Detection of Slowloris Attacks using Machine Learning Algorithms

Vinícius de Miranda Rios
Instituto Federal de Educação,
Ciência e Tecnologia do Tocantins
Palmas, Tocantins, Brazil
vinicius.rios@ifto.edu.br

Pedro R. M. Inácio
Instituto de Telecomunicações
Universidade da Beira Interior
Covilhã, Portugal
inacio@di.ubi.pt

Damien Magoni
LaBRI - CNRS
Université de Bordeaux
Talence, France
damien.magoni@u-bordeaux.fr

Mário M. Freire
Instituto de Telecomunicações
Universidade da Beira Interior
Covilhã, Portugal
mario@di.ubi.pt

ABSTRACT

The Slowloris attack, a variant of the slow Denial-of-Service (DoS) attack, is a stealthy threat that aims to take down web services provided by companies and institutions. It is able to pass through the traditional defense systems, due to the low amount and high latency of its attack traffic, often mimicking legitimate user traffic. Therefore, it is necessary to investigate techniques that can detect and mitigate this type of attack and simultaneously prevent legitimate user traffic from being blocked. In this work, we investigate nine machine learning algorithms for detecting Slowloris attacks, as well as a new combination based on Fuzzy Logic (FL), Random Forest (RF), and Euclidean Distance (ED) that we call FRE. We first generate Slowloris attack traffic traces in various environments. We then assess these algorithms under two scenarios: hyperparameters with default values and optimized hyperparameters. We show that most of these machine learning algorithms perform very well, with the random forest leading to the best classification results with test accuracy values reaching 99.52%. We also show that our FRE method outperforms all these algorithms, with test accuracy values reaching 99.8%.

CCS CONCEPTS

• **Networks** → **Denial-of-service attacks**; **Web protocol security**; **Network monitoring**; • **Computing methodologies** → **Supervised learning**; **Supervised learning by classification**; **Feature selection**.

KEYWORDS

Denial of Service (DoS) attack, fuzzy logic, low-rate DoS attack, machine learning, Slowloris.

SAC '24, April 8–12, 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04.
https://doi.org/10.1145/3605098.3635919

ACM Reference Format:
Vinícius de Miranda Rios, Pedro R. M. Inácio, Damien Magoni, and Mário M. Freire. 2024. Detection of Slowloris Attacks using Machine Learning Algorithms. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC'24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605098.3635919

1 INTRODUCTION

The *Slowloris attack* [15, 32] consists in sending partial HTTP requests with an encoding `\r\n` tag at the end of the header for opening connections between a computer and a targeted web server and on maintaining those connections open in order to stop or severely slow down the target. Since the appearance of this kind of attack, several solutions have been proposed for mitigating its impact. These solutions employ strategies such as the limitation of the number of connections per user or the setup of timeouts for each connection. However, these kind of limitations can impede the connections of legitimate users which go above a pre-established limit, e.g., users who access pages with numerous objects in a non-persistent HTTP connection mode [15]. Consequently, this has motivated the development of several approaches for detecting these attacks [1, 3, 6, 10, 15, 17, 20, 24, 34], rather than equally limiting all traffic. The proposed solutions nonetheless involve a high degree of complexity and require time-consuming responses, limited or unreported classifier performance, or were designed for specific network types (e.g., wireless mesh networks).

This paper focuses on detecting Slowloris attacks by employing and analyzing two different approaches. The first approach involves the evaluation of nine machine learning (ML) algorithms for Slowloris attacks detection, specifically K-Nearest Neighbors (K-NN), Gaussian Naive Bayes (GNB), Multi-layer Perceptron (MLP) neural network, Support Vector Machine (SVM), Decision Tree (DT), Multinomial Naive Bayes (MNB), Random Forest (RF), Gradient Boosting (XGB) and Light Gradient Boosting Machine (LGBM). These nine algorithms have been widely investigated for detecting high-rate DDoS attacks, with convincing results (e.g., MLP and k-NN [18], SVM [23], MNB [28], GNB [18], DT [37], RF [19], XGB [2] and LGBM [22]). The second approach, is a novel method which combines three different algorithms: (i) Fuzzy Logic (FL), (ii) Random Forest (RF) and (iii) Euclidean Distance (ED). We have called

Table 1: Comparison of related works to this article.

Works	Testbed Environment	Network Type	Detection Mechanism(s)	Attack Software	Train/ Test Ratio	Dataset	Performance
Aiello et al. (2013) [1]	LAN	-	SBID	-	-	self-gathered	High anomaly detection, with low probability of false positive rate.
Dantas et al. (2014) [10]	Simulated	Ethernet	SeVen	slowloris.py	-	self-gathered	High availability, greater robustness and less traffic
Mongelli et al. (2015) [24]	LAN	Ethernet	Fast Fourier Transform	-	-	self-gathered	Efficient attack detection by exploiting the flow connection entropy
Katkar et al. (2015) [20]	LAN	-	IDS with Naive Bayesian classifier	slowloris.pl	80/20	self-gathered	Detection accuracy of 97.15%
Aqil et al. (2015) [3]	LAN	Ethernet	High and low traffic threshold	-	-	self-gathered	High TPR and FPR decreased by 66%
Hirakawa et al. (2016) [17]	LAN	Ethernet	Amount of connections per IP addr. and duration time	slowhttptest	-	self-gathered	Highly resistant and and effective detection
Singh and De (2017) [34]	-	-	MLP-GA	slowhttptest	-	public and self-gathered	Detection accuracy of 98.04%
Araújo et. al. (2017) [12]	LAN	Ethernet	Snort and Suricata	slowhttptest	-	self-gathered	Suricata detects less attacks than Snort
Muraleedharan and Janet in (2020) [25]	-	-	Deep learning algorithm	slowhttptest	60/40	public	Detection accuracy of 99.61%
Faria et al. in (2020) [15]	LAN	Wi-fi	SDToW	-	-	self-gathered	Low incidence of false positive errors
Deolindo et al. in (2021) [13]	LAN	Ethernet	QDA	-	60/40	public	Detection accuracy of 98.32%
Oktivasari et al. in (2022) [27]	LAN	Ethernet	iptables	slowloris.py	-	self-gathered	Effective Detection
Murthy et al. in (2023) [26]	LAN	Wi-fi	Thresholds limit and timeout connection by Monte Carlo model	-	-	public and self-gathered	Effective Detection
This article (2023)	Emulated LAN and MAN	Ethernet	ML algorithms, fuzzy logic and Euclidean distance	slowhttptest	75/25	self-gathered	see Section 5

this method FRE and it leverages the best characteristics of each algorithm. The contributions of this work are therefore threefold:

- We generate and provide four datasets of network data traffic containing distributed Slowloris attacks, collected in various real-life and emulated environments (LAN and MAN).
- We train, test and evaluate nine ML algorithms with both default and optimized hyperparameters for detecting Slowloris attacks in these datasets.
- We propose and evaluate a novel method, called FRE, built by combining three algorithms (FL, RF, and ED).

