Distributed real-time SlowDoS attacks detection over encrypted traffic using Artificial Intelligence

Norberto Garcia, Tomas Alcaniz, Aurora González-Vidal, Jorge Bernal Bernabe, Diego Rivera, Antonio Skarmeta

PII: S1084-8045(20)30336-2

DOI: https://doi.org/10.1016/j.jnca.2020.102871

Reference: YJNCA 102871

To appear in: *Journal of Network and Computer Applications*

Received Date: 3 April 2020

Revised Date: 25 August 2020

Accepted Date: 13 October 2020

Please cite this article as: Garcia, N., Alcaniz, T., González-Vidal, A., Bernabe, J.B., Rivera, D., Skarmeta, A., Distributed real-time SlowDoS attacks detection over encrypted traffic using Artificial Intelligence, *Journal of Network and Computer Applications* (2020), doi: https://doi.org/10.1016/j.jnca.2020.102871.

Credit author statement

Norberto Garcia: Investigation, Software, Data curation, Writing- Original draft preparation

Tomas Alcaniz: Data curation, Software, Writing- Original draft preparation

Aurora Gonzalez-Vidal: Investigation, Methodology, Writing- Original draft preparation, Supervision

Jorge Bernal Bernabe: Conceptualization, Investigation, Methodology, Writing- Original draft preparation, Supervision

Diego Rivera: Software, Writing- Original draft preparation

Antonio Skarmeta: Conceptualization, Writing- Reviewing and Editing, Supervision

Distributed real-time SlowDoS attacks detection over encrypted traffic using Artificial Intelligence

Norberto Garcia[a], Tomas Alcaniz[a], Aurora González-Vidal[a], Jorge Bernal Bernabe[a,*], Diego Rivera[b], Antonio Skarmeta[a]

[a]*Department of Information and Communication Engineering, University of Murcia, Facultad de Informatica, 30100, Murcia, Spain*
[b]*Montimage, 75013, Paris, France*

Abstract

SlowDoS attacks exploit slow transmissions on application-level protocols like HTTP to carry out denial of service against web-servers. These attacks are difficult to be detected with traditional signature-based intrusion detection approaches, even more when the HTTP traffic is encrypted. To cope with this challenge, this paper describes and AI-based anomaly detection system for real-time detection of SlowDoS attacks over application-level encrypted traffic. Our system monitors in real-time the network traffic, analyzing, processing and aggregating packets into conversation flows, getting valuable features and statistics that are dynamically analyzed in streaming for AI-based anomaly detection. The distributed AI model running in Apache Spark-streaming, combines clustering analysis for anomaly detection, along with deep learning techniques to increase detection accuracy in those cases where clustering obtains ambiguous probabilities. The proposal has been implemented and validated in a real testbed, showing its feasibility, performance and accuracy for detecting in real-time different kinds of SlowDoS attacks over encrypted traffic. The achieved results are close to the optimal precision value with a success rate 98%, while the false negative rate takes a value below 0.5%.

Keywords: cybersecurity, artificial intelligence, cyberattacks, machine learning

1. Introduction

Denial-of-Service-(DoS) based attacks are continuously evolving increasing its complexity and range, thereby making more and more difficult to perform a timely and accuracy detection. Traditional DoS attacks, that aims to incapacitate a resource from serving its genuine clients, have been intensively studied in the literature through different schemes intended to protect network infrastructures [1]. DoS attacks, concretely application-layer DoS attacks are recently getting research attraction [2], since they are able to compromise a web-server through other means beyond traditional ones such as network flooding or exhausting server's resources such as sockets, memory, CPU, and I/O bandwidth. In particular, SlowDoS attacks [3] are a type of application-layer DoS using low-rate packet transmission [4]. Namely, most of the SlowDoS attacks, such as Slowris or SlowPost, exploits HTTP protocol, widely adopted in application-layer services, by sending incomplete http requests, or keeping the connection with the server busy through sending the HTTP posts using slow ratio and without reaching content-length values.

Furthermore, the prominence of the Internet of Things [5] is introducing millions of interconnected network devices which suggest new scenarios and connectivity models that are increasing the attack surface. Since neither high-performance systems nor immense network bandwidth are needed to perform a SlowDoS attack, simple constrained IoT devices are suitable to carry out a denial. Indeed, the biggest advantage of SlowDoS attacks relies on the scarce bandwidth needed, meaning few IoT attackers –i.e bots– using a low-rate packet transmission can overwhelm their victim.

SlowDoS attacks are arduous to be detected using traditional signature-based intrusion detection systems, as it becomes hard to inspect and match an attack signature [6] affecting legitimate application-level HTTP requests. Anomaly-based Network Intrusion Detection Systems (NIDS) analyze incoming network traffic in order to detect any invasive or illegitimate behaviour. This approach performs properly for detecting not only known attacks, but also unknown or "zero-day" attacks [6], [7]. There are some research works that focus on detecting SlowDoS attacks using probability distributions [8] or [9] that analyzes HTTP messages. However, those proposals are not applicable over encrypted application traffic or have been not properly validated in a real testbed dynamically monitoring and detecting SlowDoS in real-time.

To fill this gap, this paper proposes a real-time, Artificial Intelligence (AI)-based system for dynamic anomaly-

*Corresponding author
Email addresses: norberto.garcia@um.es (Norberto Garcia), t.alcanizcascales@um.es (Tomas Alcaniz), aurora.gonzalez2@um.es (Aurora González-Vidal), jorgebernal@um.es (Jorge Bernal Bernabe), diego.rivera@montimage.com (Diego Rivera), skarmeta@um.es (Antonio Skarmeta)

based detection of application-level SlowDoS attacks over encrypted traffic. Our system monitors in real-time the network traffic, processing and aggregating packets in conversation flows, extracting additional meaningful network features as statistics that are dynamically analyzed in streaming for distributed AI-based anomaly detection. To this aim, our AI model, running in a distributed way in Apache Spark-streaming, combines clustering analysis and deep learning techniques to increase accuracy in attack detection. This system consists of two algorithms aimed to capture normal behavior patterns and detect anomalies. The first one is based on clustering models and the second on exploits Deep Learning techniques intended to detect attacks with ambiguous probabilities in the clustering.

The proposed anomaly-detection system – and its associated AI model – has been implemented and extensively validated in a real testbed by running different kinds of SlowDoS attacks. These tests show its performance to monitor, aggregate and analyze real-time network traffic in a distributed way, as well as the capability of detecting in streaming the attacks run over application-level encrypted traffic with high precision average accuracy 0.98 for different SlowDoS attacks.

The rest of this paper is structured as follows. Section 2 describes the related work and Section 3 the background in Intrusion Detection System (IDS) and SlowDos attacks. Section 4 presents the AI-Based security framework. Section 5 delves into the AI-based model. Extensive performance and accuracy evaluation is done in section 6. Finally, section 7 concludes the paper.

2. Related work

There are several recent surveys that analyze the application of artificial intelligence mechanisms [10], on the security of machine learning in malware C&C detection [11, 12, 13], for anomaly detection [14] and specific surveys [15] that focuses on detecting DoS attacks and enrich defense mechanisms using AI. Another survey [6] introduces a heuristic-based detection system for scripting languages such as javascript, that is able to detect malicious code using machine learning approaches. Thus, this method substantiates the inefficiency of signature-based IDS to detect "zero-day" attacks. In [16] a queue management algorithm is proposed as the first line of defense for a network when facing a DDoS attack.

One of the applications of anomaly-based IDS in cybersecurity is the article presented by Diro and Chilamkurti [17], that proposed a distributed deep learning based Fog-computing detection system. The experiment was motivated by the rapidly increasing of IoT devices and the need to be protected from new variants of attacks. They demonstrated that distributed techniques work better than centralized algorithms in order to efficiently detect cyber-attacks due to the sparse nature of IoT deployments. Other research work also used cloud computing to implement an intrusion detection system. Alzahrani and Hong proposed a hybrid signature and anomaly-based IDS over a cloud computing scenario using deep learning techniques [18]. Apache Spark implemented the artificial intelligent system that is capable of detecting intruders. The joint of these two IDS approaches with the distributed capabilities of Spark showed the effectiveness of the anomaly-based IDS in contrast to signature-based. Despite cloud computing is a good approach for the timely detection of cyber-attacks, they do not follow any real-time approach in order to mitigate them in a high-performance scenario.

Similar approaches have been discussed on [19], proposing mixing signature and anomaly based detection. This paper presents machine learning techniques (Neural Networks [20]) for developing a flow-based NIDS. Moreover, they proposed the integration of this system for Software-Defined Networks (SDN) since flows can be easily blocked to prevent attacks. The flow statistics are sent to a SDN switch by a controller over a certain time interval. When the statistics are available, they are used to detect anomaly behaviour. The detected malicious traffic is mitigated through flow modifications using SDN techniques. Likewise, [21] introduces a NIDS for SDN-based, cloud IoT networks, which has three layers of IDS nodes (Edge-IDS, Fog-IDS and Cloud-IDS). This particular IDS has an effective collaboration among its nodes and makes use of machine learning/deep learning methods for detecting network threats from IoT devices, making this proposal suitable for real-time scenarios. Similarly, [22] propose using SDN for intrusion detection and use an AI-based, two-stages IDS enhanced by SDN technologies.

Unsupervised anomaly detection is widely adopted in many practical applications [23, 24], including the detection of intrusions. For example in [25] they quantitatively compare a pool of twelve unsupervised anomaly detection algorithms on five attacks datasets. A variant of Support Vector Machines is proposed in [26] together with several outlier detection algorithms for system call intrusion detection. Also, in [27] the points that lie in sparse regions of the feature space are considered anomalies in system call traces. The feature maps are based on data-dependent normalization and a spectrum kernel. [28] propose an online and scalable unsupervised network method. Finally, [29] proposes a methodology to identify relations between attack families, anomaly classes and algorithms in order to select an unsupervised algorithm that maximizes detection capability.

All these works propose techniques for cyber-attack detection, however, none of them focus on encrypted traffic. Moreover, they do not target Slow DoS attacks – which is one of the main threat in IoT scenarios – and do not include a real-time analysis.

Other papers discuss a different detection system. Alauthman et al. [30] a reinforcement-based botnet detection approach that proposes a traffic reduction method in order to deal with a high volume of network traffic. Cusack et al. [31] focus on Slow Dos/DDoS attack detection using a distance based metric for evaluating the similarity be-

tween current and benchmark logs. The approach focuses on mobile network since these network devices are rapidly increasing. Likewise, in [32] authors introduce a method for detecting SlowDoS attacks extrapolating data referred to real traffic traces. A similar approach can be found in [4], focusing on detection without packet inspection which analyzes specific spectral features. These works describe good methods for classification and detection but they can not work with encrypted traffic.

Regarding research work focused on real-time scenarios, in [9] authors present a flow-based intrusion detection system using streaming technologies. Moreover, they focus on DoS attacks and use Spark Streaming for their stream processing-based IDS. AI methods are used in order to identify an attack. Extracted features are obtained using the stream methods that Spark Streaming provides. They had used Naïve Bayes, Logistic Regression and Decision Tree as the AI algorithms capable of classifying typical DDoS attacks such us TCP flooding, UDP flooding and ICMP flooding. Despite the authors propose a close to a real-time IDS for detecting cyber-attacks successfully, they do not focus in SlowDoS nor encrypted attacks. Similarly, in [33] authors use Naïve Bayes Classifications and due to the distributed agents on the multi-agent system, the total load is distributed among all the participants in the network.

The article [34] proposes an anomaly-based IDS focused on Application-layer DDoS attacks with encrypted traffic. They convert network traces into conversations with the desired features for applying machine learning and deep learning methods. The features contain statistic- and time-related characteristics which works successfully with clustering methods. This paper actually resolves the problematic of encrypted traffic but has not been tested in a distributed and real-time setting, monitoring in streaming the attacks, as it is done herein. In addition, our model outperforms that paper interms of accuracy at will be shown in section 6

Despite the ML and DL algorithms used for detecting anomalies, a proper training dataset has to be used for training the algorithm. The NSL-KDD dataset is made of several cyber-attacks and contains features that are useful for detecting with an anomaly-based approach [35], however it lacks SlowDoS attacks. Moreover, it includes features from the application layer, unable to work if the payload is encrypted. The IDS-2017 dataset follows a similar approach and none of them aggregate specific features from SSL/TLS connections, so they would be problematic for encrypted traffic. These datasets seem to work for traditional cyber-attack, but it is necessary to generate a new dataset for SlowDoS attacks over encrypted traffic.

