V. Hoecke, "Convolutional Neural Network Based Fault Detection for Rotating Machinery," *Journal of Sound and Vibration*, vol. 377, pp. 331 – 345, 2016.

[13] A. Vilamala, K. H. Madsen, and L. K. Hansen, "Deep Convolutional Neural Networks for Interpretable Analysis of EEG Sleep Stage Scoring," *Proc. of MLSP*, 2017.

[14] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. Joon Ahn, "Deep android malware detection," in *Proc. of CODASPY*, 2017.

[15] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, March 2019.

[16] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. of ICOIN*, 2017.

[17] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park, "Flow-based malware detection using convolutional neural network," in *Proc. of International Conference on Information Networking*, 2018.

[18] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated Vulnerability Detection in Source Code Using Deep Representation Learning," in *Proc. of ICMLA*, 2018.

[19] K. Wu, Z. Chen, and W. Li, "A Novel Intrusion Detection Model for a Massive Network Using Convolutional Neural Networks," *IEEE Access*, vol. 6, pp. 50 850–50 859, 2018.

[20] S. Potluri, S. Ahmed, and C. Diedrich, "Convolutional Neural Networks for Multi-class Intrusion Detection System," in *Proc. of MIKE*, 2018.

[21] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proc. of ICACCI*, 2017.

[22] M. Abdelaty, R. Doriguzzi-Corin, and D. Siracusa, "AADS: A Noise-Robust Anomaly Detection Framework for Industrial Control Systems," in *Proc. of ICICS*, 2019.

[23] P. Criscuolo, "Distributed denial of service, tribe flood network 2000, and stacheldraht CIAC-2319, Department of Energy Computer Incident Advisory Capability (CIAC)," *UCRLID-136939, Rev*, vol. 1, 2000.

[24] H. A. Herrera, W. R. Rivas, and S. Kumar, "Evaluation of Internet Connectivity Under Distributed Denial of Service Attacks from Botnets of Varying Magnitudes," in *Proc. of ICDIS*, 2018.

[25] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical Approaches to DDoS Attack Detection and Response," in *Proceedings DARPA Information Survivability Conference and Exposition*, 2003.

[26] P. Bojović, I. Bašičević, S. Ocovaj, and M. Popović, "A practical approach to detection of distributed denial-of-service attacks using a hybrid detection method," *Computers & Electrical Engineering*, vol. 73, pp. 84–96, 2019.

[27] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint Entropy-Based DDoS Defense Scheme in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2358–2372, Oct 2018.

[28] S. B. I. Shah, M. Anbar, A. Al-Ani, and A. K. Al-Ani, "Hybridizing entropy based mechanism with adaptive threshold algorithm to detect ra flooding attack in ipv6 networks," in *Computational Science and Technology*. Singapore: Springer Singapore, 2019, pp. 315–323.

[29] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1545–1559, 2018.

[30] J.-H. Jun, C.-W. Ahn, and S.-H. Kim, "Ddos attack detection by using packet sampling and flow features," in *Proc. of the 29th Annual ACM Symposium on Applied Computing*, 2014.

[31] S. Yu, W. Zhou, R. Doss, and W. Jia, "Traceback of DDoS Attacks Using Entropy Variations," *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[32] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed ddos detection mechanism in software-defined networking," in *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015.

[33] M. E. Ahmed, S. Ullah, and H. Kim, "Statistical application fingerprinting for ddos attack mitigation," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1471–1484, 2019.

[34] J. Wang, L. Yang, J. Wu, and J. H. Abawajy, "Clustering Analysis for Malicious Network Traffic," in *Proc. of IEEE ICC*, 2017.

[35] Y. Yao, L. Su, and Z. Lu, "DeepGFL: Deep Feature Learning via Graph for Attack Detection on Flow-Based Network Traffic," in *Proc. of IEEE Military Communications Conference (MILCOM)*, 2018.

[36] E. Min, J. Long, Q. Liu, J. Cui, , and W. Chen, "TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest," *Security and Communication Networks*, 2018.

[37] J. Cui, J. Long, E. Min, Q. Liu, and Q. Li, "Comparative Study of CNN and RNN for Deep Learning Based Intrusion Detection System," in *Cloud Computing and Security*, 2018, pp. 159–170.

[38] M. Roopak, G. Yun Tian, and J. Chambers, "Deep Learning Models for Cyber Security in IoT Networks," in *Proc. of IEEE CCWC*, 2019.

[39] S. Homayoun, M. Ahmadzadeh, S. Hashemi, A. Dehghantanha, and R. Khayami, "BoTShark: A Deep Learning Approach for Botnet Traffic Detection," in *Cyber Threat Intelligence*. Springer, 2018, pp. 137–153.

[40] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. of IEEE symposium on security and privacy*, 2010.