The first contribution had to be fulfilled due to the lack of datasets containing distributed Slowloris attacks among data traffic. The best known dataset containing Slowloris traffic is the CIC-IDS 2017 [7], which contains a Slowloris attack trace coming from only one source. In order to build our datasets, we have launched several simultaneous Slowloris attacks from various multiple sources.

The paper is structured as follows: Section 2 presents related prior works concerning this topic. Section 3 details the algorithms used in our work to detect Slowloris attacks. Section 4 describes the design of the dataset collection process, while Section 5 shows the results of our evaluation experiments.

2 RELATED WORK

Although machine learning methods have been extensively investigated as a means of detecting traditional high-rate DoS/DDoS attacks, few papers have focused on DoS attacks that purposely use low-rate traffic attacks such as Slowloris attacks. Table 1 compares the main features and characteristics of the related works briefly discussed below.

In [1], Aiello et al. investigated the detection of anomalous traffic patterns and proposed a Statistical Based Intrusion Detection (SBID) approach to detect Slowloris attacks. The study employs statistical approaches based on signatures for IP traffic classification, leveraging machine learning techniques for flow clustering and traffic classification. One of the main threats highlighted is the Slowloris attack. The detection methodology involves comparing unknown and potentially anomalous traffic to a representation of legitimate traffic. Their research showed that the use of an average-based method, although time-consuming, provides near-absolute

accuracy in detecting anomalous traffic. Dantas et al. [10] proposed a defense mechanism called SeVen (Selective Defense for Application Layer DDoS Attacks), which uses the notion of state in the messages of the HTTP protocol to detect Slowloris attacks. In [24], Mongelli et al. proposed a detection method that uses the Fast Fourier Transform in the current time horizon in the frequency domain to identify the attack. Katkar et al. [20] configured an IDS system with a Naive Bayesian classifier with different methods for data pre-processing for detecting DoS/DDoS attacks against HTTP servers. Aqil et al. [3] proposed a detection algorithm that checks various features against high and low thresholds. If certain features exceed their high thresholds while others remain below their low thresholds, an attack is flagged. In [17], Hirakawa et al. described a 3-step detection scheme. The first one checks if the number of connections that are monitored is greater than a threshold during a normal time. If the threshold is trespassed, then in step 2 all the IPs belonging to the same flow and appearing most frequently are disconnected. In step 3, if the connections are below the threshold, stop the disconnection process and the cycle starts over.

Singh and De [34] proposed a method entrenched on an MLP with a genetic algorithm (GA) to detect DDoS application layer attacks. Experimental results show that MLP-GA method achieved an accuracy of 98.04% having a rate of 2.21% of False Positive and the use of MLP-GA leads to a better performance than the traditional classifiers e.g., Naive Bayes, Radial Basis Function (RBF) Network, MLP, J48, and C45. However, it does not make the distinction between DDoS attacks at the application layer and flash events (which are not attacks). Furthermore, the time-consuming response is not analyzed in the paper. In [12], Araújo et al. made a comparison between the Snort and Suricata IDSs (Intrusion Detection Systems) in order to verify which one works better at detecting Slowloris attacks. Snort has been more effective in detecting such attacks, as well as in memory and CPU consumption. However, Suricata was quicker to detect the attack, which means that, although it was fast, it still failed to detect all attacks throughout the experiments.

Muraleedharan and Janet [25] used a deep learning approach to detect Slowloris attacks, which led to an experimental result of 99.61% of accuracy. In [15], Faria et al. developed a tool called SDToW (Slowloris Detecting Tool for WMNs) with the aim to detect and block Slowloris attacks in WMNs. Deolindo et al. [13] employed Machine Learning techniques and specifically focused on QDA (Quadratic Discriminant Analysis) as a classification method for IDS to distinguish between legitimate network traffic and malicious attacks. Their research aimed at enhancing the capability of IDS to identify and classify Port Scan and DoS Slowloris attacks more effectively. Asch et al. [4] developed an asynchronous classifier of network flows for detecting Slowloris attacks. This classifier was implemented using random forests and its effectiveness was measured by the area under the ROC curve (AUC). They found that using features individually was sufficient to obtain reliable detection results, with two individual features having an AUC greater than 0.95. However, this metric can't be easily compared to the other studies which typically use the accuracy metric.

Murthy et al. in [26] proposed a method to analyze the performance of the Apache2 server using both versions of HTTP, i.e., version 1.1 and 2, in the context of both traditional networks and Software Defined Networks. A threshold is set for parameters such as time to connect to the webpage and latency in obtaining a response from clients using the Monte Carlo model. If the server detects values exceeding these thresholds or if a timeout occurs, it identifies it as malicious activity. Once the attacker's source is pinpointed, the server blocks the client's route, severing the connection between them. Oktivasari et al. in [27] studied the application of iptables as a firewall tool to detect and mitigate the Slowloris attack. The authors emphasize the effectiveness of iptables in managing network traffic, including the ability to limit and block IP addresses directly or completely.

3 METHODS FOR DETECTING SLOWLORIS ATTACKS

3.1 Classification Features

We manually selected features that lead us to a significant performance during classification without impacting operation in real-time, similarly to other approaches, e.g., [8]. After preliminary experiments with four features, summarized in Figure 1, we selected two features to represent the traffic from the HTTP connections: entropy (which captures dispersion well) and the amount of distinct ports. We follow the definition of entropy given by Shannon applied to values for five network traffic features, including a representation of the transport protocol (e.g., TCP, UDP, ICMP), source and destination IP addresses, and the corresponding port numbers and we also take into consideration the amount of distinct source ports in a given time frame and by transport protocol in the traffic trace.

We use the following method for computing the two features. To a specific trace of traffic, a sliding window of time is applied with a ΔT length according to the period in the Timestamp field for traces marked by the `tshark` tool. For this paper, we consider $\Delta T = 1$ s by default. Initially, for a 1 s sliding window of time, we assess these two features for all traced traffic packets.

In the first time window, we evaluate the entropy of those packets and the amount of distinct ports found in them. Next, we shift the time window by 1 s, evaluating the amount of distinct ports for the packets and the entropy within this new window. This process is repeated by sliding the time window by $\Delta T = 1$ s up until the end of the trace is reached. The last time window can be a fraction of ΔT, as the duration of the trace may not be a multiple of ΔT. In the next subsections some classification approaches that use the values of entropy and amount of distinct ports obtained for each sliding time window to classify the network traffic are described. The set of values of these two features obtained from each sliding window of time is herein called a dataset.