3. Background

3.1. Intrusion Detection Systems (IDS)

Traditional *Intrusion Detection Systems (IDS)* follow a signature-based approach to detect DoS, monitoring ongoing network traffic in real-time and looking for sequences or patterns that may match an attack signature previously introduced. Signatures can be identified based on packet headers and network addresses, that contain sequences of data that are known to be a particular familiar attack. Signature-based IDS like *Snort* a widely-used solution since it is based on open source software and works effectively at detecting common attacks. Although this solution has excellent results with attacks already known (patterns available in the signature database), it can not detect every cyber-attack successfully, moreover recognizing HTPP patterns over encrypted traffic is very complex. Besides, using traditional IDS, unknown attacks (zero-day) go through the network infrastructure unnoticed, and only when they are discovered and pushed its signature up to the database can be detected. Therefore, during this process, network devices remain exposed and the consequences may be overwhelming.

The lack of fruitful results in some attacks led to the rise of anomaly-based IDS that focuses on monitoring behaviours that do not follow a regular genuine client behaviour, instead of checking signatures belonging to an attacker. Moreover, this kind of IDS provides alerts about a cyber-attack that has never been seen before, also known as "zero-day" attacks. In order to classify a behaviour as normal or anomalous, it uses heuristics approaches.

3.2. Slow Rate Denial-of-Service against Web Applications

DoS attacks aim to exhaust a resource and make it unavailable to its legitimate clients. The strategy followed by an attacker consists in forcing the victim (resource) to remain in a saturation state, incapable of serving user requests. Many of them have a great impact on communication systems, being one of the most popular threats.

In SlowDoS attacks, the attacker aims to force the victim to process only the illegitimate requests, although they are seen as genuine. If an attacker fills the victim service queue, any genuine client will not be served, as the connection is refused due to the lack of resources in the server, hence leading to a denial of service condition. To sidestep security systems based on statistical detection – mainly used by signature-based approaches – a low-rate bandwidth mechanism is used. Unlike traditional DoS attacks such as *Ping-of-death* or *SYN flood*, where thousand of meaningless packets are sent to exhaust the victim due to high-rate, in a SlowDoS every packet is essential to accomplish the attack goal.

Some SlowDoS attacks, also known as Slow HTTP Denial-of-service, are growing briskly. Some kind of this particular attack use GET requests to overstep the server's connections limit – usually 150 parallel connections in Apache servers. Attackers' goal is to prevent other users from using the website and allowing themselves to establish multiple connections to the server. Unlike traditional DoS attacks, this can be launched using only one system

with multiple fake clients or bots sending requests in a low-rate [4].

When an attack is planned to be performed, attackers seek out security holes in protocols to use a service illegitimately. In HTTP servers, there is a built-in threat: connection is only freed up when a complete header is received. This means that attackers can use this vulnerability to perform a SlowDos attack. In this scenario an attacker can send a simple request and the connection/socket will not be closed until the request is completed. Sending incomplete requests at a slow-ratio with a large number of clients takes up all the available connections of a web server. If an attacker maintains open a large number of concurrent connections, the web server will keep waiting for the end of the connections, reaching a DoS condition.

According to [36], SlowDoS attacks to web applications can be categorized in three types : (i) *pending requests DoS*, based on sending incomplete requests, (ii) *long responses DoS*, sending genuine requests that slow down the server's responses, and (iii) *multi-layer DoS*, that do not fully act in application-layer but use a slow-rate mechanism to perform a DoS.

Currently, although there are a numerous variety of SlowDoS attacks, this paper focuses on SlowDoS to Web Applications. In *Pending Request DoS*, attacks are based on sending incomplete requests to the server which becomes busy. The following attacks can be found: *Slowloris*, known as Slow Headers or Slow HTTP GET, was designed by Robert "RSnake" Hansen. As described before, the attacker sends incomplete HTTP requests in order to keep the socket occupied. The slow sending is accomplished sending a specific string repeatedly: *X-a: b\r\n*. This process is followed by multiple fake clients until every available connection becomes allocated and therefore the web server unreachable.

SlowPost, also known as R.U.D.Y. (R U Dead Yet?) or SlowBody. Attackers take advantage of HTTP POST to send forms. The post request alerts the web server that a protracted piece is about to be sent. In this attack, the Content-Length value is used to specify the size of the message body, forcing the server to wait for the complete message. The attacker sends this form in a very slow ratio without reaching the Content-Length value, usually about one byte each packet. Sending these packets to the server keep the connection busy.

Regarding *Long Responses DoS*, legitimate requests are sent but are designed to slow down the server response.In this category several SlowDoS attacks are introduced. *Apache range headers* relies on the byte range parameter of HTTP which is used to obtain a resource [3]. This attack force an Apache Server to create a lots of copies of a specified file. However, it is not effective anymore if the system is up to date. Other SlowDoS attack known as *HashDoS*, attackers aim to exploit a vulnerability related to hash tables' performance. They force the server to generate a lot of collisions, slowing down its performance. The *ReDoS* attack focus on a vulnerability related to regular expressions. The attacker sends *evil regexes* causing the server to slow down, as it is validating the expression.

Finally, *Multi-layer DoS* focus on operating not only at the application layer but use network and transport either. The *SlowRead* attack was proposed by Serge Shekyan of Qualys Security Labs. It aims at sending a genuine HTTP request to the server, delegating the main part to a lower layer (TCP). In this attack, all the available connection are occupied and the servers reply to the attacker in a slow-rate. SlowRead is performed at transport-layer specifying the size of a custom window. TCP Maximum Segment Size encapsulated in an ethernet frame is 1448 bytes which is the size of packets in most communications. In this attack, the server is forced to use a reduced window size, specifying it in the initial SYN packet. This will make the server reply at a slow rate. Due to the large number of connections it would lead to a DoS. Although this attack uses TCP methods to exploit a vulnerability and therefore it is classified as *Multi-layer Dos*, it may be either classified as *Long Response DoS* since the server response is slowed down.

Another SlowDoS attack can be found in this category. *LoRDAS*, also known as Low-Rate DoS against Application Servers, has been proposed by Maciá-Fernández [37]. In this attack the attacker aims to identify the instants in which the server free up its resources to occupy them afterwards.

The low bandwidth requirements of these attacks make IoT devices suitable for this environment. Moreover, if attacks are performed over encrypted traffic, it would be arduous to match a signature on the HTTP headers and messages. Therefore, an anomaly-based approach for detecting, as proposed in this paper, may cope with this problematic.

4. AI-based cyber-attacks detection framework for application-level encrypted traffic

This section describes the AI-based framework aimed to detect in real-time SlowDoS attacks over encrypted application-level traffic. The infrastructure is an anomaly-based Network Intrusion Detection System (NIDS) that continuously monitors the network traffic, inspecting and aggregating traffic in real-time and following an AI-based approach for detecting DoS-based attacks over encrypted traffic. Furthermore, our framework, unlike traditional signature-based IDS, has the ability to detect attacks over encrypted traffic and it paves the way to detect "zero-days" attacks.

The detection of these kind of DoS-based attacks includes the analysis of conversations between a web server and its clients. Using artificial intelligence methods, our system builds a normal user behaviour model that divides conversations into clusters and examines distribution amongst them, in order to determine whether or not the system is being attacked. To fulfill this objective we have designed and developed a distributed framework based on

3 layers: (1) Real-time Network Monitoring, (2) Conversation Processing, (3) Artificial Intelligence Methods.

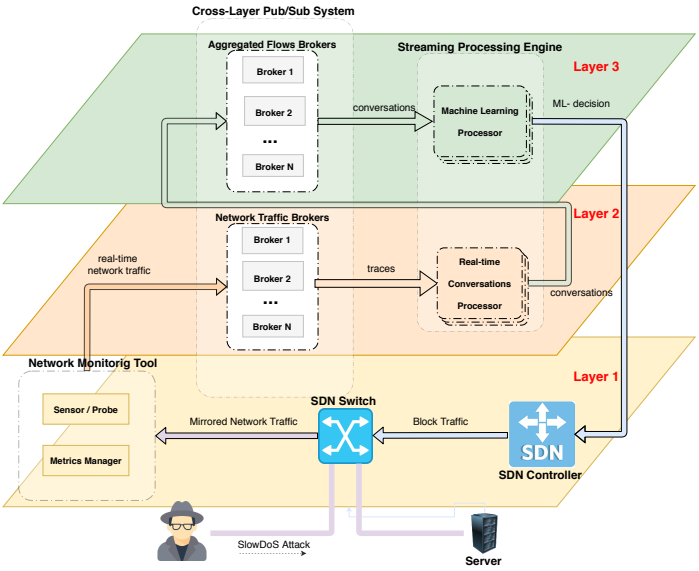

Figure 1: Framework Design

Figure 1 shows the infrastructure design proposal and its general flow. The three main components are connected through a publish/subscriber system in order to fulfill real-time requirements and cope with scalability requirements. Layer (1) includes the distributed "Network Monitoring Tool" needed inspect the traffic. Layer (2) deals with Real-Time network Conversation processing and feature extraction, and layer (3) embraces distributed artificial intelligence engines to detect anomalies. Additionally, these layers are based on streaming technologies, being capable of processing input data in real-time and produce timely detection.

4.1. Real-time Network Monitoring

In order to obtain the raw data from the network flows, it is required to use real-time network monitoring. The market provides several tools that allow monitoring the network in real-time and extract the required data by the proposed approach. For example, *nDPI* [38] provides support to a large number of protocols and applications to extract metadata, however it does not provide a performance suitable for real-time applications due to an overhead incurred when classifying traffic [39]. Library-based solutions (like *PACE2* [40], *libprotoident [41]* or *libpcap*) provide a programmatic way to access the flow data of the network traffic. Despite this gives flexibility, it also introduces the requirement of implementing a real-time network sniffer that uses these libraries and sends the extracted data to the platform for its further evaluation.

Despite these limitations, the market also offers probe solutions that can sniff packets and extract the required information. *QosmosixEngine* [42] is a proprietary tool that enables to directly sniff the packets and generate network events related with the information of the flows. The same approach is implemented by *suricata*, but with the advantage of integrating an Intrusion Detection/Prevention System (IDS/IPS) used to test security properties using the extracted information. Despite these advantages, the tools provide a non-convenient output mechanism, based on APIs and logging respectively. To make these tools usable in the proposed approach, it introduces the requirement of developing an adaption layer capable of reading the input and feed continuously to the pub/sub system of the architecture. In this sense, it was crucial to find a monitoring solution that can ease the deployment of the proposed approach.

The Montimage Monitoring Tool (MMT) [43], follows a modular approach to extract network statistics and feed them into a temporal logic analysis engine that performs temporal logic analysis for security analysis. Using its Deep Packet Inspection (DPI) module, the MMT tool is capable of monitoring the packets going through the network, and extracting a set of network-related data from each packet. Additionally, MMT supports a set of different outputs for the extracted data and security analysis, being able to write the information on logs (as ixEngine and Suricata) but also directly on pub/sub systems such as Apache Kafka.

Considering direct support for Kafka and the real-time monitoring capabilities of MMT, it was used in the platform to actively monitor the network traffic and extract data using its DPI engine. Using its DPI features, MMT was configured to generate a report per packet that contains the following information: (1) Timestamp, (2) Source IP, (3) Destination IP, (4) Source TCP Port, (5) Destination TCP Port, (6) IP Total Length, (7) TCP Window Size, (8) Time-to-live, (9) TCP FIN Flag, (10) TCP SYN Flag, (11) TCP RST Flag, (12) TCP PSH Flag, (13) TCP ACK Flag, (14) TCP URG Flag, (15) TLS Content Type. Additionally, MMT was also configured to publish these reports directly on the architecture's pub/sub system.

To ease the integration with other tools (and the next steps of the behaviour analysis), an extra layer was inserted, in order to change the MMT proprietary format into a standard CSV. Figure 1 depicts this in the *Real-time Conversations Processor*. The result of this process is published back to the general broker, to make the information available for the next step.

It is important to remark that these features are extracted from each packet, and they do not suffice to determine the presence of a cyber-attack, since they present packet metadata rather than flow statistics. To derive conversation-related information, it is needed to aggregate the data into conversations, grouping packets that belong to the same flow (IP addresses, port numbers, and protocol) into a single aggregated metadata about the flow under analysis.

4.2. Conversation Processing

The *Conversation Processor* component in layer 2 receives the features extracted by the MMT (transformed into a standard format) in the first layer, processing and aggregating them into conversations as well as extracting meaningful statistics.