[41] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[42] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for ddos detection," *Applied Intelligence*, vol. 48, no. 10, pp. 3193–3208, 2018.

[43] K. J. Singh, T. Khelchandra, and T. De, "Entropy-Based Application Layer DDoS Attack Detection Using Artificial Neural Networks," *Entropy*, vol. 18, p. 350, 2016.

[44] Z. He, T. Zhang, and R. B. Lee, "Machine learning based DDoS attack detection from source side in cloud," in *Proc. of CSCloud*, 2017.

[45] K. S. Hoon, K. C. Yeo, S. Azam, B. Shunmugam, and F. De Boer, "Critical review of machine learning approaches to apply big data analytics in DDoS forensics," in *Proc of ICCCI*, 2018.

[46] R. Primartha and B. A. Tama, "Anomaly detection using random forest: A performance revisited," in *Proc. of ICoDSE*, 2017.

[47] A. Koay, A. Chen, I. Welch, and W. K. G. Seah, "A new multi classifier system using entropy-based features in ddos attack detection," in *Proc. of ICOIN*, 2018.

[48] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, 2017.

[49] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proc. of IEEE CISDA*, 2009.

[50] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Proc. of MilCIS*, 2015.

[51] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *Proc. of IEEE ICDCS*, 2018.

[52] J. Song, H. Takakura, and Y. Okabe, "Description of Kyoto University Benchmark Data," http://www.takakura.com/Kyoto_data/BenchmarkData-Description-v5.pdf, [Accessed: 31-Oct-2019].

[53] C. Callegari, S. Giordano, and M. Pagano, "Statistical network anomaly detection: An experimental study," in *Proc. of FNSS*, 2016.

[54] CAIDA, "DDoS Attack 2007 Dataset," https://www.caida.org/data/passive/ddos-20070804_dataset.xml, 2019, [Accessed: 31-Oct-2019].

[55] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, 2012.

[56] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *Proc. of ICISSP*, 2018.

[57] The Canadian Institute for Cybersecurity, "CSE-CIC-IDS2018 dataset," https://www.unb.ca/cic/datasets/ids-2018.html, 2018, [Accessed: 31-Oct-2019].

[58] Imperva, "HOIC," https://www.imperva.com/learn/ application-security/high-orbit-ion-cannon, 2019, [Accessed: 31-Oct-2019].

[59] Imperva, "LOIC," https://www.imperva.com/learn/ application-security/low-orbit-ion-cannon, 2019, [Accessed: 31-Oct-2019].

[60] The Guardian, "Thousands download LOIC software for Anonymous attacks - but are they making a difference?" https://www.theguardian.com/technology/blog/2010/dec/10/hackers-loic-anonymous-wikileaks, 2010, [Accessed: 31-Oct-2019].

[61] The Canadian Institute for Cybersecurity, "Datasets," https://www.unb.ca/cic/datasets/index.html, 2019, [Accessed: 31-Oct-2019].

[62] Keras-team, "Keras: Deep Learning for humans," https://github.com/keras-team/keras, 2019, [Accessed: 31-Oct-2019].

[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016.

[64] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. of ICLR*, 2014.

[65] A. Jacovi, O. Sar Shalom, and Y. Goldberg, "Understanding convolutional neural networks for text classification," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 56–65. [Online]. Available: https://www.aclweb.org/anthology/W18-5408

[66] V. Bukac, "Traffic characteristics of common dos tools," *Masaryk University, Technical report FIMU-RS-2014-02*, pp. 74–78, 2014.

[67] I. Corona, G. Giacinto, and F. Roli, "Adversarial Attacks Against Intrusion Detection Systems: Taxonomy, Solutions and Open Issues," *Inf. Sci.*, vol. 239, pp. 201–225, 2013.

[68] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On Evaluating Adversarial Robustness," *CoRR*, vol. abs/1902.06705, 2019.

[69] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014.

[70] H. Chang and A. Hari and S. Mukherjee and T. V. Lakshman, "Bringing the cloud to the edge," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014.

[71] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *USENIX Security Symposium*, 2017.

[72] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese, "Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case," *IEEE Access*, vol. 7, 2019.

[73] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *Proc. of ACM CoNEXT*, 2018.

[74] NVIDIA Corporation, "NVIDIA Jetson TX2 Series datasheet," http://developer.nvidia.com/embedded/dlc/jetson-tx2-series-modules-data-sheet, 2018, [Accessed: 31-Oct-2019].

[75] NVIDIA Corporation, "cuDNN Developer Guide," https://docs.nvidia.com/deeplearning/sdk/pdf/cuDNN-Developer-Guide.pdf, 2019, [Accessed: 31-Oct-2019].

[76] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2019, [Accessed: 31-Oct-2019].

[77] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "ENORM: A Framework For Edge NOde Resource Management," *IEEE Transactions on Services Computing*, 2018.