3.2 Settings of the Machine Learning Algorithms

To evaluate the feasibility of the selected features and analyze the performance of the nine ML algorithms, we resorted to implementations readily available at *scikit-learn* [29] with their default settings, as summarized in Table 2, except for the parameters therein emphasized in bold, which were adjusted according to the result of the `RandomizedSearchCV` algorithm and which are presented here with their default values and in Table 3 with their optimized values.

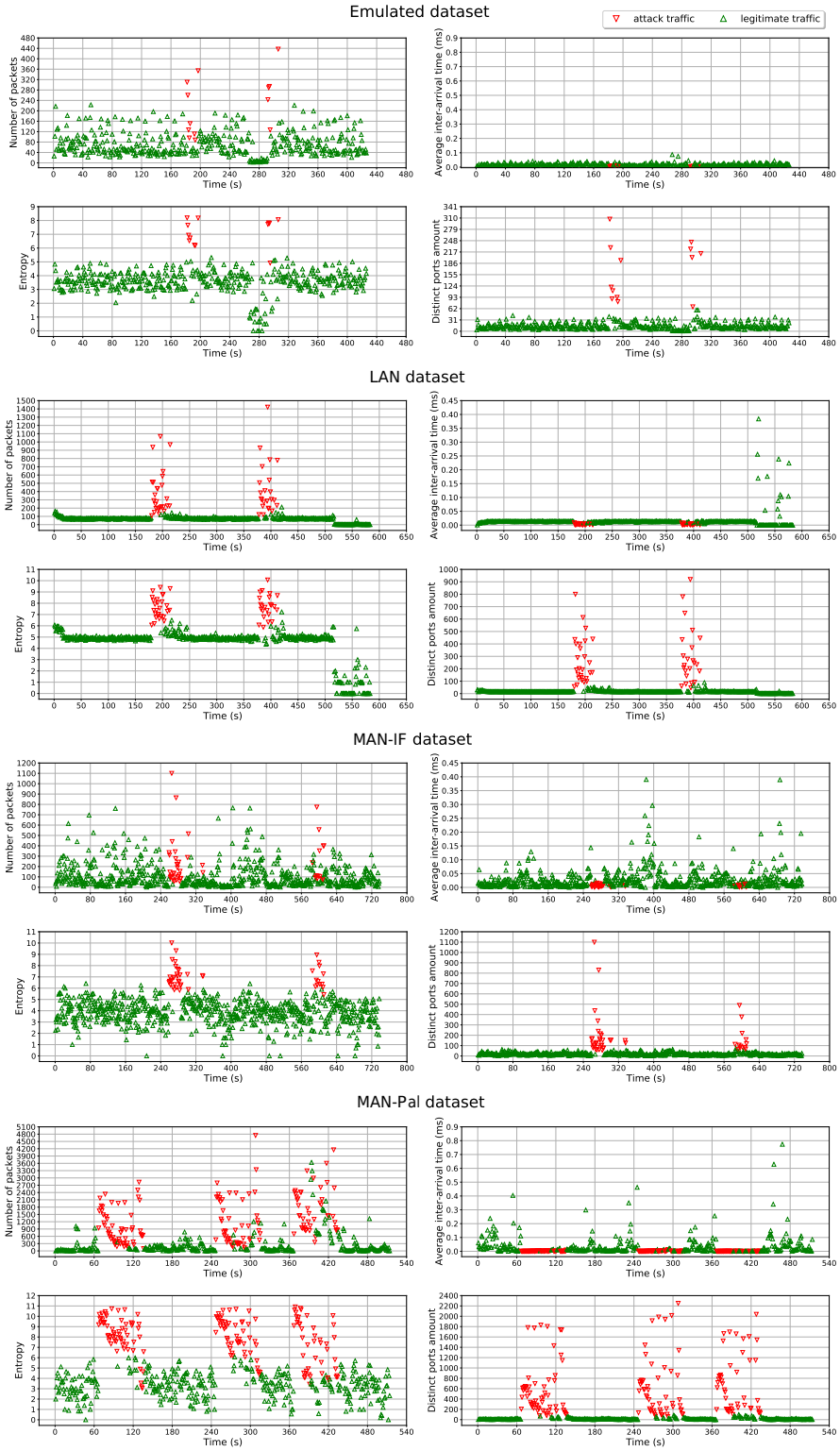

Figure 1: Number of packets (top left), Average inter-arrival time (top right), Entropy (bottom left), and Amount of distinct ports (bottom right) for the emulated, LAN, MAN-IF, and MAN-Pal datasets (with a time window of length ΔT = 1 s).

Table 2: Hyperparameters' default settings.

Algorithm	Settings
MLP	**activation**=relu, alpha=0.0001, batch_size=auto, beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, **hidden_layer_sizes**=100, **learning_rate**=constant, learning_rate_init=0.001, max_fun=15000, **max_iter**=200, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, **random_state**=None, shuffle=True, solver=adam, tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False
K-NN	algorithm=auto, leaf_size=30, metric=minkowski, metric_params=None, n_jobs=None, **n_neighbors**=5, p=2, weights=uniform
SVM	**C**=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=ovr, degree=3, **gamma**=scale, **kernel**=rbf, max_iter=-1, probability=False, **random_state**=None, shrinking=True, tol=0.001, verbose=False
MNB	**alpha**=1.0, class_prior=None, fit_prior=True
GNB	priors=None, **var_smoothing**=1e-09
DT	ccp_alpha=0.0, class_weight=None, **criterion**=gini, **max_depth**=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, random_state=0, splitter=best
RF	bootstrap=True, ccp_alpha=0.0, class_weight=None, **criterion**=gini, max_depth=None, max_features=auto, max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators**=100, n_jobs=None, oob_score=False, **random_state**=None, verbose=0, warm_start=False
LGBM	bootstrap=True, ccp_alpha=0.0, class_weight=None, criterion=gini, **max_depth**=None, max_features= auto, max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators**=100, n_jobs=None, oob_score=False, **random_state**=None, verbose=0, warm_start=False
XGB	ccp_alpha=0.0, criterion=friedman_mse, init=None, **learning_rate**=0.1, loss=deviance, **max_depth**=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators**=100, n_iter_no_change=None, **random_state**=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False

The parameters chosen to be adjusted were based on [5, 33, 35]. To adjust them, the randomized search was selected instead of the grid search because, although they explore exactly the same space of parameters and the result in parameter settings is quite similar, the run time for the randomized search is drastically lower [29], which makes it more adequate for this work, leading to the configurations summarized in Table 3.

Table 3: Hyperparameters' optimized settings.