Our conversation processor implementation maintains the initial features extracted from the packets, but also add numerous time-related statistics: (1) Uplink and Downlink IP[1], (2) Uplink and Downlink Port[1], (3) Duration in microseconds, (4) Number of packets sent in one second in uplink and downlink, (5) Number of bytes sent in one second in uplink and downlink, (6) Maximal, minimal and average packet size in uplink and downlink, (7) Maximal, minimal and average size of TCP window in uplink and downlink, (8) Maximal, Minimal and average time-to-live (TTL) in uplink and downlink, (9) Percentage of packets with different TCP flags (FIN, SYN, RST, PSH, ACK, URG) in uplink and downlink, (10) Percentage of packets with different properties (chgcipher, alert, handshake, appdata, heartbeat) in uplink and downlink, (11) Number of new connections to the same destination host as the current connection in the last 5 seconds, (12) Number of new connections to the same destination host as the current connection that connect to the same service in the last 5 seconds, (13) Percentage of new connections from the current host which have the same destination service in the last 5 seconds, (14) Percentage of new connections from the current host which have different destination service in the last 5 seconds, (15) Total number of active connections to the same destination host as the current connection, (16) Total number of active connections to the same destination host as the current connection that connect to the same service, (17) Percentage of active connections from the current host which have the same destination service, (18) Percentage of active connections from the current host which have different destination service. These features are shown in detail in table 1.

As shown in the same table, features from 40 to 49 include information about encrypted SSL/TLS connections, which makes our approach different from other state of the art works using public available datasets such as NSL-KDD or IDS-2017, that do not focus on this type of network traffic. Specifically, some datasets including NSL-KDD take advantage of application-layer signatures to aggregate features, thus being unable to work over encrypted traffic. To cope with this problem, our feature-extraction approach does not examine application level and leverages on lower layers, including the aforementioned features to incorporate encrypted statistics. Hence, our framework builds a model tailored to cope with SlowDoS attacks over encrypted traffic.

These features provide us aggregated statistics about traffic and flow information which greatly increase classification results. The information reported by raw network packets do not suffice to classify their purpose. Indeed, normal behavioural flow patterns distinctly differ from anomalous models, thus the selected extracted flow-based features address to detect SlowDoS attacks. The features include statistical time-related information of the stream of packets set up between a client and a web server, providing representative data for further analysis. It is not possible to extract similar characteristics from raw network packets without flow considerations. Furthermore, characteristics of SSL/TLS packets that are taken into consideration –eg. features from 40 to 49– make this model suitable for encrypted traffic since they report differences between legitimate and anomalous flows in HTTPs.

It is worth mentioning that features from 1 to 4, which include information about IPs and ports, are not used for training the model. They only provide information about flow identification to enforce a proper mitigation action.

Once those aggregated statistics are computed successfully, they are sent again to the pub/sub system to be analyzed by the ML module in layer 3. Since conversations are made by joining different packets during a concrete period, the *Conversation Processor* defines two conditions for submitting to the layer 3 for further analysis. If a conversation ends successfully (TCP Connection Termination), it is closed, sent immediately and removed from the *Conversation Processor*. Otherwise, if a conversation remains open, it is continuously sent at a specific time interval[2] and continues open. In this case, a conversation is closed and entirely removed from the *Conversation Processor* if no packet is received at other intervals. This policy allows conversation processing to run smoothly without overloading resources, making this stage suitable for real-time environments.

4.3. Applying Artificial Intelligence Techniques

The layer 3 of the framework is intended to perform the AI-based attack detection. As we have described earlier, our AI model combines neural networks and clustering to classify whether a conversation monitored in layer 1 , and aggregated in layer 2, is abnormal or does not belong to a genuine client.

Conversations are received from layer 2 through the pub/sub system and analysed in a distributed way by different instances of the *Machine Learning* component that are launched in parallel. The detection process analyzes the communication behaviours that are genuine, in such a way that those communications that differ from these normal behaviour are considered as suspicious. Normal behavior patterns are learned by both, clustering algorithms

[1]Using the MMT DPI engine, the "uplink" IP and port are recognized from the host that sent the first SYN packet to start the communication.

[2]New packets may not be classified to the conversation between adjacent time intervals. In order to increase performance, the conversation is sent only if a new packet arrived.

Feature	Description	Feature	Description
F1	Uplink IP	F26	Minimal time-to-live (TTL) in Downlink
F2	Downlink IP	F27	Average time-to-live (TTL) in Downlink
F3	Uplink Port	F28	Percentage of packets with FIN TCP Flag in Uplink
F4	Downlink Port	F29	Percentage of packets with SYN TCP Flag in Uplink
F5	Duration in microsenconds	F30	Percentage of packets with RST TCP Flag in Uplink
F6	Number of packets sent in one second in Uplink	F31	Percentage of packets with PSH TCP Flag in Uplink
F7	Number of packets sent in one second in Downlink	F32	Percentage of packets with ACK TCP Flag in Uplink
F8	Number of bytes sent in one second in Uplink	F33	Percentage of packets with URG TCP Flag in Uplink
F9	Number of bytes sent in one second in Downlink	F34	Percentage of packets with FIN TCP Flag in Downlink
F10	Maximal packet size in Uplink	F35	Percentage of packets with SYN TCP Flag in Downlink
F11	Minimal packet size in Uplink	F36	Percentage of packets with RST TCP Flag in Downlink
F12	Average packet size in Uplink	F37	Percentage of packets with PSH TCP Flag in Downlink
F13	Maximal packet size in Downlink	F38	Percentage of packets with ACK TCP Flag in Downlink
F14	Minimal packet size in Downlink	F39	Percentage of packets with URG TCP Flag in Downlink
F15	Average packet size in Downlink	F40	Percentage of packets with Chgcipher property in Uplink
F16	Maximal size of TCP window in Uplink	F41	Percentage of packets with Alert property in Uplink
F17	Minimal size of TCP window in Uplink	F42	Percentage of packets with Handshake property in Uplink
F18	Average size of TCP window in Uplink	F43	Percentage of packets with Appdata property in Uplink
F19	Maximal size of TCP window in Downlink	F44	Percentage of packets with Heartbeat property in Uplink
F20	Minimal size of TCP window in Downlink	F45	Percentage of packets with Chgcipher property in Downlink
F21	Average size of TCP window in Downlink	F46	Percentage of packets with Alert property in Downlink
F22	Maximal time-to-live (TTL) in Uplink	F47	Percentage of packets with Handshake property in Downlink
F23	Minimal time-to-live (TTL) in Uplink	F48	Percentage of packets with Appdata property in Downlink
F24	Average time-to-live (TTL) in Uplink	F49	Percentage of packets with Heartbeat property in Downlink
F25	Maximal time-to-live (TTL) in Downlink		

Feature	Description
F50	Number of new connections to the same destination host as the current connection in the last 5 seconds
F51	Number of new connections to the same destination host as the current connection that connect to the same service in the last 5 seconds
F52	Percentage of new connections from the current host which have the same destination service in the last 5 seconds
F53	Percentage of new connections from the current host which have different destination service in the last 5 seconds
F54	Total number of active connections to the same destination host as the current connection
F55	Total number of active connections to the same destination host as the current connection that connect to the same service
F56	Percentage of active connections from the current host which have the same destination service
F57	Percentage of active connections from the current host which have different destination service

Table 1: Aggregated final features extracted by our Conversation Processor in layer (2)

and using Machine Learning. These algorithms help each other to diagnose the nature of a new conversation. The main advantage of this type of analysis is that it does not require to specify at any time what are the usual features of an attack, hence, it has great versatility in different types of threats, beyond SlowDoS attacks, including "zero-day" attacks.

These techniques are described in detail in section 5. Indeed, the obtained results that will shown in section 6.3 point that the ML model works successfully for detecting SlowDoS attacks with performance needed to carry out the detection in real-time.

4.4. Publisher/Subscriber System

Network traces and conversations are distributed through a publisher-subscriber system also known as *broker*. In software architecture, a pub/sub system is a messaging pattern where senders, known as publisher, send messages to a category instead of a specific receiver. Likewise, receivers are known as subscribers since they subscribe to a category instead of directly receive messages from senders. This kind of system is one of the main pillars of our architecture since all the layers make use of it. There are two categories or topics where data are stored. The first one includes network traces with the raw features extracted by MMT in its proprietary format. In this layer, the format transformation is also performed, publishing its results in a separate broker topic. The conversation processor is a subscriber to that topic, where initial features are transformed to the final aggregated statistics, and also publishes conversations to a final, independent topic. Finally, the artificial intelligence layer subscribes to the last topic, in order to process the aggregated information.

4.5. Streaming Processing

Streaming processing is a technology widely used for Big Data analysis and real-time applications. It can analyze the continuous stream of data in order to detect conditions and process them rapidly. This new paradigm introduces a new way to analyze data where queries and data flow in a continuous manner and react to explicit events.

The figure 1 depicts our proposed framework that takes advantage of streaming processing in layers (2) and (3): conversation processing and artificial intelligence techniques. By using this technology it is possible to receive data from multiple sources in a real-time manner, and aggregate them in a common channel that will be used by the machine learning algorithm.

As illustrated by Figure 2, our framework consists of 3 main components (Network Monitoring Tool, Conversation and Machine learning Processors). Network packets are extracted by the monitoring tool and sent to the conversation processor for its real-time grouping. Finally, the data is sent to the ML processor, in order to detect

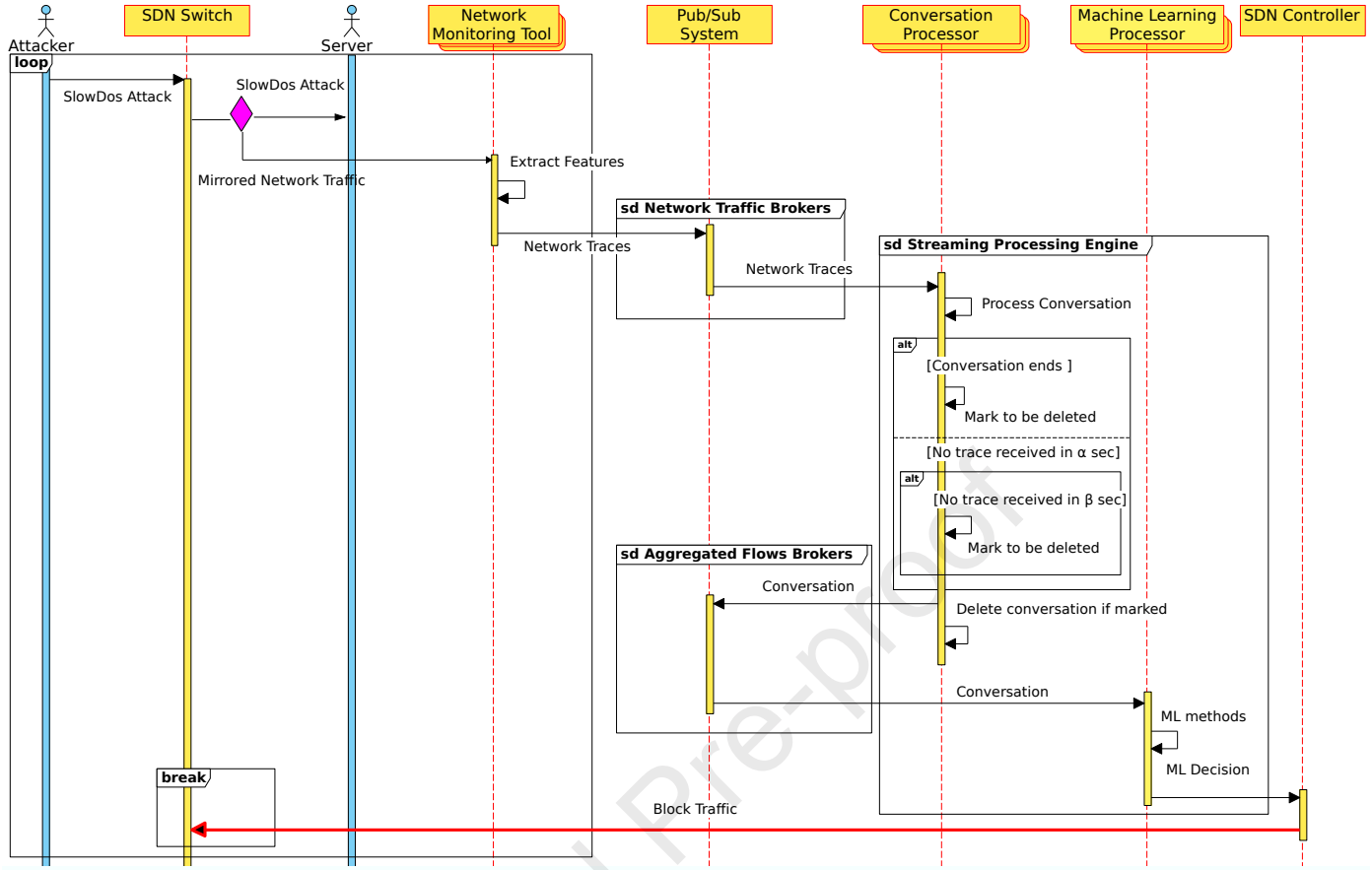

Figure 2: Overview of the interactions amongst the components of the framework

anomalies and potentially a SlowDoS attack. As depicted in the same figure, the communication among the components is provided by the publisher/subscriber system. This methodology allows a real-time delivery of the data. Once an attack is detected it can be dynamically counter, using an SDN approach. Namely, enforcing a *flow-mod* Openflow message from the SDN Controller to the switch adding a new rule in the switch to drop traffic from the attacker as shown in our previous work [44].