Algorithm	Parameters
MLP	activation=["identity", "logistic", "tanh", "relu"], hidden_layer_sizes=[100, 200, 300, 400], learning_rate=["constant", "invscaling", "adaptive"], max_iter=[2000, 2500, 3000]
K-NN	n_neighbors=range(1, 50, 1)
SVM	C=[1.0, 10.0, 40.0, 50.0], kernel=["linear", "poly", "rbf", "sigmoid"], gamma=["scale", "auto"]
MNB	alpha=range(1, 1000, 1)
GNB	var_smoothing=np.logspace(0, -9, num=100)
DT	criterion=["gini", "entropy", "log_loss"], max_depth=[2, 10, 15, 50, 100, 200]
RF	criterion=["gini", "entropy", "log_loss"], n_estimators=[100, 200, 300, 400, 500]
LGBM	n_estimators=range(100, 1000, 100), max_depth=range(-1, 50, 1)
XGB	n_estimators=range(100, 1000, 50), learning_rate=np.arange(0.1, 10, 0.1), max_depth=range(3, 100, 3)

To generate the training and test sets we used the split ratio of 75:25 [14, 36] and the default value of 10 in k-fold for the cross-validation [30]. Among the selected ML algorithms, MLP, RF, SVM, LGBM, DT, and XGB algorithms had the `random_state` field changed to static to avoid the random generation of the values of the features in each iteration round, respectively.

3.3 Method Built on Fuzzy Logic, Random Forest and Euclidean Distance

This method derives from a mix of three methods: FL, RF, and ED. First, the traffic is classified in a given trace using the FL method described below. It is next classified (in the same trace) with the RF algorithm, described in the previous subsection, due to the satisfactory classification results of this algorithm (as shown in Section 5).

If both traffic classification results obtained with FL and RF are the same, then that is the classification final result. Otherwise, we perform a further step, that consists in applying ED to obtain, for each value of the amount of distinct ports feature and entropy, the minimum distance. In this latter case, this additional step completes what we call the FRE method. This leads to the two approaches yielding different classifications for every value of the amount of distinct ports and entropy in the dataset used for training. A classification is then obtained using ED for the amount of distinct ports and entropy because the values of the distinct ports amount and entropy in the dataset used for training match known traffic beforehand tagged as legitimate or attack. Finally, we compare the classification of the amount of distinct ports and entropy using the ED with the classification obtained through the fuzzy logic approach. Given that we have two classifications now, when the quantity classified as "attack type" is higher than the quantity classified as "legitimate type" classifications, which means that it is an attack and vice versa. When the classification achieved through ED for the amount of distinct ports and the classification achieved through fuzzy logic is not the same, in the two-classification set, it means one classification corresponds to an attack and the other to legitimate, which gives us a warning as the final result. Ensuing, warning alert packets are studied and contrasted to the attack traffic packets that have been blocked. If the two packet sets are alike related to the great values for the amount of distinct ports (distinct ports amount higher than 80), the suspected network traffic is going to be blocked. If not, the destination receives the network packets that pointed to warning alerts.

Regarding FL, we consider a typical fuzzy expert system, where three linguistic terms are used to configure the fuzzy, specifically `low`, `medium` and `high` (for each of the two features entropy and amount of distinct ports). Each linguistic term was chosen based on the amount of traffic a web server can receive. Low traffic concerns situations with low customer access; medium traffic concerns situations caused by some regional event in the institution, resulting in higher activity than usual; and high traffic concerns situations caused by some kind of DDoS attack or a flash crowd type of event. The ranges of values for each linguistic term were based on the average values generated for each attribute per dataset, as shown in Figure 2.

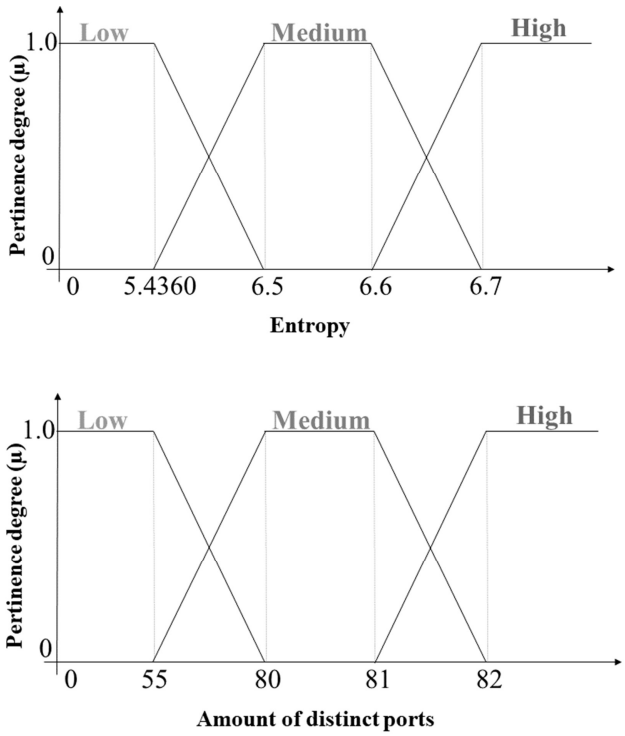

Figure 2: Pertinence function for the features entropy and amount of distinct ports.

The pertinence function that was chosen to normalize the values of the data is, for every feature, trapezoidal. The selected value range for the linguistic terms in the universe of discourse for every single variable is derived from the datasets, more specifically from the values of the two evaluated features from it. Consequently, the `low` term is used for the lowest values of the features, `high` for the highest values, and `medium` for the average ones. Following fuzzification, the fuzzy rules set receives the degree of pertinence values created by each input variable. It is made up of 9 rules, that must cover all possible scenarios for system fuzzy behaviour, as shown in Figure 3. The Mamdani inference model [21] is applied in all the rules.

RULE	IF (ANTECEDENT)						THEN (CONSEQUENT)
	AMOUNT OF DISTINCT PORTS			ENTROPY			CLASSIFICATION
	LOW	MEDIUM	HIGH	LOW	MEDIUM	HIGH	
1			■			■	ATTACK
2			■		■		ATTACK
3			■	■			ATTACK
4		■				■	ATTACK
5		■			■		ATTACK
6		■		■			ATTACK
7	■					■	ATTACK
8	■				■		LEGITIMATE
9	■			■			LEGITIMATE

Figure 3: Base rules for the inference module.

The `AND` operator is applied on the predecessors of every rule, and the lowest value is kept among the triggered rule pertinence degree values, as a consequence. When the inference module has triggered every single rule, the defuzzification starts. To translate the pertinence degree values into an output providing accurate numerical values, we use the Center-of-Area (CoA) method [16] to defuzzify them. The resulting output is then categorized according to the linguistic terms *"legitimate"* or *"attack"*.

4 EXPERIMENTAL ENVIRONMENTS AND DATASETS

Due to the lack of publicly available distributed Slowloris attack traffic traces, we created such traces in three distinct environments: emulated, local (LAN) area, and metropolitan (MAN) area environments. The emulated environment was created with four attackers, three normal clients, and one web server, using the *Netkit* [9] software in all emulated computers and network devices.

The local environment was created with five attackers and one web server at IFTO (Instituto Federal de Educação, Ciência e Tecnologia do Tocantins) in Brazil, which has a network infrastructure with routers distributed among different buildings, that allow data traffic coming from different locations. We created two metropolitan area environments with the same architecture, but with different web server locations: one with the web server located at IFTO (MAN-IF) and another one with the web server located at the Palmas municipal government (MAN-Pal). Each metropolitan environment was created with four attackers distributed around the city of Palmas (Brazil) using the available Internet infrastructure to attack the web server located at IFTO (MAN-IF) or located at Palmas municipal government (MAN-Pal). Normal clients in LAN and MAN environments are the users accessing those servers. The security assessments mimicking real attacks on web servers of IFTO and Palmas municipal government were carried out for research purposes only, with the required institutional authorizations and in compliance with the values of academic integrity and code of ethics. The attack procedure follows the flowchart shown in Figure 4 for the three environments. Each attack-mimicking process takes 20 minutes, with a continuous collection of normal client traffic. The attack is initiated 180 s after the beginning and lasts for 60 s. The traffic traces obtained have the following sizes: 40.6 MB (emulated), 1.0 GB (LAN), 165.51 MB (MAN-IF), and 23.6 MB (MAN-Pal). These four datasets are open data and are freely available for the research community on the CERN's Zenodo platform [11].

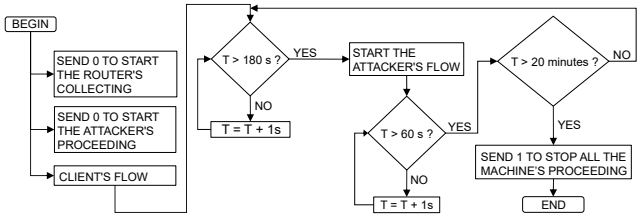

Figure 4: Flowchart of the attack mimicking procedure.

We merged the above four traffic traces (emulated, LAN, MAN-IF, and MAN-Pal) into just one to facilitate the application of the 10-fold cross-validation, since this larger dataset is randomly partitioned in a ratio of 75%/25%, in each of the 10 folds (75% is used for training and 25% is used for testing).

Table 4: Classification results (sorted by Test accuracy %) for the default hyperparameters and for the optimized hyperparameters (given as a delta in square brackets when ≠ 0). N: Normalized data.

Algorithm	Traffic	Precision in %	Recall in %	F1-Score in %	Confusion matrix *attack*	Confusion matrix *legitimate*	Test accuracy in %	Train accuracy in %	Standard deviation
MNB	*attack*	93.47[-1.10]	89.25[+5.52]	91.30[+2.25]	TP=59.80[+3.70]	FN=4.20[+1.10]	*97.98[+0.46]*	*97.79[+0.46]*	*0.011508[-0.001998]*
	legitimate	98.56[+0.73]	99.16[-0.22]	98.86[+0.26]	FP=7.20[-3.70]	TN=493.80[-1.10]			
SVM(N)	*attack*	99.01[-3.29]	89.70[+5.97]	94.10[+1.56]	60.10[+4.00]	0.60[+2.30]	*98.67[+0.30]*	*98.53[+0.76]*	*0.007353[-0.001289]*
	legitimate	98.63[+0.79]	99.88[-0.46]	99.25[+0.17]	6.90[-4.00]	497.40[-2.30]			
GNB	*attack*	93.31[+1.21]	99.70[-1.34]	96.36	66.80[-0.90]	4.90[-1.00]	*99.10[+0.02]*	*99.07*	*0.007501*
	legitimate	99.96[-0.18]	99.02[+0.20]	99.48[+0.01]	0.20[+0.90]	493.10[+1.00]			
MLP(N)	*attack*	98.61[+0.17]	94.93[+0.90]	96.72[+0.54]	63.60[+0.60]	0.90[-0.10]	*99.24[+0.12]*	*99.10[+0.19]*	*0.006776[-0.000852]*
	legitimate	99.32[+0.12]	99.82[+0.02]	99.57[+0.07]	3.40[-0.60]	497.10[+0.10]			
DT	*attack*	96.79[+0.43]	97.31[+0.30]	97.04[+0.36]	65.20[+0.20]	2.20[-0.30]	*99.29[+0.09]*	*99.10[+0.11]*	*0.006886[-0.000591]*
	legitimate	99.64[+0.04]	99.56[+0.06]	99.60[+0.05]	1.80[-0.20]	495.80[+0.30]			
K-NN(N)	*attack*	97.35[+0.17]	97.16[-0.15]	97.24[-0.01]	65.10[-0.10]	1.80[-0.10]	*99.35*	*99.23[-0.17]*	*0.006700[+0.000813]*
	legitimate	99.62[-0.02]	99.64[+0.02]	99.63	1.90[+0.10]	496.20[+0.10]			
XGB	*attack*	97.39[+0.42]	98.36[-0.30]	97.85[+0.07]	65.90[-0.20]	1.80[-0.30]	*99.49[+0.02]*	*99.33[-0.23]*	*0.006253[+0.000652]*
	legitimate	99.78[-0.04]	99.64[+0.06]	99.71[+0.01]	1.10[+0.20]	496.20[+0.30]			
LGBM	*attack*	97.66[-0.29]	98.36[+0.15]	98.00[-0.07]	65.90[+0.10]	1.60[+0.20]	*99.52[-0.02]*	*99.36*	*0.006494*
	legitimate	99.78[+0.02]	99.68[-0.04]	99.73[-0.01]	1.10[-0.10]	496.40[-0.20]			
RF	*attack*	97.25[+0.28]	98.81[-0.15]	98.01[+0.07]	66.20[-0.10]	1.90[-0.20]	*99.52[+0.02]*	*99.37[-0.01]*	*0.006124[-0.000166]*
	legitimate	99.84[-0.02]	99.62[+0.04]	99.73[+0.01]	0.80[+0.10]	496.10[+0.20]			
FRE	*attack*	98.68	99.70	99.18	67.40[-0.10]	0.20	*99.80*	*99.37[-0.01]*	*0.006124[-0.000166]*
	legitimate	99.96	99.82	99.89	0.90	494.90[+0.30]			

5 PERFORMANCE EVALUATION

This section offers an assessment of the approaches used for classifying the traces of traffic listed above. We employ the metrics *Precision*, *Recall*, *F1-Score* and *Accuracy* (hereby referred to as *Test accuracy* and *Train accuracy*) as typically described in the literature (e.g., [31]). For some ML algorithms, we also normalized the dataset so that values range from 0 and 1, in accordance with the respective input definition, though we do not specifically mention this afterwards in the paper.