Figure 3 depicts a simplified view of conversation management during a period of time. Network packets are represented by different colors where red ones belong to an attacker, otherwise to normal clients. A conversation is built by gathering network packets belonging to the same socket (same color). When a network packet is received by the *Conversations Processor*, it is processed in T_a time. While no packet is received again, a time window T_w is opened and maintained where nothing is classified in that conversation. Then, if this window exceeds a timeout T_{ts}, the conversation is ready to be sent to the ML layer (see conversation 2, 3 and 4). The sending procedure relies on specific time windows in which either, finished or timed out conversations are sent, this window may coincide with T_{ts}. Likewise, if T_w exceeds the timeout T_{td}, the conversation is marked as finished and is completely deleted from the *Conversations Processor* cache. This timeout is always greater than the send timeout, i.e. $T_{td} > T_{ts}$.

Alternatively, a conversation may finish with an end-packet (following the TCP Connection Termination), as in conversation C_1, where it is sent just after receiving the final packet and utterly deleted afterwards. The machine learning layer ingest batches sequentially from the pub/sub system, so these batches may be made from multiple conversations depending on the batch interval. Conversation C_1 and C_2 are sent in the same batch interval, therefore a 2-conversation batch is set up.

The time taken by a batch in the queue is $T_{qf} - T_{q0}$, where T_{q0} represents the time when the batch enters the queue and T_{qf} when it exits. Then, batches are processed outputting a detection result in $T_{mlf} - T_{ml0}$. If a batch has multiple conversations, each conversation is distributed in parallel amongst the different worker nodes, increasing the processing performance, and avoiding delays in the queue processing time.

5. Artificial Intelligence for SlowDoS attacks detection

This section presents the developed AI model devised and optimized to detect SlowDoS attack. Our model con-

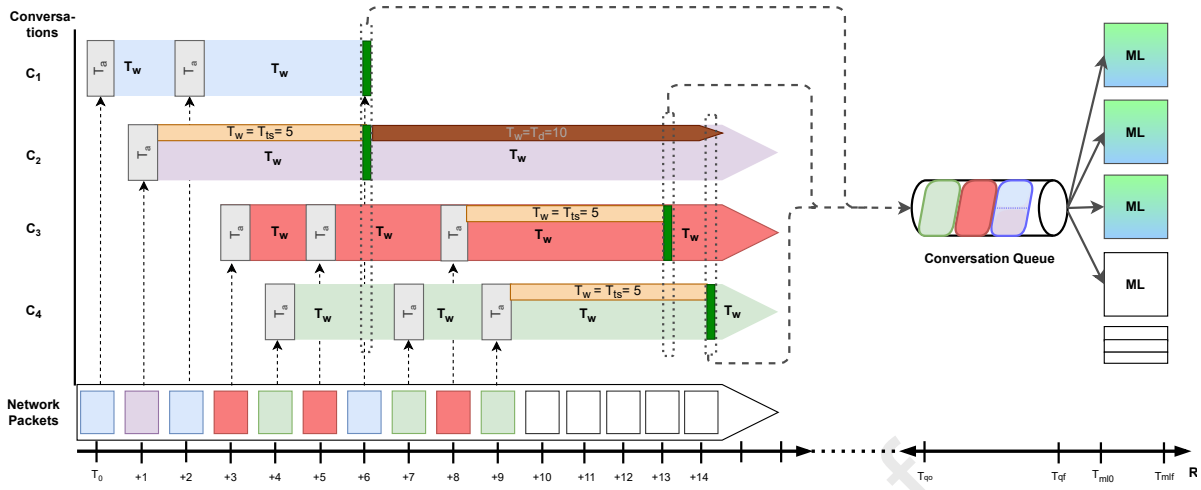

Figure 3: Representation of the Aggregator and Machine Learning layer timeline, where network packets are processed into conversation and sent to the ML layer

sists of an algorithm in five phases: preprocessing, clustering, histogram matrix, Deep Learning and detection. The AI-model combines clustering with deep learning techniques to increase overall accuracy. The clustering technique alone did not reach the desired level of precision, since this does not take into account aspects supported by DL module, such as the correlation between variables.

The use of clustering serves to determine the normal behavior patterns of the users, thereby obtaining a probability of belonging to a conversation to one of these groups. On the other hand, if the uncertainty were too great and it was not possible to determine whether or not a conversation belongs to one of these groups, then it would be necessary to use a neural network that allows finding the correlations between variables in order to have a second test.

However, before carrying out the training and detection process, it is necessary to carry out a pre-processing step, in which the variables are normalized and filtered, giving rise to a smaller number of variables. The main criterion used for this task is based on eliminating those variables that present very little variability and therefore do not provide useful information to the model.

5.1. Model training

This section details the method used to train the models that will be used to detect attacks: on the one hand, the clustering-based model and, on the other, the DL-based model.

5.1.1. Phase 1. Preprocessing

First of all, the relevant variables obtained in real-time from the pub/sub system, i.e. those identified in Section 4.2, are processed and normalized. Once this is done, it can be seen how the different types of communications are distributed in figure 4. It shows the difference between instances belonging to normal activity and attacks when they are represented according to the first principal components.

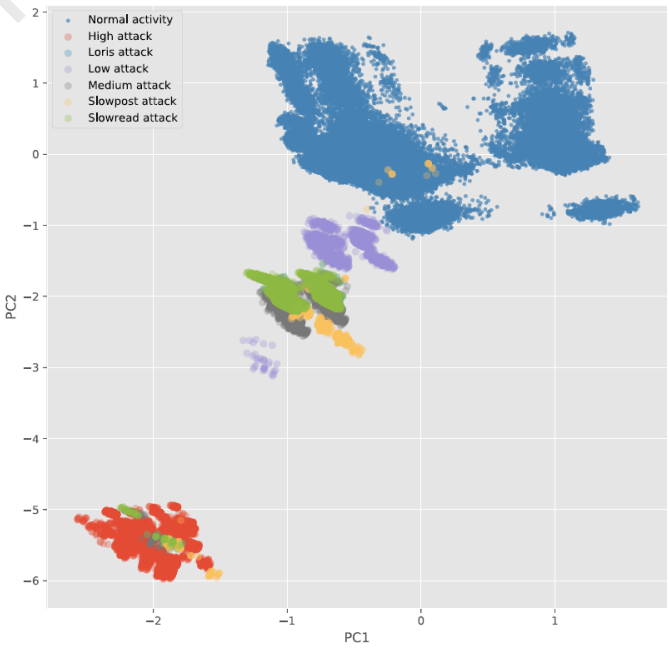

Figure 4: Representation of system traffic for normal activity (blue) and different kind of attacks for the two first principal components.

5.1.2. Phase 2. Clustering

Our attack detection algorithm is based on performing clustering of normal conversations in order to check whether a new conversation belongs to these clusters (normal activity) or not (attack). The clustering model selected is Gaussian Mixture Model (GMM) [45] as it usually achieves better quality results when compared to other

clustering methods as K-Means. As will be shown in section 6, our model has been compared with state of the art, that employed K-Means as attack detector mechanism.

Gaussian Mixture Models (GMM) have soft boundaries, where data points can belong to multiple clusters at the same time but with different degrees of belief. A Gaussian Mixture is a function composed by K individual Gaussian functions, where K is the number of types of behavior in which our data is grouped (clusters). Each Gaussian function has the following parameters: a mean μ that determines the center, a covariance Σ that determines the width of the function and a mixing probability π that defines the size of the Gaussian subject to $\sum_{k=1}^{K} \pi_k = 1$ and $0 < \pi_k < 1$. Therefore, the objective of the method is to find the optimal value of these parameters that best adapt to the probability distributions followed by the normal behavior data. In other words, to figure out which point comes from which Gaussian distribution. In order to do so, the Expectation Maximization (EM) algorithm is used and it consists in two stepts:

- Expectation: with the currently assigned means and variances, the probability of each data point i is computed as follows

$$r_{ik} = \frac{\pi_k \mathcal{N}(x_i|\pi_k, \sum_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\pi_j, \sum_j)} \tag{1}$$

  .

- Maximization: these probabilities are used to re-estimate the Gaussians' mean and variance in the current iteration (it) to better fit the data points as follows. Let $N_k^{it} = \sum_{i=1}^{N} r_{i,k}$, then $\pi_k = \frac{N_k^{it}}{N}$ shows how much each cluster is represented over all data points. Now we update

$$\hat{\mu_k} = \frac{1}{N_k^{it}} \sum_{i=1}^{N} r_{i,k} x_i \tag{2}$$

  and

$$\hat{\Sigma_k} = \frac{1}{N_k^{it}} \sum_{i=1}^{N} r_{i,k} (x_i - \hat{\mu_k})(x_i - \hat{\mu_k})^T \tag{3}$$

  .

Thus, GMM-based clustering provides the probabilities of membership of each communication to the set of clusters, in such a way that those with a high value will be taken as normal and those with a low value will be taken as abnormal. However, the main drawback of GMM related to our problem is that the central ranges of probabilities are controversial in order to make a decision. E.g when the probabilities are between 45% and 55% , GMM can lead us to missclasify a conversation For that reason, this clustering method requires the help of other Deep Learning tools that allow us to break the uncertainty of these ambiguous probabilities, explained in Phase 3 and Phase 4 described below.

5.1.3. Phase 3. Histogram matrix

During this phase, for each network conversation, the percentage of characteristics belonging to each of the clusters is computed. To do so, the standard deviations of the training data are calculated in such a way that a characteristic of a vector will be within a given cluster if it is within the range $[C_{j,k} - \sigma_k, C_{j,k} + \sigma_k]$ where $C_{j,k}$ is the cluster centroid coordinate j for the variable k. Notice, that it is a variation from the methodology of [34], because in that publication they compute the number of vectors belonging to a cluster considering a certain amount of vectors that depend on the time, instead of number of characteristics.

This calculation leads to a vector of size n (number of clusters) $h^i = (h_1^i, h_2^i, ..., h_n^i)$ where i refers to a specific communication. When joining each of the conversations, a matrix of size $n \times D$ where n is the number of clusters and D the number of conversations is obtained. This methodology offers a dynamism that was not found in [34] since it allows to construct histogram vectors and histogram matrices without taking into account the time factor. That is to say, communications can be taken one by one (as if it was an attack detector) or the average can be weighted to several nearby communications in time for group i and obtain an attacker detector.

5.1.4. Phase 4. Deep Learning Training

Once the histogram matrix has been constructed for the data belonging to normal conversations, these are used to train a deep learning technique. To this aim our AI-based attack detector uses an Auto-Encoder (AE), which is a neural network that consists of a visible input layer, a hidden layer, and a reconstruction layer. The last and first layer are of the same size. The objective is, therefore, to reproduce the input values at the output phase. For training, the histogram matrix of normal conversations is used. Only one hidden layer was considered because the literature supports the fact that one single hidden layer is enough for the vast majority of practical problems [46, 47], creating a fast solution that provides accurate results and its number of neurons was estimated using grid search with cross-validation.

Mathematically, an autoencoder consists of the encoder and the decoder, which can be generally defined as transitions $\phi : \chi \rightarrow \mathcal{F}$ and $\psi : \mathcal{F} \rightarrow \chi$, where $\phi, \psi = argmin_{\phi,\psi} |X - (\phi \circ \psi)X|^2$. The encoder function maps the original data x, to a latent space F, which is present at the bottleneck. The decoder function maps the latent space F at the bottleneck to the output. In a sense, we recreate the original data after some generalized non-linear compression. The encoding network can be represented by the standard neural network function passed through an activation function, where z is the latent dimension (where x is mapped to) $z = \sigma(Wx+b)$. The decoding network can be represented similarly, with other weight, bias, and potentially activation functions being used: $x' = \sigma(W'z+b')$. The objective is to minimize the mean square error between the input and output vector (reconstruction error),

so when the system has stored enough information will be able to reproduce the input on the output. This technique works on the assumption that there is a correlation between the variables so that these variables are very different for normal and anomalous conversations. These AEs can be stacked one after the other in order to minimize information loss.

Thus, in our case, for each row of the histogram matrix, the reconstruction error is given by

$$E^i = \sqrt{\sum_{j=1}^{nC} (H_j^i - \hat{H}_j^i)^2} \tag{4}$$

where H_j^i is the original entry and $\hat{H}_j^i$ is the reconstruction. These values are therefore used to compute a threshold $T^E = \mu_i^E + \omega\sigma_i^E$ where μ_i^E is the average value of the reconstruction error values, σ_i^E the deviation and ω is an hyperparameter.