As explained at the end of Section 4, to apply the 10-fold cross-validation with a split ratio of 75:25 for training and testing, we merged the four datasets into just one. The values shown in Table 4 are thus the average of the classification results for both the nine ML algorithms and the FRE method in the different portions of the dataset used for training and for testing in the 10 iterations.

The *Train accuracy* and *Standard deviation* metrics were generated through the *cross_val_score* function with a k-fold of size 10. The value of the *Train accuracy* metrics is an average of the averages produced from 10 runs generated by the *cross_val_score* function for each one of the 10 portions of the dataset randomly generated from the single dataset mentioned above. The same occurs with the *Standard deviation* metric, which is generated based on the *Train accuracy* metric as the result of the *cross_val_score* function, but its value is the average of the 10 portions of the dataset.

The confusion matrix related to the traffic type (i.e., attack or legitimate), has been computed for each algorithm and for the FRE method. It contains the average absolute values of true positives (detected attacks) on the upper left corner, true negatives (regular traffic) on the lower right, false positives (false attacks) on the lower left corner and false negatives (undetected attacks) on the upper right corner.

5.1 Classification Results

Table 4 summarizes the results of the classification approaches using both default and optimized values or the hyperparameters. The values in the table in plain display were obtained for the default values of the hyperparameters; in square brackets we present the differences of the results to the ones outputted for the optimized hyperparameters.

The table highlights that, for the nine ML algorithms with default parameters, *Precision* ranges from 93.47% to 99.01% for the attack traffic. The legitimate traffic is also properly classified with values ranging from 98.56% to 99.96%, with two algorithms above 99.80%. *Test accuracy* ranges from 97.98% to 99.52% and *Train accuracy* ranges from 97.79% to 99.37%, which shows that these ML algorithms are efficient when applied to Slowloris attack detection.

Results in Table 4 prove that most of the ML algorithms are generalizing well. The *Standard deviation* shows that the training accuracy of the algorithms has a low dispersion on all the iterations, except for the MNB algorithm with default hyperparameters, which has a slightly worse deviation value than the others. In both default and optimized cases, the FRE method and the RF algorithm have the same values for the *Train accuracy* and *Standard deviation* metrics. This is due to the inclusion of the RF algorithm in the FRE method.

The FRE method with default parameters reaches a *Test accuracy* of 99.80% and a *Train accuracy* of 99.37%. These values are higher than the ones obtained for the ML algorithms as well as the ones

found in the literature (when available) as show in Table 1. The FRE method scores at 98.68% for *Precision*, 99.70% for *Recall*, and 99.18% for *F1-Score* on the attack traffic. These high scores are also reflected in the confusion matrix of the FRE method where the number of false negatives and false positives are extremely low.

The traffic marked as *"warning"* is analyzed and, if it has the same characteristics as the attack traffic, it will be blocked; otherwise, it will be allowed to proceed to the destination. As can be seen in Table 5, the features *amount of distinct ports* and *entropy* have similar values between legitimate and attack traffic. This can happen because a sliding window with a length of 1 s can contain a lot of legitimate traffic from different sources causing a high value for this feature, sometimes leading to a high rate of false positives. Therefore, in the dataset, the traffic with no attack traffic but with these values was classified as legitimate, and vice versa.

Table 5: Warning alerts.[a]

Dataset round	# of distinct ports	Entropy	ED port classif.	ED entropy classif.	FL classif.	RF classif.
1	63	5.2937	attack	legitimate	attack	legitimate
	54	6.3059	legitimate	attack	legitimate	attack
	53	6.5016	legitimate	attack	legitimate	attack
2	67	5.1669	attack	legitimate	attack	legitimate
	59	3.4315	attack	legitimate	attack	legitimate
	[78]	[5.8722]	attack	legitimate	attack	legitimate
3	(52)	(6.5016)	legitimate	attack	legitimate	attack
	65	5.9644	legitimate	attack	attack	legitimate
4	71	3.8757	attack	legitimate	attack	legitimate
5	56	5.8129	legitimate	attack	attack	legitimate
	53	6.5016	legitimate	attack	legitimate	attack
6	76	4.9252	attack	legitimate	attack	legitimate
	(53)	(6.5016)	legitimate	attack	legitimate	attack
	54	6.3059	legitimate	attack	legitimate	attack
	63	5.2937	attack	legitimate	attack	legitimate
7	(65)	(5.9644)	legitimate	attack	attack	legitimate
	78	3.7911	attack	legitimate	attack	legitimate
	76	2.8301	attack	legitimate	attack	legitimate
	78	2.8301	attack	legitimate	attack	legitimate
8	63	5.2937	attack	legitimate	attack	legitimate
	59	3.4357	attack	legitimate	attack	legitimate
	53	6.4621	legitimate	attack	legitimate	attack
9	56	4.8959	attack	legitimate	legitimate	attack
	57	4.7329	attack	legitimate	attack	legitimate
10	78	5.8722	attack	legitimate	attack	legitimate
	71	3.8741	attack	legitimate	attack	legitimate
	66	4.9212	attack	legitimate	attack	legitimate
	68	5.1669	attack	legitimate	attack	legitimate

[a] in parentheses if only found with default hyperparameters, in square brackets if only found with optimized hyperparameters.

Contextualizing our results with the related work, and although the datasets are different (and therefore the results are not directly comparable), the FRE method led, as far as we know, to the best classification results for Slowloris attacks achieved so far.

5.2 Usage of Resources

A computer with a quad-core 64-bit processor, 8 GB of RAM, and running Ubuntu 20.04 was used to perform the experiments reported herein and obtain values related to the computational resources consumed by the FRE method as well as the ML algorithms. Table 6 displays the resources used by each classifier during the classification process that led to the results presented in Table 4. The depicted values are an average of 10 execution rounds for each algorithm and for the FRE method. CPU usage, RAM usage, and execution times were measured with the GNU `time` command. We can see that CPU usage is above 100.00%, meaning that more than 1 core was required for classifying the entire dataset. The algorithms are sorted from the one consuming the least CPU to the one using the most CPU, when using default hyperparameters.

Table 6: Computational resource usage (sorted by CPU%).

Algorithm	Default parameters	Optimized parameters
FRE	CPU: 137.10%	CPU: 148.10%
	RAM: 112.85MB	RAM: 100.56MB
	Execution time: 0.69 s	Execution time: 0.80 s
LGBM	CPU: 137.90%	CPU: 148.10%
	RAM: 115.78MB	RAM: 130.55MB
	Execution time: 0.63 s	Execution time: 1.07 s
XGB	CPU: 138.40%	CPU: 143.50%
	RAM: 88.98MB	RAM: 98.96MB
	Execution time: 0.54 s	Execution time: 0.73 s
RF	CPU: 139.30%	CPU: 145.10%
	RAM: 89.16MB	RAM: 99.36MB
	Execution time: 0.54 s	Execution time: 0.79 s
SVM	CPU: 153.70%	CPU: 172.20%
	RAM: 82.88MB	RAM: 92.80MB
	Execution time: 0.41 s	Execution time: 0.46 s
K-NN	CPU: 154.10%	CPU: 170.20%
	RAM: 84.62MB	RAM: 94.20MB
	Execution time: 0.41 s	Execution time: 0.49 s
DT	CPU: 156.50%	CPU: 159.30%
	RAM: 85.97MB	RAM: 95.62MB
	Execution time: 0.40 s	Execution time: 0.49 s
GNB	CPU: 159.40%	CPU: 175.60%
	RAM: 77.73MB	RAM: 86.52MB
	Execution time: 0.37 s	Execution time: 0.43 s
MNB	CPU: 160.60%	CPU: 179.60%
	RAM: 78.23MB	RAM: 86.95MB
	Execution time: 0.38 s	Execution time: 0.42 s
MLP	CPU: 246.30%	CPU: 282.30%
	RAM: 79.72MB	RAM: 90.01MB
	Execution time: 1.73 s	Execution time: 3.01 s

As can be seen in Table 6, the FRE method consumes less CPU than all ML algorithms for hyperparameters with default values. It is the same for hyperparameters with optimized values, except for the RF and XGB algorithms which have a few % lower usage. It also consumes less RAM than the LGBM algorithm for hyperparameters with default and optimized values. However, compared to the other algorithms, it consumes around one-third more RAM with default hyperparameters and one-tenth with optimized hyperparameters. The superior performance of the FRE method has a cost in terms of execution time, only being less costly than the MLP algorithm and the LGBM algorithm with optimized values. In order to perform the classification, FRE requires 0.69 s when using default hyperparameters and 0.8 s when using optimized hyperparameters. Meanwhile, the other ML algorithms require from 0.37 s to 1.07 s for both types of hyperparameters, except for the MLP algorithm, which is the slowest.

The FRE method shows the best classification results compared to the ML algorithms but it also requires more RAM and more execution time. However, those resources remain in the same order

of magnitude than the ones used by the ML algorithms. Therefore, the FRE method is perfectly suitable for online operation. Indeed, looking at both tables 4 and 6, we can observe that the algorithms can be roughly placed in three categories: (i) the first category has a fast execution time but a low accuracy and includes the GNB, MNB, and SVM algorithms; (ii) the second category has both an average execution time and an average accuracy and includes the K-NN, DT, and XGB algorithms; and (iii), the third category has a slow execution time and a high accuracy and includes the LGBM, RF and FRE algorithms. MLP is off-category with a high CPU usage and a very high execution time despite a reasonable accuracy. Therefore, a trade-off can be made depending on the requirements of the detection system regarding real-time processing and traffic volume.

6 CONCLUSION

We assessed and compared nine ML algorithms as well as a combined method with three algorithms (Fuzzy Logic, Random Forest, and Euclidean Distance) for detecting distributed Slowloris attacks. Those algorithms and the FRE method were evaluated using a union of emulated, LAN, and MAN traffic traces. For both the hyperparameters with default values and the hyperparameters with optimized values, the best classification results were obtained by our FRE method with a *Test accuracy* of 99.8%, although the gap was thin with some other ML algorithms such as the Random Forest and LGBM. Indeed, among the nine ML algorithms evaluated, five of them (DT, K-NN, XGB, LGBM, and RF) had all their scores above 95% which makes them appropriate candidates for the task. However, only our FRE method had all its scores above 98.5%.

In most of the cases, using the optimized hyperparameters only made a small improvement to the results obtained by using the default hyperparameters. This can ease the work of practitioners who will not be required to optimize the hyperparameters before use if they can cope with a small loss of accuracy. We showed that the algorithms exhibit a trade-off between speed and accuracy, and could be selected depending on the requirements of the detection system. Regarding the resource usage, FRE is the least CPU-intensive algorithm but is on the upper side regarding the RAM consumption and the execution time. The latter still being totally compatible with an online operation mode for a detection system.

For future work, we intend to compare our proposal with unsupervised machine learning algorithms, measuring the classification accuracy by the attributes chosen by the algorithms for the detection of Slowloris attacks. In addition, we also plan to generate new datasets with a huge volume of data traffic produced by new environments with the addition of many endpoint and intermediate devices, expanding the paths on which the attack traffic travels to reach the target. In this way, new values of the attributes will be generated, which could be used for the detection of such attacks.

7 ACKNOWLEDGMENTS

This work is funded by Portuguese FCT/MCTES through national funds and, when applicable, co-funded by EU funds under the project UIDB/EEA/50008/2020 and by FCT/COMPETE/FEDER under the project SECURIoTESIGN with ref. number POCI-01-0145-FEDER-030657, and funded by operation Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, co-funded by the European Regional Development Fund (ERDF) through the Programa Operacional Regional do Centro (Centro 2020), and by the Brazilian CAPES Foundation through the grant contract BEX 9095/13-6 of the doctoral student Vinícius de Miranda Rios.