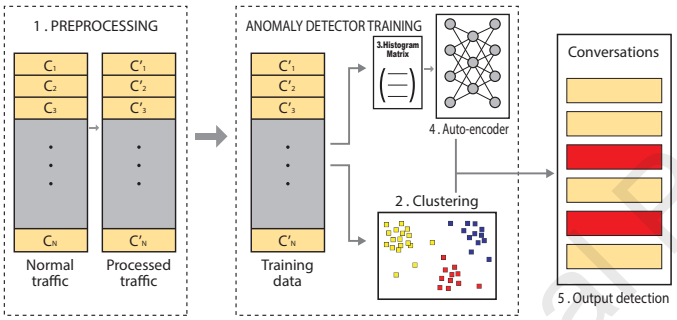

Figure 5: Architectural overview of the proposed ML-based training model

Figure 5 shows the process that follows the information from the conversations and shows the process flow of the model used in this paper and the associated phases. As a first step, the data belonging to normal activity is preprocessed (phase 1), to be used to train both a clustering method (phase 2) and an Auto-Encoder (phase 4) with the histogram matrix (phase 3). The latter helps the clustering algorithm to determine if a new conversation belonging to a test data set is an attack or normal activity, as described in the next section. We have called this method 'strengthened GMM" (sGMM) because it combines GMM with the computation of the percentage of characteristics and the use of an AE in order to obtain a finer result.

5.2. Attack detection

This section shows how the algorithm determines whether a new communication is taken as an attack or a normal communication using the already trained models.

When a client initiates a communication, it will be classified in one of the behavior groups, and its histogram vector h is calculated, encoded and decoded by the AE, so that its reconstruction error is computed. If the error is greater than the T_E threshold then the communication is classified as an anomaly.

Therefore, in order to break the uncertainty and improve the performance of the GMM model, the ambiguous communications are classified using the DL method. In order to determine the range of ambiguous probabilities, a grid search was carried out to determine the optimal value of P^+ and P^- , giving rise to the interval (P^-, P^+), in which the use of the Auto Encoder is necessary to determine the nature of the conversation.

Section 6.3 will show the results achieved for each of the models and their efficiency will be compared with different attacks settings and other clustering methods.

Data: User conversation: $\vec{r}$
Result: Nature of conversation: Attack/Normal
Variable filtering: $\vec{r} \rightarrow \vec{r}'$;
Evaluate the probability density of the GMM in $\vec{r}'$: $P(\vec{r}')$;
if $P(\vec{r}') \leq P^-$ **then**
  return Attack;
else if $P(\vec{r}') \geq P^+$ **then**
  return Normal;
else
  Calculate histogram vector: $\vec{h}$
  Calculate reconstruction error with input to AE ($\vec{h}$) and output ($\vec{h}'$): $E_{rec} = \sqrt{\sum_{i=1}^{nC}(h_i - h_i')^2}$
  if $E_{rec} \leq T^E$ **then**
    return Normal
  else
    return Attack

Algorithm 1: Pseudocode of the proposed sGMM algorithm

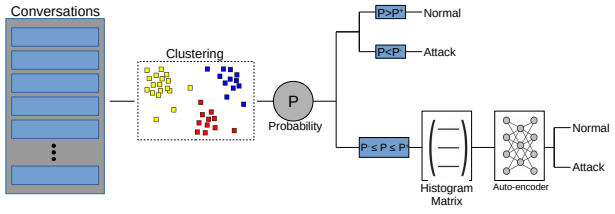

Figure 6: Architectural overview of the proposed ML-based attack detection model

Figure 6 shows the detection process: first, the probability P of belonging to a group is calculated, then this is compared with P^+ and P^- if it becomes sufficiently low or high enough that clustering is enough to determine whether a conversation is normal or an attack. However, when P takes a value between P^+ and P^- it will be necessary to calculate the histogram matrix and use the autoencoder to calculate the reconstruction error and thus determine the type of communication.

6. Implementation and Evaluation

This section introduces the tools used to implement the 3-layer framework presented previously, the process for obtaining our data-set and the evaluation. The data-set has been obtained using specific monitoring of diverse SlowDoS attacks in our streaming platform. Notice that actual security datasets public available (e.g. NSL-KDD), do not consider SlowDoS attacks. In addition, this section also presents the quality results focused on evaluating the accuracy of the AI-based model using the streaming dataset. Furthermore, this section presents the performance of the overall system in a real testbed with different attacks, including the time taken by different stages of the real-time attack detection framework.

6.1. System Implementation and testbed

Table 2 recaps the tools used associated which its specific layer as well as other implementation characteristics.

Pub/Sub System	Apache Kafka 2.3.0
Stream Processing	Spark Streaming 2.4.3
Network Monitoring Tool	Montimage Monitoring Tool
Overall Development Kit	Open JDK 12
ML Development Kit	Python Spark 2.4.3

Table 2: Framework Implementation tools

Network monitorization is carried out using a proprietary software called Montimage Monitoring Tool (MMT) [43]. This tool inspects mirrored traffic from the SDN switch, extracts initial features described in section 4, and publishes them in the Kafka Broker. The main feature of this software includes the ability of publishing features directly into Kafka without any other additional add-on or software. Additionally, multiple MMT instances may be used in order to analyse the network traffic at different points. Apache Kafka implements the pub/sub system which uses two topics for the messages. The first one manages network traces with initial features extracted by the monitoring tool while the other includes conversations with the final features.

Finally, our framework uses Apache Spark Streaming for stream processing (layer 3 of the architecture of Figure 1) which allows huge traffic analysis and an efficiently data processing in real-time. This software has been designed to be robust enough to withstand failures and not abort the whole processing procedure. Spark Streaming store intermediate data in RDD (Resilient Distributed Dataset) which is an immutable collection that can be operated in parallel. This technology provides a lot of transformations in its API, such us *map*() which maps each input element in the source stream to a new value, or *groupByKey*() which groups all key-values pairs with the same key together. This last operation is of paramount importance in our framework, as it is used in order to group network packets that belongs to the same socket. That means, spark streaming can parallelize conversation processing joining network traces in the same processor. Furthermore, we have tested multiple T_{ts} to figure out which time perform better. Using a timeout between 5 and 10 seconds, our infrastructure perform at its best. Therefore, since timely results fits better with a real-time scenario, a T_{ts} of 5 seconds has been chosen. Besides, the timeout of deleting T_{td} was set to 30 seconds, as it can be considered an unusual long period for a conversation without a new packet arriving.

Conversation processing was implemented using the Java language whereas machine learning procedures were implemented in Python. Furthermore, ML module uses PySpark module in order to get a final result. This module executes its processing method in parallel using a broadcast variable which contains several operations to classify a conversation as genuine or anomalous. Using the operation *repartiton*(), conversations are distributed efficiently amongst the worker nodes and then, they can use the broadcast variable through *foreachPartition*() method, decreasing the queue and processing time.

The conversation processor follows a different approach. It uses a lazy-initialized pool of flow processors running in Spark, creating one per worker node to take care of network packets in parallel.

6.2. Generated Dataset description

This section defines the dataset obtained by monitoring and processing network traffic at layer 2 in our framework, thereby obtaining the network traffic features explained in Section 4.2. For collecting different datasets for various attacks and settings, particular software and tools were employed. Two categories of datasets have been generated: normal and anomalous communication (i.e. SlowDoS attacks). In order to gather genuine real-time data, we have implemented a traffic generator leveraging *JMeter* for normal HTTPs traffic and *slow-http-test* to launch the different SlowDoS attacks. In this scenario, to obtain normal traffic needed to train the machine learning model, our application was stopped at the ML module and its output (all generated conversations) were written into a standard CSV file. This dynamic procedure generates real-time conversations which fit better to our model rather than an static implementation.

For carrying out the SlowDoS attack we ran the tool *slow-http-test*[3] software using TLS connections. To measure the performance of the machine learning model in a real scenario, several types of SlowDoS attacks have been launched. These attacks belong to *pending requests DoS* (Slowloris and SlowPost) and *multi-layer DoS* (SlowRead) categories introduced in section 3.2. No SlowDoS attack from *long reponses DoS* was included since they depend on the server implementation and do not entail a global threat. For instance, Apache Range Headers is not a hazard nowadays if the server is up-to-date. Others such as

[3]URL of the project: https://github.com/shekyan/slowhttptest/

HashDoS or ReDoS need some type of input to perform a lookup/insertion in the database or introduce a regular expression if they are available. Hence, they do not represent an actual risk for every webserver. .Furthermore, the effectiveness of these attacks depends on the specific server implementation and the number of parallel connections it can handle. Therefore, three datasets have been generated using a different amount of clients and concurrent connections.

The first one – known as "High-connections", comprises a lot of connections (up to 3000), represents a heavy and direct attack causing a denial of service in a few seconds. Notice that SlowDoS are characterized by causing deny of service with few connections, so considering having more than 3000 connections simultaneously are not representative of these kind of attacks. Another intense attack called "Medium-connections", eventually causes a denial of service but uses less connections (up to 1000). The last one, "Low-connections" (up to 500), occupies a few connection in order to reproduce a lighter attack which do not manage to perform a denial of service. All of these attack implementations were configured to encrypt the HTTPs connections. In order to validate the feasibility and performance of our system under different conditions, the experiments were conducted using several kind of SlowDoS attacks. Concretely, the kind of attacks used were: Slowloris, SlowPost and SlowRead, described in section 3.2. As shown in table 3, the attacks have the following structure:

- High-connections attack: Tries 3000 connections to the server (1000 per attack).
- Medium-connections attack: Tries 1000 connections to the server (333 per attack).
- Low-connections attack: Tries 500 connections to the server (166 per attack).

Besides these parameters, we have used an interval between follow up data of 3 and 20 seconds respectively for Slowloris and SlowPost. Likewise, for SlowRead attacks, a interval of 5 seconds is established between read operations. Furthermore, the maximum length of follow up data in Slowloris is 10 bytes and 24 in Slowpost. Regarding SlowRead, we used 512 bytes as the start of the advertised TCP window size and 1024 bytes as the end. Moreover, 2 bytes are received in every read operation and the resource is requested 2 times per socket. The SlowDoS attacks parameters used are summarized in table 3. It should be noticed that we have tested other different SlowDoS configurations beyond those parameters, but the quality results using other configurations does not influence in the accuracy of the classification model.

On the other hand, regarding normal traffic, we have simulated genuine clients behaviour using Apache JMeter during 1 hour against an Apache https server set up in the testing network. This software is able to reproduce legitimate behavioural patterns that we have designed, in total there are 600 clients connected simultaneously and they are renewed every iteration. These patterns include 15 different behaviours against a website using GET and POST requests, waiting a random interval of time between each operation as a genuine client would do. Moreover, clients are distributed in multiple virtual machines to setup a real environment, including both normal and encrypted connections. The Apache server was configured to accept at most 1600 connections in parallel in order to handle heavy number attacks and a high-clients scenario.

	High connections Dataset	**Medium connections Dataset**	**Low connections Dataset**
SlowPost connections	1000	333	166
Slowloris Connections	1000	333	166
SlowRead connections	1000	333	166
Total connections	3000	1000	500
Connections per second (at most)	1000	333	166
Denial of Service	Yes	Eventually	No

	SlowRead
Interval between read operations (seconds)	5
Start of advertised TCP windows size (bytes)	512
End of advertised TCP windows size (bytes)	1024
Bytes received every read operation	2
Times resource is requested (per socket)	2

	Slowloris	**SlowPost**
Interval between follow up data (seconds)	3	20
Maximum length follow up data (bytes)	10	24

Table 3: Attacks and main parameters

The aforementioned configurations were used to generate a dataset used to build the normal behaviour model and train the machine learning model. Figure 7 represents the average bandwidth and network packets against clients in our scenario. Since genuine traffic is generated using a broad set of patterns, bandwidth and network packets suffer of high/low peaks. Therefore the figure depicts an average measure.

6.3. Quality Results

This section introduces the results obtained by the classifier. The metrics used to evaluate the quality of the attack detection models were both, Receiver Operating Characteristics (ROC) curve and Area Under the Curve (AUC). For this purpose, a 10-fold validation was performed using the 25% of randomly chosen test data, respecting a balance between normal and anomalous communications. To evaluate the accuracy of our proposed model, we have compared our model with the state of the

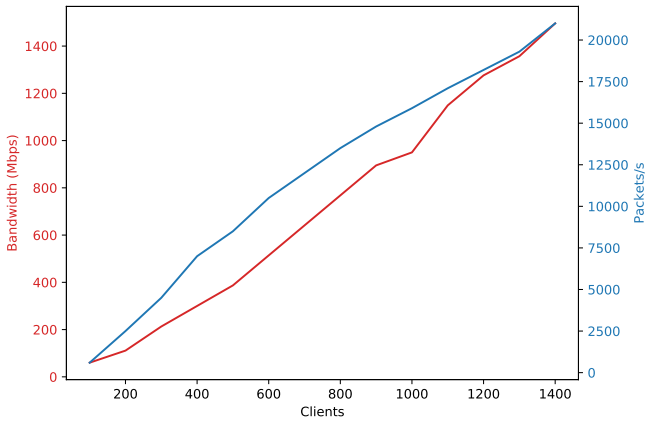

Figure 7: Relation amongst genuine clients, bandwidth and network packets sniffed by the monitoring tool

art. In particular, our AI model called sGMM, that combines GMM and DL, is compared with the state of the art [34] that employs K-Means clustering based on thresholds to detect the attacks.

ROC curve is a performance measurement for classification problem. It is generated by plotting true positive rate (TPR) against false positive rate (FPR). Likewise, AUC describes the model sensitivity at distinguishing different classes.

In the previous section three general anomalous datasets with different amount of connections were introduced. The metric obtained are depicted in figures 8, 9, 10. Although these datasets contain multiple anomalous behaviour, a metric for each individual attack is evaluated. The attacks includes: Slowloris, SlowPost and SlowRead. Furthermore, they share the configuration of the medium-connections attack. Their performance is shown in figures 11, 12, 13.

As can be seen in the figures 8, 9 and 10 the performance of the strengthened GMM model is superior to the GMM model, therefore, in the following datasets both models have not been compared.

The results show that the performance of our strengthened GMM model (sGMM) is superior to K-Means in every dataset according to the AUC metric. Additionally, figure 14 depicts the AUC of the GMM model every 5 seconds since the attack was launched. Three types of a Slowloris attack have been evaluated. They used a different interval between follow up data: 1 , 3 and 10 seconds. The performance at second 10 reached 98.2%. Nonetheless, since T_{ts}=5, the attack with interval 10 suffers of peaks of low entropy where only a few conversations are received. This behaviour is clearly shown, as the performance eventually decrease and recovers when more conversations arrive. Despite this behaviour, the performance of the model remains stable when there are available data and different configurations do not affect to the overall detection performance.

Besides to AUC, another quality metric has been obtained for the different models and datasets. The confusion matrices collected are shown in table 4, in which the average values of TP, FN, FP and TN are described for each dataset and model.

	High K-Means	High sGMM
TP	42.57	50.00
FN	7.42	0.00
FP	7.59	1.62
TN	42.4	48.37
	Medium K-Means	Medium sGMM
TP	43.24	50.02
FN	6.78	0.00
FP	7.43	1.62
TN	42.59	48.39
	Low K-Means	Low sGMM
TP	45.91	50.00
FN	4.08	0.00
FP	7.62	1.74
TN	42.37	48.25
	SlowPost K-Means	SlowPost sGMM
TP	47.91	49.27
FN	2.08	0.72
FP	36.21	1.60
TN	13.78	48.39
	SlowRead K-Means	SlowRead sGMM
TP	42.17	50.00
FN	7.82	0.00
FP	7.68	1.58
TN	42.31	48.41
	Slowloris K-Means	Slowloris sGMM
TP	44.98	50.00
FN	5.01	0.00
FP	7.62	1.62
TN	42.37	48.37

Table 4: Percentage confusion matrix of the models for the different datasets.

In addition, the quantities collected in confusion matrices can be used to calculate the Accuracy, Precision, Recall, F1-score and Specificity metrics, which provide information beyond the success rate [23]:

- Accuracy is the ratio of the correctly labeled subjects to the whole pool of subjects.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

- Precision is the ratio of the correctly positive labeled subject to all positive labeled subjects.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

- Recall is the ratio of the correctly positive labeled to all who are positive in reality.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

- F1-score is the harmonic mean(average) of the precision and recall.

$$F1 - Score = \frac{2 \cdot (Recall \cdot Precision)}{(Recall + Precision)} \tag{8}$$

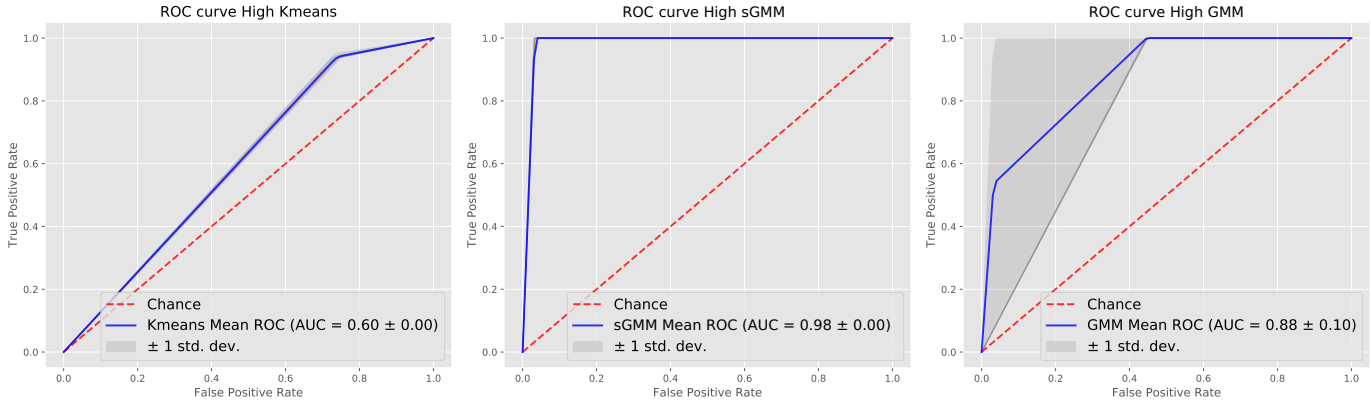

Figure 8: ROC curve for model K-Means(left), strengthened GMM (center) and GMM (right), using high-connections

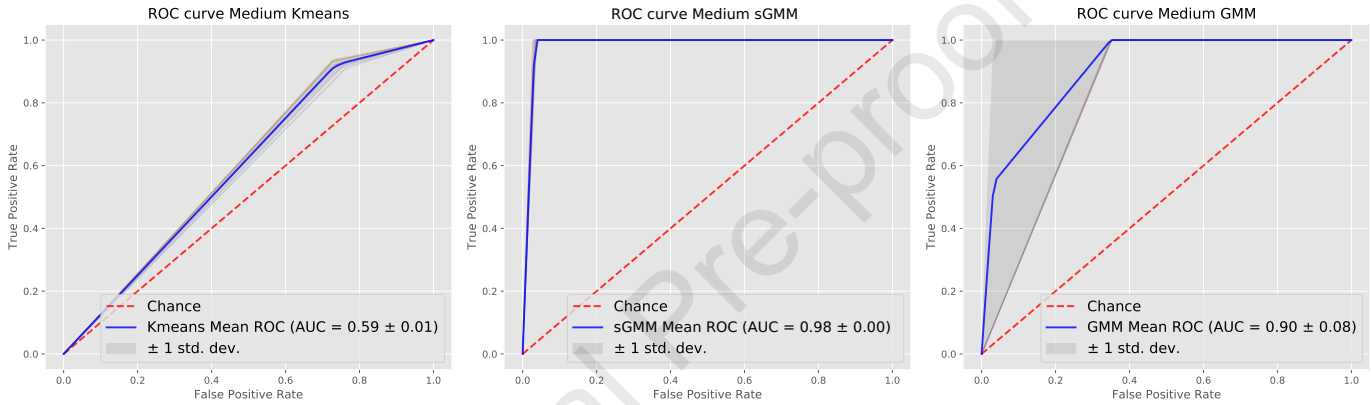

Figure 9: ROC curve for model K-Means(left), strengthened GMM (center) and GMM (right), using medium-connections setting

- Specificity is the correctly negative labeled by the algorithm to all who are negative in reality.

$$Specificity = \frac{TN}{TN + FP} \tag{9}$$

The measurements extracted for metrics are shown in the table 5.

The information collected in the table 5 shows that our GMM-based model outperforms the compared work, having greater robustness and stability in its metrics for different datasets. This implies that it is able to adapt to more types of attacks and thus obtain better accuracy for different scenarios.

6.4. System Performance Results

In this section we present the performance of our system in an isolated environment. For building our scenario we have used multiple virtual machines sharing the same network topology.

1. Clients: 3 machines. 8GB memory and 8 cores.
2. Apache Kafka: 1 machine. 8GB memory and 16 cores.
3. Webserver and Network Monitoring: 1 machine. 16GB memory and 16 cores.
4. Streaming Processor: 2 machines. 8GB memory and 16 cores each. It includes layer (2) and (3).

Every virtual machine CPU is running at 2.1 GHz and have a network bandwidth of 16 Gbps. The batch interval is set to 1 second, meaning Spark starts computation every second and gather all records into the same batch to be processed as a task. Moreover, backpressure is enabled, which allows Spark to monitor the batch delays and processing times in order to optimize and receive data as fast as it can process it.

To evaluate the performance of detection time in streaming, we have build a test software that starts normal communications during a period of time and eventually launch an SlowDoS attack. Timestamps are obtained in different stages to measure the time required. This software is a python script capable of communicating with every machine to launch multiple processes. Indeed, it is able to distribute attackers and normal clients among the different available machines. Besides, in order to mitigate attacks and obtain the detection time, another topic was included in Kafka in which the results of the machine learning module are sent. The script reads from this topic and stops the attack when it is detected. Moreover, we have launched attacks equivalent in number with the configuration of the

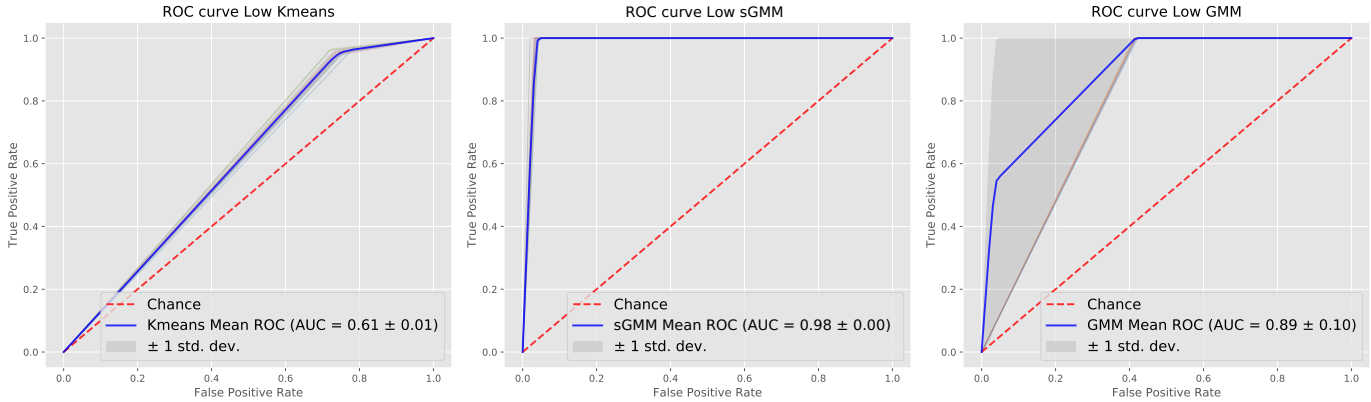

Figure 10: ROC curve for model K-Means(left), strengthened GMM (center) and GMM (right), using low-connections setting

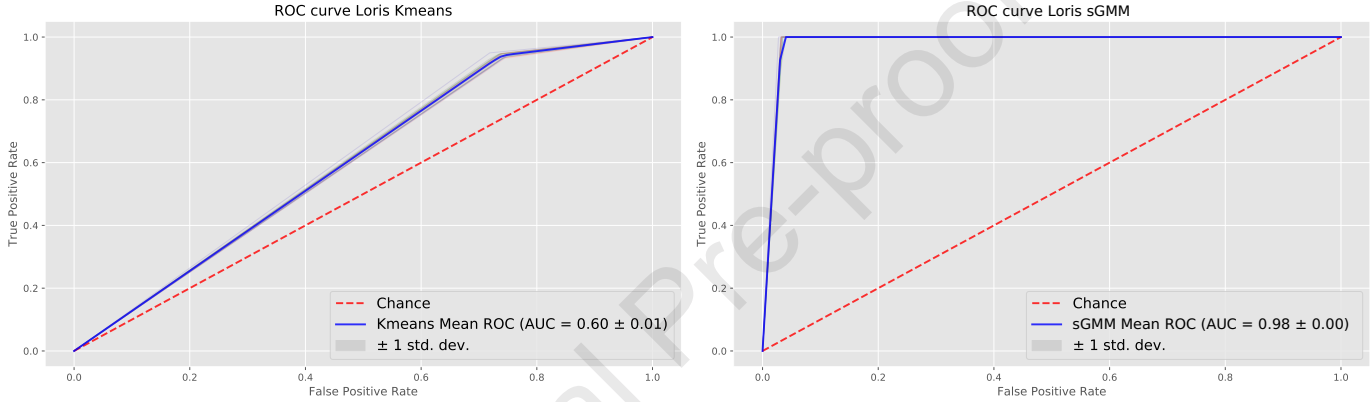

Figure 11: ROC curve for model K-Means(left) and strengthened GMM (right). Slowloris attack

medium-connection setting, including Slowloris, SlowPost and SlowRead attacks. Furthermore, 1000 normal clients were accessing the website in parallel. In order to measure the detection time, this test has been run 50 times. We have obtained that the average time required for detecting is 9.8 seconds with a deviation of ±2.4 seconds in some tests. This time includes every stage of our application: network monitoring, conversation processing and machine learning classification. This mean performance time is broken-down in figure 15, that depicts the time required by layers 2 (Conversation Processor) and 3 (ML module) to process and obtain a result. In this scenario we have tested a number of clients ranging from 50 to 1450, using an increment of 50. As can be seen, conversation processing does not require a huge amount of time and its queue time is minimal, around 1 to 10 milliseconds, whereas processing time grows to 450 ms at 1450 clients.