REFERENCES

[1] Maurizio Aiello, Enrico Cambiaso, Silvia Scaglione, and Gianluca Papaleo. 2013. A similarity based approach for application DoS attacks detection. In *IEEE International Symposium on Computers and Communications (ISCC)*. 430–435.
[2] Hassan A. Alamri and Vijey Thayananthan. 2020. Bandwidth control mechanism and extreme gradient boosting algorithm for protecting software-defined networks against DDoS attacks. *IEEE Access* 8 (2020), 194269–194288.
[3] A. Aqil, A. Atya, T. Jaeger, S. Krishnamurthy, K. Levitt, P. McDaniel, J. Rowe, and A. Swami. 2015. Detection of stealthy TCP-based DoS attacks. In *IEEE Military Communications Conference*. 348–353.
[4] Christian Asch, Gabriel Gálvez, Eric Ríos, Juan José Vargas, Luis Quesada, Gabriela Barrantes, and Adrián Lara. 2021. Asynchronous Detection of Slowloris Attacks Via Random Forests. In *IEEE Jornadas Costarricenses de Investigación en Computación e Informática*.
[5] Raj Kumar Batchu and Hari Seetha. 2021. A generalized machine learning model for DDoS attacks detection using hybrid feature selection and hyperparameter tuning. *Computer Networks* 200 (2021), 108498.
[6] Chad L. Calvert and Taghi M. Khoshgoftaar. 2019. Impact of class distribution on the detection of slow HTTP DoS attacks using Big Data. *Journal of Big Data* 6, 1 (2019), 67.
[7] Canadian Institute for Cybersecurity. 2017. Intrusion Detection Evaluation Dataset (CIC-IDS2017). https://www.unb.ca/cic/datasets/ids-2017.html
[8] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. 2018. Power spectrum entropy based detection and mitigation of low-rate DoS attacks. *Computer Networks* 136 (2018), 80–94.
[9] Computer Networks Laboratory, Roma Tre University. 2020. Netkit emulator. https://www.netkit.org/
[10] Yuri Gil Dantas, Vivek Nigam, and Iguatemi E. Fonseca. 2014. A selective defense for application layer ddos attacks. In *IEEE Joint Intelligence and Security Informatics Conference*. 75–82.
[11] Vinícius de Miranda Rios, Pedro R. M. Inácio, Damien Magoni, and Mário Marques Freire. 2023. Detection of Slowloris Attacks using Machine Learning Algorithms. https://doi.org/10.5281/zenodo.8316038
[12] Tiago Emílio de Sousa Araújo, Fernando Menezes Matos, and Josilene Aires Moreira. 2017. Intrusion detection systems' performance for distributed denial-of-service attack. In *Conference on Electrical, Electronics Engineering, Information and Communication Technologies*. 1–6.
[13] Vinícius M. Deolindo, Bruno L. Dalmazo, Marcus V. B. da Silva, Luiz R. B. de Oliveira, Allan de B. Silva, Lisandro Zambenedetti Granville, Luciano P. Gaspary, and Jéferson Campos Nobre. 2021. Using Quadratic Discriminant Analysis by Intrusion Detection Systems for Port Scan and Slowloris Attack Classification. In *International Conference on Computational Science and Its Applications*. 188–200.
[14] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In *IEEE Security and Privacy Workshops*. 7–12.
[15] Vinicius da Silva Faria, Jéssica Alcântara Gonçalves, Camilla Alves Mariano da Silva, Gabriele de Brito Vieira, and Dalbert Matos Mascarenhas. 2020. SDToW: A Slowloris Detecting Tool for WMNs. *Information journal* 11, 12 (2020), 544.
[16] A. Haripriya and K. Kulothungan. 2019. Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. *EURASIP Journal on Wireless Communications and Networking* 1 (2019), 1–15.
[17] Tetsuya Hirakawa, Kanayo Ogura, Bhed Bahadur Bista, and Toyoo Takata. 2016. A defense method against distributed slow HTTP DoS attack. In *19th International Conference on Network-Based Information Systems*. 152–158.
[18] S. Hosseini and M. Azizi. 2019. The hybrid technique for DDoS detection with supervised learning algorithms. *Computer Networks* 158 (2019), 35–45.
[19] M. Idhammad, K. Afdel, and M. Belouch. 2018. Detection system of HTTP DDoS attacks in a cloud environment based on information theoretic entropy and random forest. *Security and Communication Networks* (2018).
[20] V. Katkar, A. Zinjade, S. Dalvi, T. Bafna, and R. Mahajan. 2015. Detection of DoS/DDoS attack against HTTP servers using naive Bayesian. In *IEEE International Conf. on Computing Communication Control and Automation*. 280–285.
[21] E. H. Mamdani and S. Assilian. 1999. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of human-computer studies* 51, 2 (1999), 135–147.

[22] Murk Marvi, Asad Arfeen, and Riaz Uddin. 2021. A generalized machine learning-based model for the detection of DDoS attacks. *International Journal of Network Management* 31, 6 (2021), e2152.
[23] Nisharani Meti, D. G. Narayan, and V. P. Baligar. 2017. Detection of distributed denial of service attacks using machine learning algorithms in software defined networks. In *International Conference on Advances in Computing, Communications and Informatics*. 1366–1371.
[24] Maurizio Mongelli, Maurizio Aiello, Enrico Cambiaso, and Gianluca Papaleo. 2015. Detection of DoS attacks through Fourier transform and mutual information. In *IEEE International Conference on Communications*. 7204–7209.
[25] N. Muraleedharan and B. Janet. 2021. A deep learning based HTTP slow DoS classification approach using flow data. *ICT Express* 7, 2 (2021), 210–214.
[26] Anusha A. Murthy, Prathima Mabel John, and Rama Mohan Babu Kasturi Nagappasetty. 2023. Hypertext transfer protocol performance analysis in traditional and software defined networks during Slowloris attack. *International Journal of Electrical & Computer Engineering* 13, 4 (2023).
[27] Prihatin Oktivasari, Ayu Rosyida Zain, Maria Agustin, Asep Kurniawan, Fachroni arbi Murad, and Muhammad fabian Anshor. 2022. Analysis of Effectiveness of Iptables on Web Server from Slowloris Attack. In *International Conference of Computer and Informatics Engineering*. 215–219.
[28] Mrutyunjaya Panda, Ajith Abraham, and Manas Ranjan Patra. 2010. Discriminative multinomial naive bayes for network intrusion detection. In *International Conference on Information Assurance and Security*. 5–10.
[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[30] S. V. Jansi Rani, Iacovos Ioannou, Prabagarane Nagaradjane, Christophoros Christophorou, Vasos Vassiliou, Sai Charan, Sai Prakash, Niel Parekh, and Andreas Pitsillides. 2023. Detection of DDoS attacks in D2D communications using machine learning approach. *Computer Communications* 198 (2023), 32–51.
[31] Vinícius de Miranda Rios, Pedro R. M. Inácio, Damien Magoni, and Mário Marques Freire. 2021. Detection of Reduction-of-Quality DDoS Attacks Using Fuzzy Logic and Machine Learning Algorithms. *Computer Networks* 186 (2021), 107792.
[32] Vinícius de Miranda Rios, Pedro R. M. Inácio, Damien Magoni, and Mário Marques Freire. 2022. Detection and Mitigation of Low-Rate Denial-of-Service Attacks: A Survey. *IEEE Access* 10 (2022), 76648–76668.
[33] Odnan Sanchez, Matteo Repetto, Alessandro Carrega, and Raffaele Bolla. 2021. Evaluating ML-based DDoS detection with grid search hyperparameter optimization. In *IEEE International Conference on Network Softwarization*. 402–408.
[34] Khundrakpam Johnson Singh and Tanmay De. 2017. MLP-GA based algorithm to detect application layer DDoS attack. *Journal of Information Security and Applications* 36 (2017), 145–153.
[35] Raniyah Wazirali. 2020. An improved intrusion detection system based on KNN Hyperparameter tuning and cross-validation. *Arabian Journal for Science and Engineering* 45, 12 (2020), 10859–10873.
[36] Xin Wei, Lulu Zhang, Hao-Qing Yang, Limin Zhang, and Yang-Ping Yao. 2021. Machine learning for pore-water pressure time-series prediction: Application of recurrent neural networks. *Geoscience Frontiers* 12, 1 (2021), 453–467.
[37] Y. Wu, H. Tseng, Wuu Yang, and R. Jan. 2011. DDoS detection and traceback with decision tree and grey relational analysis. *International Journal of Ad-Hoc and Ubiquitous Computing* 7, 2 (2011), 121–136.