Additionally, the machine learning layer requires more time in order to compute a result. Queue time reach around 100 ms on average, slightly higher than Conversation Processor's queue time. Even though it is an acceptable time for a real-time scenario. Likewise, the processing time is around 900 ms at 1450 clients. The total delay which is the combination of queue and processing time is around 1 second.

Figure 16 shows the performance of the Conversation Processor (Aggregator) during an SlowDoS attack. Firstly, every network packet, corresponding with genuine clients, is processed. Then, when the attack begins, it causes a denial of service almost instantly. As the connections are occupied by the attacker, legitimate clients cannot request any service to the server. The packet-rate decrease quickly to very low values. This is due to the nature of the SlowDos attack that use a slow-rate strategy. When the attack is detected and mitigated, genuine clients can be served normally, so initial packet-rate is restored.

Depending on the T_{ts} used, the input rate of machine learning module changes. The figure 17 represents the input stream of conversation at layer (3). Since the sending timeout is 5 seconds, every 5 seconds a peak of conversation is received and classified. The conversations analyzed between peaks include finished conversations, therefore there is a important difference in the input rate amongst peaks at T_{ts} interval and in between.

In this environment we have conducted variety of experiments to measure our framework performance. The results obtained point that our proposal is able to timely detect SlowDoS attacks in a reasonable time, less than 9.8

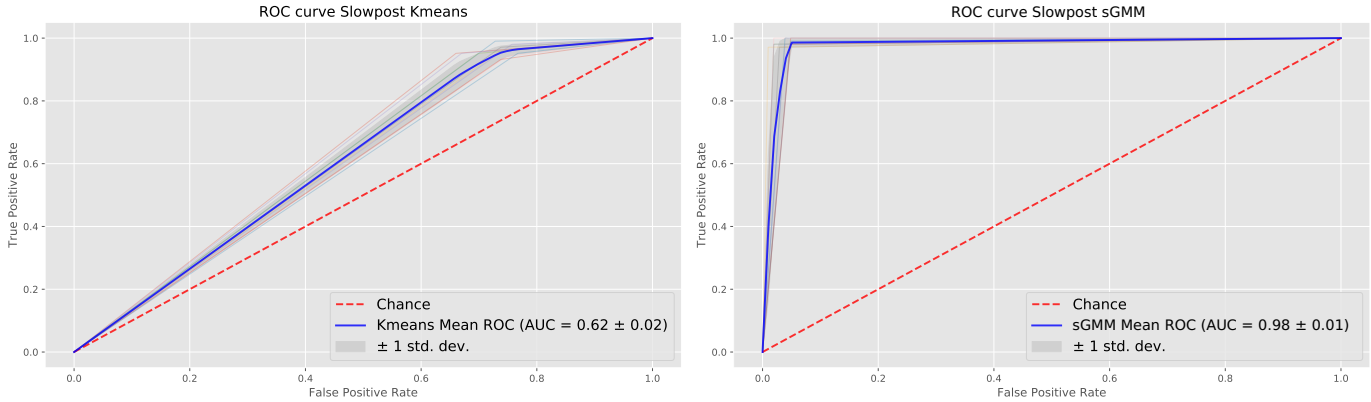

Figure 12: ROC curve for model K-Means(left) and strengthened GMM (right). Slowpost attack

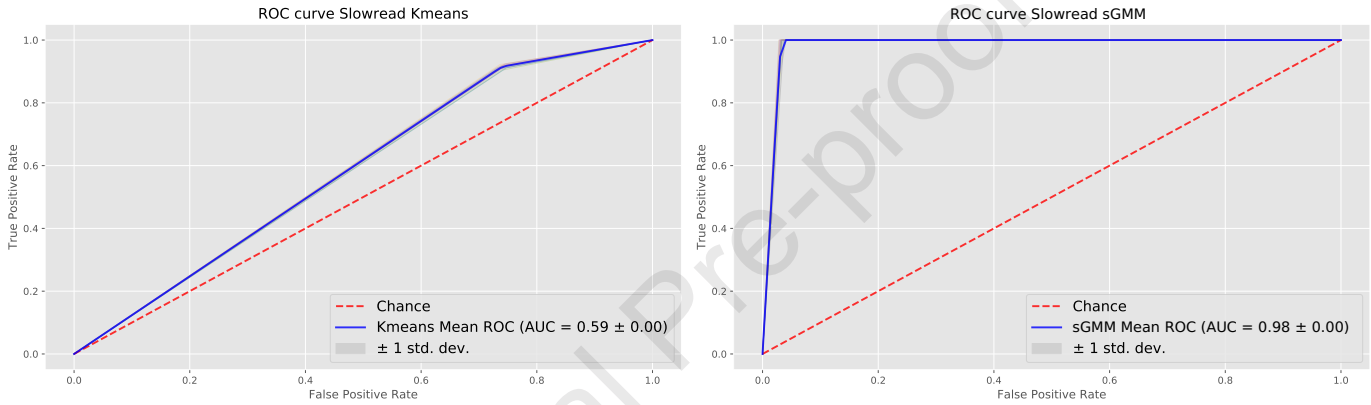

Figure 13: ROC curve for model K-Means(left) and strengthened GMM (right). SlowRead attack

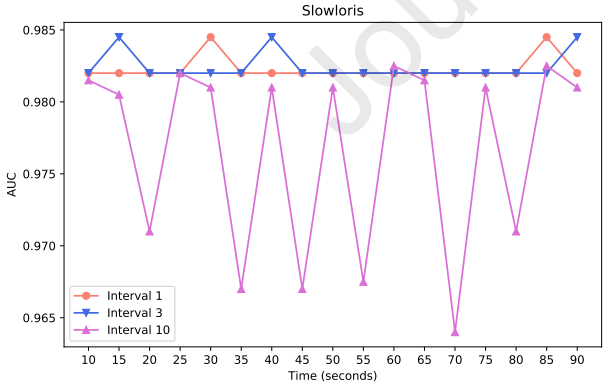

Figure 14: AUC of Slowloris with different intervals between follow up data. Detection results are shown every 5 seconds.

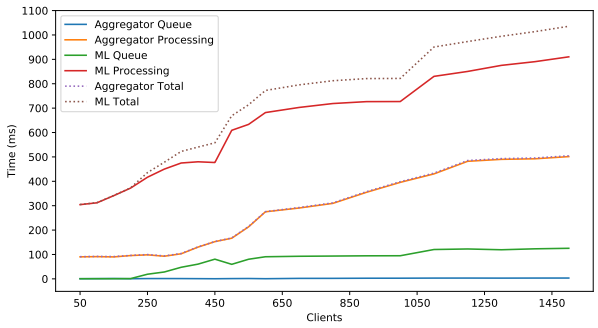

Figure 15: Time required by the aggregator and ML layer to ingest and process a batch versus the number of clients accessing the website

seconds in average. As illustrated in figure 15, the time required for layer (2) and (3) peaks to 1400 ms in total at 1450 parallel clients. Namely, the results obtained by the analyzer shows that the accuracy reach more than 98%, indicating that our proposal achieves optimal results. Additionally, the system successfully mitigates and recovers from an cyber-attack, even though when the attacker manages to accomplish the denial of service.

7. Conclusions and future work

This paper has described a distributed AI-based anomaly detection system aimed to detect SlowDoS attacks at application-level over encrypted traffic. The layered framework deeps inspect the network traffic in real-time, aggregating network flows and conversations, ex-

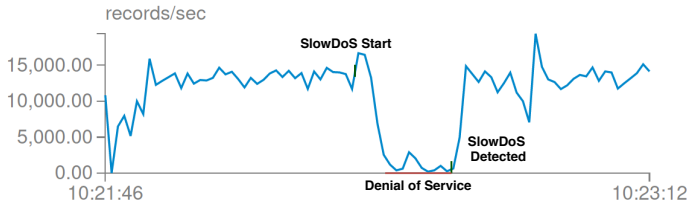

Figure 16: Spark Streaming logs at the aggregator before and after launching a SlowDoS attack with genuine traffic. A record represents a network packet.

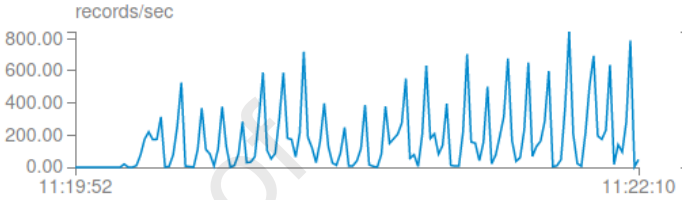

Figure 17: Spark Streaming logs at the machine learning module with $T_{ts} = 5$. A record represents a conversation.

Metric	High K-Means	High sGMM
Accuracy	0.84	0.98
Precision	0.84	0.96
Recall	0.85	1.0
F1-score	0.85	0.98
Specificity	0.84	0.96
Metric	Medium K-Means	Medium sGMM
Accuracy	0.85	0.98
Precision	0.85	0.96
Recall	0.86	1.0
F1-score	0.85	0.98
Specificity	0.85	0.96
Metric	Low K-Means	Low sGMM
Accuracy	0.88	0.98
Precision	0.85	0.96
Recall	0.91	1.0
F1-score	0.88	0.98
Specificity	0.84	0.96
Metric	SlowPost K-Means	SlowPost sGMM
Accuracy	0.61	0.97
Precision	0.56	0.96
Recall	0.95	0.98
F1-score	0.71	0.97
Specificity	0.27	0.96
Metric	SlowRead K-Means	SlowRead sGMM
Accuracy	0.84	0.98
Precision	0.84	0.96
Recall	0.84	1.0
F1-score	0.84	0.98
Specificity	0.84	0.96
Metric	Slowloris K-Means	Slowloris sGMM
Accuracy	0.87	0.98
Precision	0.85	0.96
Recall	0.89	1.0
F1-score	0.87	0.98
Specificity	0.84	0.96

Table 5: Quality metrics of the models for the different datasets.

tracting valuable features used by the distributed AI-model to detect dynamically attacks.

The solution has been implemented, and extensively validated in a real testbed running different kind of SlowDoS attacks. The paper has shown the feasibility, performance and accuracy (0.98) to successfully and timely detect SlowDoS cyber-attacks, outperforming the state of the art.

As future work, we envisage to extend and adapt our monitoring system, network flow processors, and AI-based model to dynamically detect other kind of cyberattacks, and considering other kind of network traffic, including 5G traffic.

Acknowledgements

This work has been sponsored by the European Commission through H2020 CyberSec4Europe project (contract 830929), H2020 INSPIRE-5Gplus project (contract 871808) and H2020 IoTCrawler project (contract 779852). It has been also partially funded by AXA Postdoctoral Scholarship awarded by the AXA Research Fund, as well as funded by MINECO and the EDRF funds of project PERSEIDES (ref. TIN2017-86885-R), ERDF funds of project UMU-CAMPUS LIVING LAB EQC2019-006176-P. It was also co-financed by the European Social Fund (ESF) and the Youth European Initiative (YEI) under the Spanish Seneca Foundation (CARM).

[1] J. Yuan, K. Mills, Monitoring the macroscopic effect of ddos flooding attacks, IEEE Tran. Dependable and Secure Computing 2 (2005) 324–335.

[2] S. Zeebaree, K. Hussein, R. Muhamad, Application layer distributed denial of service attacks defense techniques : A review, Academic Journal of Nawroz University 7 (2018) 113–116. doi:10.25007/ajnu.v7n4a279.

[3] E. Cambiaso, G. Papaleo, G. Chiola, M. Aiello, Slow dos attacks: definition and categorisation, International Journal of

Trust Management in Computing and Communications 1 (3-4) (2013) 300–319.

[4] M. Aiello, E. Cambiaso, M. Mongelli, G. Papaleo, An on-line intrusion detection approach to identify low-rate DoS attacks, Proceedings - International Carnahan Conference on Security Technology 2014-October (October) (2014). doi:10.1109/CCST.2014.6987039.

[5] D. Y. Perwej, M. Omer, O. Sheta, H. Harb, M. Adrees, The future of internet of things (iot) and its empowering technology Volume 9 (2019) Pages 20192 – 20203.

[6] N. Khan, J. Abdullah, A. S. Khan, Defending malicious script attacks using machine learning classifiers, Wireless Communications and Mobile Computing 2017 (2017).

[7] S. Thakare, P. Kaur, Denial-of-service attack detection system, 2017, pp. 281–285. doi:10.1109/ICISIM.2017.8122186.

[8] N. Tripathi, N. Hubballi, Y. Singh, How Secure are Web Servers? An empirical study of Slow HTTP DoS attacks and detection, Proceedings - 2016 11th International Conference on Availability, Reliability and Security, ARES 2016 (2016) 454–463doi:10.1109/ARES.2016.20.

[9] B. Zhou, J. Li, J. Wu, S. Guo, Y. Gu, Z. Li, Machine-learning-based online distributed denial-of-service attack detection using spark streaming, IEEE International Conference on Communications 2018-May (2018). doi:10.1109/ICC.2018.8422327.

[10] W. G. Hatcher, W. Yu, A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends, IEEE Access 6 (2018) 24411–24432. doi:10.1109/ACCESS.2018.2830661.

[11] J. Gardiner, S. Nagaraja, On the security of machine learning in malware C&C detection: A survey, ACM Computing Surveys 49 (3) (2016) 1–38. doi:10.1145/3003816.

[12] M. Kedziora, P. Gawin, M. Szczepanik, I. Jozwiak, Malware detection using machine learning algorithms and reverse engineering of android java code, International Journal of Network Security & Its Applications (IJNSA) Vol 11 (2019).

[13] D. Gibert, C. Mateu, J. Planes, The rise of machine learning for detection and classification of malware: Research developments, trends and challenges, Journal of Network and Computer Applications 153 (2020) 102526. doi:https://doi.org/10.1016/j.jnca.2019.102526.

[14] R. Chalapathy, S. Chawla, Deep Learning for Anomaly Detection: A Survey (2019) 1–50arXiv:1901.03407.
URL http://arxiv.org/abs/1901.03407

[15] B. A. Khalaf, S. A. Mostafa, A. Mustapha, M. A. Mohammed, W. M. Abduallah, Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods, IEEE Access 7 (2019) 51691–51713. doi:10.1109/ACCESS.2019.2908998.

[16] W. Wei, H. Song, H. Wang, X. Fan, Research and simulation of queue management algorithms in ad hoc networks under ddos attack, IEEE Access 5 (2017) 27810–27817.

[17] A. A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for Internet of Things, Future Generation Computer Systems 82 (2018) 761–768. doi:10.1016/j.future.2017.08.043.
URL http://dx.doi.org/10.1016/j.future.2017.08.043

[18] S. Alzahrani, L. Hong, Detection of distributed denial of service (ddos) attacks using artificial intelligence on cloud, Proceedings - 2018 IEEE World Congress on Services, SERVICES 2018 (2018) 37–38doi:10.1109/SERVICES.2018.00031.

[19] A. Abubakar, B. Pranggono, Machine learning based intrusion detection system for software defined networks, Proceedings - 2017 7th International Conference on Emerging Security Technologies, EST 2017 (2017) 138–143arXiv:1708.04571, doi:10.1109/EST.2017.8090413.

[20] J. Kim, J. Kim, H. L. T. Thu, H. Kim, Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection, 2016 International Conference on Platform Technology and Service, PlatCon 2016 - Proceedings (2016). doi:10.1109/PlatCon.2016.7456805.

[21] T. G. Nguyen, T. V. Phan, B. T. Nguyen, C. So-In, Z. A. Baig, S. Sanguanpong, SeArch: A Collaborative and Intelligent NIDS Architecture for SDN-Based Cloud IoT Networks, IEEE Access 7 (2019) 107678–107694. doi:10.1109/ACCESS.2019.2932438.

[22] J. Li, Z. Zhao, R. Li, H. Zhang, Ai-based two-stage intrusion detection for software defined iot networks, IEEE Internet of Things Journal 6 (2) (2019) 2093–2102. doi:10.1109/JIOT.2018.2883344.

[23] M. Goldstein, S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PloS one 11 (4) (2016) e0152173.

[24] A. González-Vidal, J. Cuenca-Jara, A. F. Skarmeta, Iot for water management: Towards intelligent anomaly detection, in: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), IEEE, 2019, pp. 858–863.

[25] F. Falcão, T. Zoppi, C. B. V. Silva, A. Santos, B. Fonseca, A. Ceccarelli, A. Bondavalli, Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 318–327.

[26] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, J. Srivastava, A comparative study of anomaly detection schemes in network intrusion detection, in: Proceedings of the 2003 SIAM international conference on data mining, SIAM, 2003, pp. 25–36.

[27] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, S. Stolfo, A geometric framework for unsupervised anomaly detection, in: Applications of data mining in computer security, Springer, 2002, pp. 77–101.

[28] J. Dromard, G. Roudiere, P. Owezarski, Online and scalable unsupervised network anomaly detection method, IEEE Transactions on Network and Service Management 14 (1) (2016) 34–47.

[29] T. Zoppi, A. Ceccarelli, L. Salani, A. Bondavalli, On the educated selection of unsupervised algorithms via attacks and anomaly classes, Journal of Information Security and Applications 52 (2020) 102474.

[30] M. Alauthman, N. Aslam, M. Al-kasassbeh, S. Khan, A. Al-Qerem, K.-K. R. Choo], An efficient reinforcement learning-based botnet detection approach, Journal of Network and Computer Applications 150 (2020) 102479. doi:https://doi.org/10.1016/j.jnca.2019.102479.

[31] B. Cusack, R. Lutui, R. Khaleghparast, Detecting slow DDos attacks on mobile devices, Proceedings of the 27th Australasian Conference on Information Systems, ACIS 2016 (2016) 1–12.

[32] M. Mongelli, M. Aiello, E. Cambiaso, G. Papaleo, Detection of DoS attacks through Fourier transform and mutual information, IEEE International Conference on Communications 2015-September (2015) 7204–7209. doi:10.1109/ICC.2015.7249476.

[33] A. Mehmood, M. Mukherjee, S. H. Ahmed, H. Song, K. M. Malik, Nbc-maids: Naïve bayesian classification technique in multi-agent system-enriched ids for securing iot against ddos attacks, The Journal of Supercomputing 74 (10) (2018) 5156–5170.

[34] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, J. Siltanen, Increasing web service availability by detecting application-layer ddos attacks in encrypted traffic, in: 2016 23rd International Conference on Telecommunications (ICT), 2016, pp. 1–6. doi:10.1109/ICT.2016.7500408.

[35] L. Shilpa, J. Sini, V. Bhupendra, Feature reduction using principal component analysis for effective anomaly–based intrusion detection on nsl-kdd, International Journal of Engineering Science and Technology 2 (07 2010).

[36] E. Cambiaso, G. Papaleo, M. Aiello, Taxonomy of slow dos attacks to web applications, Vol. 335, 2012, pp. 197–202. doi:10.1007/978-3-642-34135-9_20.

[37] G. Maciá-Fernández, J. Díaz-Verdejo, P. García-Teodoro, F. Toro-Negro, Lordas: A low-rate dos attack against application servers, Vol. 5141, 2007, pp. 197–209. doi:10.1007/978-3-540-89173-4_17.

[38] ndpi – ntop, https://www.ntop.org/products/deep-packet-inspection/ndpi/, accessed: 2020-03-26.

[39] T. Bujlow, V. Carela-Español, P. Barlet-Ros, Independent comparison of popular dpi tools for traffic classification, Computer Networks 76 (2015) 75–89.

[40] Product brochure dpi engine benefits & key features, https://www.ipoque.com/news-media/resources/brochures/product-brochure-dpi-engine-benefits-amp-key-features, accessed: 2020-03-26.

[41] S. Alcock, R. Nelson, Libprotoident: traffic classification using lightweight packet inspection, WAND Network Research Group, Tech. Rep. (2012).

[42] Enea qosmos ixengine datasheet, https://www.qosmos.com/wp-content/uploads/Enea-Qosmos-ixEngine-Datasheet_20200220.pdf, accessed: 2020-03-26.

[43] B. Wehbi, E. Montes de Oca, M. Bourdelles, Events-based security monitoring using mmt tool, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 860–863. doi:10.1109/ICST.2012.188.

[44] A. Molina Zarca, J. Bernal Bernabe, I. Farris, Y. Khettab, T. Taleb, A. Skarmeta, Enhancing iot security through network softwarization and virtual security appliances, International Journal of Network Management 28 (5) (2018) e2038, e2038 nem.2038. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2038, doi:10.1002/nem.2038.

[45] G. J. McLachlan, K. E. Basford, Mixture models: Inference and applications to clustering, Vol. 38, M. Dekker New York, 1988.

[46] B. Macukow, Neural networks–state of art, brief history, basic models and architecture, in: IFIP international conference on computer information systems and industrial management, Springer, 2016, pp. 3–14.

[47] J. Heaton, Introduction to neural networks with Java, Heaton Research, Inc., 2008.

Norberto García Marín received the B.Sc degree in Computer Science from the University of Murcia. He is currently pursuing the M.Sc in New Technologies in Computer Science in the University of Murcia, specialized in networks and telematics. He has been working in projects such as CyberSec4Europe. His main interests include security, privacy and artificial intelligence.

Dra. Aurora Gonzalez Vidal graduated in Mathematics from the University of Murcia in 2014. In 2015 she got a fellowship to work in the Statistical Division of the Research Support Service, where she specialized in Statistics and Data Analysis. Afterward, she studied a Big Data Master. In 2019, she got a Ph.D. in Computer Science, focusing her research on Data Analysis for Energy Efficiency. Currently, she is a postdoctoral researcher at the University of Murcia. She has collaborated in several national and European projects such as ENTROPY, IoTCrawler, and DEMETER. Her research covers machine learning in IoT-based smart environments, missing values imputation, and time series segmentation. She is the president of the R Users Association UMUR

Dr. Jorge Bernal Bernabe received the MSc, Master and PhD in Computer Science from the University of Murcia. Currently, he is a postdoctoral researcher in the University of Murcia. He has published over 50 papers in international conferences and journals. He has been involved in the scientific committee of numerous conferences. During the last years, he has been working in several European research projects such as Inter-Trust, SocIoTal, ARIES, OLYMPUS, ANASTACIA, INSPIRE-5G, CyberSec4Europe. His scientific activity is mainly devoted to the security, trust and privacy management in distributed systems and IoT.

Tomás Alcañiz Cascales graduated in Physics from the University of Murcia in 2017 and Master's Degree in Big Data Analysis Technologies. Currently he is collaborating on the European H2020 research project IoTCrawler. His main research interests are related to data science, artificial intelligence and machine learning.

Dr. Diego Rivera received the bachelor of computer sciences and computing engineering degree from the University of Chile, Santiago, Chile, in 2011 and 2013, respectively, and the Ph.D. degree focused on the analysis of the quality of experience for OTT services from the Université Paris-Saclay, Paris, France, in 2016. From 2012 to 2013, he was a Research Assistant with NIC Chile Research Labs, University of Chile, where he was involved with research on concurrency issues in Linux UDP sockets. He is currently a Research and Project Engineer with Montimage, Paris, where he is involved in the development of security monitoring solutions for Internet of Things-based cyber-physical systems.

Dr Antonio Skarmeta received the M.S. degree in Computer Science from the University of Granada and B.S. (Hons.) and the Ph.D. degrees in Computer Science from the University of Murcia Spain. Since 2009 he is Full Professor at the same department and University. Antonio F. Skarmeta has worked on different research projects in the national and international area in the networking, security and IoT area, like Euro6IX, ENABLE, DAIDALOS, SWIFT, SEMIRAMIS, SMARTIE, SOCIOTAL, IoT6 ANASTACIA, CyberSec4Europe. His main interested is in the integration of security services, identity, IoT and Smart Cities. He has been heading of the research group ANTS since its creation on 1995. He has published over 200 international papers and being member of several program committees.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: