Scientific Journal of Informatics
Vol. 13, No. 3, August 2026
p-ISSN 2407-7658 http://journal.unnes.ac.id/nju/index.php/sji e-ISSN 2460-0040

Real-Time Multi-Class DoS Attack Detection on Proxmox VMs Using LightGBM with MikroTik Integration

Danu Candra Saputra[1*], Bambang Agus Herlambang[2], Noora Qotrun Nada[3]
[1,2,3]Department of Informatics, Universitas PGRI Semarang, Indonesia

Abstract.
Purpose: The adoption of virtualization increases the dependence on service availability, making Denial of Service (DoS) attacks a serious threat, while rule-based detection is poorly adaptive to evolving attacks. Many previous studies also rely on outdated public datasets, are evaluated offline, rarely measure inference latency, and lack automatic mitigation. This study aims to build a real-time multi-class DoS detection system on Proxmox virtual machines using LightGBM integrated with MikroTik.
Methods: A controlled testbed based on Proxmox and MikroTik was built to generate normal and attack traffic. The dataset was collected from the real infrastructure at a one-second granularity and labeled into six classes, namely the normal condition and five DoS attacks. LightGBM was proposed as the detection model, while XGBoost, Random Forest, Decision Tree, and SVM served as baselines, compared using a temporal holdout to prevent data leakage, with SMOTE applied only to the training data.
Findings: LightGBM was selected as the best model with an accuracy of 96.22%, a macro F1-score of 96.25%, and an inference latency of 1.369 ms. The four flooding attacks were detected almost perfectly, whereas Slowloris was the hardest class because it resembles normal traffic. Its PR-AUC dropped to 0.9271, and the system performed automatic mitigation at a median latency of 56.2 ms.
Originality: This study integrates lightweight real-time detection with automatic firewall-based mitigation in a closed loop on real infrastructure, emphasizing the balance between accuracy and efficiency rather than the highest accuracy alone. Future work can extend it to distributed (DDoS) attacks.

Keywords: DoS attack, LightGBM, Machine learning, MikroTik, Real-time detection
Received June 2026 / **Revised** July 2026 / **Accepted** July 2026

INTRODUCTION

The adoption of cloud computing is increasingly considered because it supports cost reduction, flexibility, efficiency, virtualization, and service availability [1]. Virtualization platforms allow several virtual machines to run on a single physical server [2], while network intrusion detection and prevention systems translate detection results into traffic filtering rules [3]. MikroTik serves as the gateway enforcing these rules in this study. The strong dependence on service availability makes network reliability critically important, because even minor disruptions can cause operational and financial losses [4]. One of the most common threats to this availability is the Denial of Service (DoS) attack, which floods the target resources until the service becomes inaccessible to legitimate users [5], [6]. DoS and DDoS attacks have many variants, ranging from volumetric attacks such as TCP SYN flood, UDP flood, and ICMP flood [7], application-layer attacks such as HTTP flood [8], to low-and-slow attacks such as Slowloris, which is difficult to detect because it resembles normal traffic [9]. Traditional signature-based detection approaches are less effective against new threats or previously unknown attack patterns, so a more adaptive approach is required [10].

Therefore, the application of machine learning to Intrusion Detection Systems (IDS) is increasingly used because it can learn network traffic patterns and classify attacks automatically [11]. Various algorithms have been applied, including Decision Tree, Random Forest, and Support Vector Machine (SVM) [12], as well as Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LightGBM) [13]. Ensemble and boosting-based algorithms have been reported to achieve high accuracy, for example

*Corresponding author.
Email addresses: danucandra100@gmail.com (Saputra), bambangherlambang@upgris.ac.id (Herlambang), noora@upgris.ac.id (Nada)
DOI: 10.15294/sji.v13i3.57873

Random Forest reaching 97.20% on CIC-IDS 2017 [14], XGBoost 99.67% on UNSW-NB15 [15], and a LightGBM-based model reaching 99.97% on the TON-IoT dataset [16]. Nevertheless, the performance of each algorithm depends on the data characteristics and the deployment scenario, so comparison among algorithms remains relevant [17].

However, previous studies still have several limitations. Many studies rely on old public datasets such as KDD Cup 99, NSL-KDD, or CICIDS that poorly reflect current real network conditions [18], [19], and most models are evaluated only offline without genuine real-time detection testing [20]. In addition, many systems only record and classify attacks without connecting to network devices for automatic mitigation [21], while the inference latency that determines real-time feasibility is rarely measured and reported [22]. Most studies also rely on a single algorithm and are therefore prone to bias, whereas studies that compare several algorithms under the same data conditions remain limited [23], [24]. These limitations point to three unresolved challenges, namely the reliance on outdated public datasets that do not reflect real networks, the absence of genuine real-time testing and latency measurement, and the lack of a direct link between detection and automatic mitigation. This study addresses these gaps by using data from real infrastructure, measuring inference latency under real-time deployment, and closing the loop from detection to mitigation, as compared against representative studies in Table 1.

Table 1. Comparison of representative studies against this study

Study	Dataset (Public/Real)	Real-time testing	Latency measured	Automated mitigation
Chavan and Alone [14]	Public (CIC-IDS 2017)	✗	✗	✗
Yulianton et al. [15]	Public (UNSW-NB15)	✗	✗	✗
Zhao et al. [16]	Public (TON-IoT)	✓	✓	✗
This study	Real (self-collected)	✓	✓	✓

Based on these problems, this study proposes a real-time multi-class DoS attack detection system on Proxmox virtual machines using LightGBM with MikroTik integration. The dataset is built directly from a MikroTik device at one-second granularity and labeled into six classes, namely the normal condition and five attack types (TCP SYN flood, UDP flood, ICMP flood, HTTP flood, and Slowloris). LightGBM is proposed as the detection model, while XGBoost, Random Forest, Decision Tree, and SVM serve as baselines compared fairly under a temporal holdout strategy to prevent data leakage, and the selected model is then deployed in a system that operates in real time and is connected to web-based monitoring and MikroTik for detection, attack source identification, and automatic blocking. Accordingly, the main contributions of this study are (1) the construction of a multi-class DoS attack dataset from real network infrastructure in real time, (2) a fair comparison in which LightGBM is evaluated as the proposed model against four baseline algorithms, with an evaluation that accounts for temporal order and measures inference latency, and (3) the integrated deployment of attack detection together with mitigation in a MikroTik and Proxmox environment.

METHODS

Research Design

This study was designed as a controlled experiment on a virtualized network testbed based on Proxmox and MikroTik. The experimental approach was chosen because it allows both normal traffic and DoS attacks to be generated under controlled conditions, so that the label of each traffic condition can be determined precisely based on the testing scenario. This design was also intended to produce data that better match the characteristics of current operational networks, given that previous RT-IDS studies still rely heavily on public datasets that have limitations and contain outdated traffic patterns [25].

As shown in Figure 1, the study proceeds through seven main stages, namely (1) testbed design, (2) generation of normal traffic and five types of DoS attacks, (3) dataset collection and automatic labeling from the traffic passing through the gateway, (4) data preprocessing and temporal splitting, (5) model development and comparison of five machine learning algorithms with hyperparameter configuration, (6) evaluation of detection performance, and (7) real-time deployment of the best model together with automatic mitigation. This staged design is consistent with real-time IDS studies that use self-collected datasets, network traffic feature extraction, data preprocessing, and the comparison of several machine learning algorithms to evaluate detection performance [26], and its step-by-step presentation adapts the staged methodological structure used in [27].

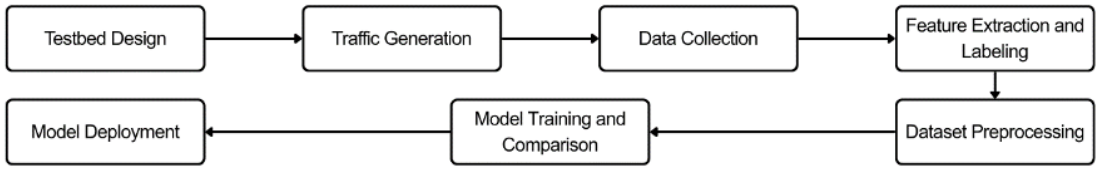

Figure 1. Overall research flow

System Architecture and Topology

The testing network topology is shown in Figure 2. The environment was arranged around a managed switch with VLAN segmentation (IEEE 802.1Q), and a MikroTik x86 served as both the gateway and the traffic observation point. Each function was separated into its own VLAN so that attack traffic was isolated and the testing conditions remained controlled [28]. The virtual machines ran on a Proxmox VE host (Dell PowerEdge R630), with the target VM and the collector VM on one segment and the attacker machine on a separate segment, so that all test traffic passed through the MikroTik and could be observed and labeled centrally.

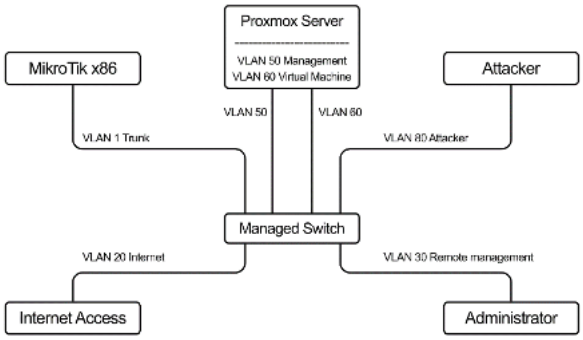

Figure 2. Testing network topology

For real-time detection, the MikroTik exports telemetry from three sources, namely Traffic Flow (NetFlow v5), connection tracking, and interface statistics, which the collector aggregates into a feature vector per one-second window that is then classified by LightGBM. Based on the detection result, the system identifies the attack source, and the automatic mitigation module writes a blocking rule on the MikroTik and then removes it automatically after a certain duration, forming a closed loop of detection and mitigation without manual intervention.

Attack Scenarios

Data collection was carried out under two conditions, namely normal traffic and attack traffic. Normal traffic represents reasonable network activity, such as web browsing and service access, as a representation of non-attack behavior. The attack traffic consists of five types of DoS attacks selected because they represent different layers [29], namely TCP SYN flood and UDP flood at the transport layer, ICMP flood at the network layer, and the high-rate HTTP flood and the low-and-slow Slowloris at the application layer [9]. So that the dataset captures the various manifestations of each class and is not confined to a single pattern, each attack was executed with several parameter variations such as sending rate, packet size, number of connections, duration, and target port [30]. A summary of the scenarios together with their tools and dataset labels is presented in Table 2.

Table 2. Attack scenarios

Class	Attack Type	Tool	Label
Normal traffic	Reasonable user activity	-	normal
TCP SYN flood	Transport-layer flooding (SYN)	hping3	TCP_SYN
UDP flood	Transport-layer flooding (UDP)	hping3	UDP_FLOOD
ICMP flood	Network-layer flooding (ICMP)	hping3	ICMP_FLOOD
HTTP flood	Application-layer flooding	ApacheBench	HTTP_FLOOD
Slowloris	Low-and-slow application layer	slowloris	SLOWLORIS

Dataset Collection and Labeling

The data collection mechanism is shown in Figure 3 and runs in four stages. First, the attacker and normal hosts generate traffic toward the target virtual machine, and all traffic passes through the MikroTik gateway. Second, the gateway continuously exports three complementary telemetry sources, namely Traffic Flow (NetFlow v5) records that describe per-flow statistics, connection tracking entries that describe active sessions, and interface counters that describe bandwidth activity. Third, a custom collector aggregates these three sources into a single feature vector for every one-second window, so that the traffic dynamics are captured temporally, and assigns a label based on the attack scenario that is active at that moment, so that the ground truth is obtained directly rather than through manual annotation. Fourth, each labeled vector is appended to a CSV file, forming the final dataset of six classes, namely one normal class and five DoS attack classes (TCP_SYN, UDP_FLOOD, ICMP_FLOOD, HTTP_FLOOD, and SLOWLORIS). To keep the normal class clean, the transition from an attack back to normal passes through a short cooling phase in which writing is paused until residual connection tracking entries from the attack have drained, so that leftover attack flows do not contaminate the normal samples.

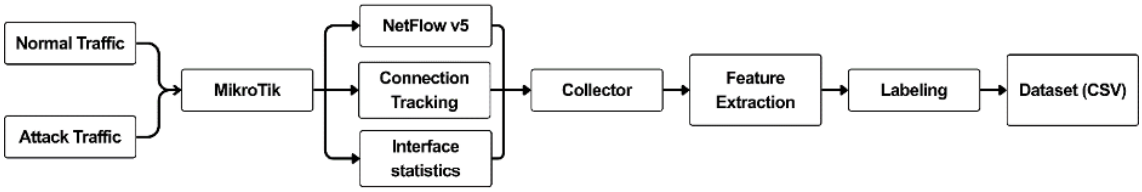

Figure 3. Dataset collection mechanism

In each window, 50 numerical features were extracted. Rather than using raw counters directly, selective domain-based feature engineering was applied, in which several derived features were constructed from network domain knowledge to strengthen the attack signal, while no automatic feature transformation or dimensionality reduction was performed so that all traffic information remained available to the models. These 50 features are grouped into eight categories based on their telemetry source, as summarized in Table 3.

Table 3. Summary of the 50 extracted features by category

Category (count)	Features
Bandwidth and interface (10)	rx_bytes_per_sec, tx_bytes_per_sec, rx_mbps, tx_mbps, rx_packets_per_sec, tx_packets_per_sec, rx_drops, tx_drops, rx_errors, tx_errors
Connection table (5)	total_connections, tcp_connections, udp_connections, icmp_connections, other_connections
TCP state (3)	tcp_syn_count, tcp_established_count, tcp_other_state_count
Source and destination diversity (5)	unique_src_ips, unique_dst_ips, top_src_ip_count, top_src_ip_pct, top_dst_ip_count
Slowloris-specific (2)	avg_orig_bytes_per_tcp_conn, low_data_tcp_count
Derived ratios (5)	conn_ratio, bps_per_conn, pps_per_conn, rx_tx_ratio, conn_per_unique_ip
Flow statistics NetFlow (18)	flows_count, total_bytes_nf, total_pkts_nf, avg_bytes_per_flow, avg_pkts_per_flow, unique_src_nf, unique_dst_nf, top_flow_proto_num, top_flow_dst_port, top_flow_bytes, top_flow_pkts, udp_flows, tcp_flows, icmp_flows, udp_bytes, tcp_bytes, max_bytes_single_flow, max_pkts_single_flow
Connection freshness (2)	conn_fresh, conn_age_s

Several derived features were formed to strengthen the attack signal. The conn_ratio feature measures the concentration of connections on a single source, namely the ratio of the number of connections from the dominant source IP c_top to the total active connections c_total as in Equation (1), so that a value close to one indicates the dominance of a single source, which is a characteristic of flooding attacks.

$$\text{conn\_ratio} = \frac{c_{top}}{c_{total}} \tag{1}$$

Meanwhile, rx_tx_ratio captures traffic asymmetry, namely the ratio of received throughput B_rx to transmitted throughput B_tx in Mbps as in Equation (2). A large value indicates that incoming traffic far exceeds outgoing traffic, a common pattern in flooding attacks. The denominator of both equations is bounded so that it does not become zero.

$$\text{rx\_tx\_ratio} = \frac{B_{rx}}{B_{tx}} \tag{2}$$

Preprocessing and Data Splitting
Before training, the raw dataset was prepared through several sequential steps. First, the non-feature columns, namely the timestamp and the one-second interval marker, were removed, while the 50 numerical features were retained as input and the attack class as the target. Second, infinite and empty values that may appear from ratio computations were replaced with zero so that every feature remained numeric. Third, the six class labels were encoded into integers using label encoding. No feature selection or dimensionality reduction was applied at this stage, so that all traffic information remained available to the models and the comparison among algorithms stayed fair.

Because the data are time series, the splitting was performed temporally per session (temporal holdout), namely the earlier portion as training data and the later portion as test data, rather than randomly, in order to prevent data leakage between temporally adjacent rows [31]. The boosting algorithms (LightGBM and XGBoost) used 65% training data, 10% validation data for early stopping, and 25% test data, whereas Random Forest, Decision Tree, and SVM used 75% training data and 25% test data. The final 25% test portion was identical for all models so that the comparison was fair.

The class distribution in the training data was imbalanced because normal traffic dominated. SMOTE was applied only to the training data, not to the validation or test data, so that the evaluation still reflected the original distribution and avoided leakage [32]. This technique generates synthetic samples through linear interpolation between a minority sample x_i and one of its k nearest neighbors x_zi with δ a random number in the range [0,1], as in Equation (3).

$$x_{new} = x_i + \delta \cdot (x_{zi} - x_i) \tag{3}$$

In this study, k = 5 was used, and every minority class was oversampled to match the majority class within the training data only. For the boosting algorithms with the 65% split, this increased the training data from 83,921 to 211,314 rows, while for the other three models with the 75% split, the training data increased from 96,850 to 243,882 rows. The validation and test portions were left at their original class distribution, so that the reported performance reflects the actual imbalance of the data.

Modeling and Evaluation
Five machine learning algorithms were compared, namely LightGBM, XGBoost [13], Random Forest, Decision Tree, and Support Vector Machine (SVM) [12]. LightGBM and XGBoost are efficient tree-based gradient boosting methods, Random Forest is a bagging ensemble of decision trees, Decision Tree serves as a single-tree baseline, and SVM separates classes using an RBF kernel. In this study, LightGBM is proposed as the primary detection model, while XGBoost, Random Forest, Decision Tree, and SVM serve as baseline methods for a fair comparison. All algorithms were trained on the same 50 features. Their hyperparameters were set manually to commonly used and reasonable values rather than through an exhaustive search such as grid or random search, since the four baseline models act as references rather than being individually optimized. The two boosting models used early stopping with a patience of 40 rounds on the validation set to determine the optimal number of trees, the SVM used standardized features through a StandardScaler pipeline, and a fixed random seed of 42 was used for all models to ensure reproducibility. The final hyperparameter configuration of all models is summarized in Table 4.

Although both LightGBM and XGBoost are gradient boosting methods, they differ in how the trees are grown, which explains their difference in efficiency. XGBoost grows trees level-wise by expanding all nodes at the same depth, whereas LightGBM grows trees leaf-wise by expanding the leaf with the largest loss reduction, producing more accurate trees with fewer nodes. LightGBM also applies Gradient-based One-Side Sampling (GOSS), which retains instances with large gradients and randomly samples those with small gradients, and it bins continuous features into discrete histograms [16]. These mechanisms reduce both memory usage and computation time [13], which makes LightGBM well suited for real-time detection where low inference latency is required [16].

Table 4. Final hyperparameter configuration of the five models

Model	Main hyperparameters
LightGBM	n_estimators=600, num_leaves=31, max_depth=5, learning_rate=0.08, subsample=0.8, colsample_bytree=0.8, min_child_samples=20, reg_alpha=0.1, reg_lambda=1.5, early stopping=40
XGBoost	n_estimators=600, max_depth=5, learning_rate=0.08, subsample=0.8, colsample_bytree=0.8, min_child_weight=3, gamma=0.1, reg_alpha=0.1, reg_lambda=1.5, early stopping=40
Random Forest	n_estimators=300, max_depth=16, min_samples_split=5, min_samples_leaf=2, max_features=sqrt
Decision Tree	criterion=gini, max_depth=12, min_samples_split=10, min_samples_leaf=4, ccp_alpha=0.0001
SVM	kernel=RBF, C=10, gamma=scale, StandardScaler

LightGBM and XGBoost build predictions additively from a number of decision trees, that is, the outputs of all trees are summed to produce the final prediction. If $\boldsymbol{x_i}$ is the input feature vector, $\boldsymbol{f_k}$ is the k-th decision tree function, and $\boldsymbol{K}$ is the number of trees, then the model prediction is computed using Equation (4).

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{4}$$

Because this study is a multi-class classification with six classes, the raw score of each class is converted into a probability using the softmax function, and the final class is then determined through the argmax rule. If $\boldsymbol{z_{i,c}}$ is the raw score of class c for sample i and $\boldsymbol{C}$ is the number of classes, then the probability of each class $\boldsymbol{p_{i,c}}$ together with its predicted class is computed using Equation (5).

$$p_{i,c} = \frac{e^{z_{i,c}}}{\sum_{j=1}^{C} e^{z_{i,j}}}, \quad \hat{y}_i = \arg\max_c p_{i,c} \tag{5}$$

The model performance was evaluated on the same test data using four main metrics computed from the confusion matrix components, namely TP (true positive), TN (true negative), FP (false positive), and FN (false negative) [33]. These metrics are Accuracy as the proportion of correct predictions, Precision as the model's correctness when predicting a class, Recall as the model's ability to recognize all samples of a class, and F1-score as the harmonic mean of precision and recall. Because the classification is multi-class, Precision, Recall, and F1-score were computed per class and then macro-averaged (F1-macro) so that each class contributes equally.

In addition to these four metrics, ROC-AUC (macro, one-vs-rest) was measured as an indicator of the model's ability to separate classes. Because ROC-AUC can be overly optimistic on imbalanced data, the area under the precision-recall curve (PR-AUC) was also computed per class, as it is more sensitive to performance on minority classes such as Slowloris. The difference between training and test accuracy (train-test gap) as an indicator of overfitting, and the inference latency (the average prediction time per sample) and training time as indicators of efficiency. The last two metrics are important considerations because the model is intended for real-time detection.

RESULT AND DISCUSSION

Result

Dataset Characteristics

The self-collected dataset consists of 129,190 rows, each representing the aggregation of 50 traffic features over a one-second time window. The data were recorded from 108 collection sessions and grouped into six classes, namely one normal traffic class and five DoS attack classes (TCP_SYN, UDP_FLOOD, ICMP_FLOOD, HTTP_FLOOD, and SLOWLORIS). All rows contain only numerical features and no duplicates, so the dataset is ready for training and testing. The sample distribution of each class is shown in Figure 4(a).

The normal class is the largest class with 54,226 rows or 41.97% of the entire data, reflecting operational network conditions that are generally dominated by normal traffic. The five attack classes have relatively balanced proportions among one another, namely HTTP_FLOOD with 16,069 rows (12.44%), UDP_FLOOD 14,965 rows (11.58%), SLOWLORIS 14,934 rows (11.56%), ICMP_FLOOD 14,570 rows (11.28%), and TCP_SYN 14,426 rows (11.17%). Thus, each attack class contributes approximately 11-12% of the total data.

In terms of the number of sessions, the normal class was collected from 54 sessions, while the attack classes came from 8 to 15 sessions per class, namely TCP_SYN 15 sessions, UDP_FLOOD 11 sessions, HTTP_FLOOD and ICMP_FLOOD 10 sessions each, and SLOWLORIS 8 sessions. This variation in the number of sessions is consistent with the parameter-variation approach applied to each attack type. The comparison between the largest and smallest classes yields an imbalance ratio of about 3.76:1, so class balancing with SMOTE was applied to the training data as explained in the Preprocessing and Data Splitting section, and the result is described in the Training Data Balancing with SMOTE section.

Training Data Balancing with SMOTE

Because normal traffic dominated, the class distribution in the training data was initially imbalanced. In the training data with the 65% split (83,921 rows), the normal class amounted to 35,219 rows as the majority class, while each of the five attack classes had only about 9,000 to 10,000 rows. SMOTE was then applied to equalize each minority class

with the majority class to 35,219 rows, so that the amount of training data increased from 83,921 to 211,314 rows with the addition of 127,393 synthetic samples. The same balancing approach was applied to the 75% split for the non-boosting models. The class distribution of the training data after SMOTE balancing is shown in Figure 4(b).

(a) (b)

Figure 4. Class distribution of (a) the full dataset and (b) the training data after SMOTE

This balancing was deliberately limited to the training data only, while the validation and test data were left at their original distribution. Thus, model evaluation was still performed on the real class distribution and was not affected by synthetic samples, so that the test results reflect the performance under actual data conditions.

Comparison of the Five Models

The five algorithms were trained and tested on the same test data (32,340 samples), and the performance comparison is presented in Table 5. The two gradient boosting algorithms occupied the top positions, with LightGBM and XGBoost reaching an almost equal F1-macro above 96%, followed by Random Forest, while Decision Tree and SVM were clearly below them. This indicates a fairly clear performance gap between the boosting-based models and the rest.

Table 5. Performance comparison of the five models

Model	Acc (%)	Prec (%)	Rec (%)	F1-macro (%)	ROC-AUC	Gap (%)	Latency (ms)	Training (s)
LightGBM	96.22	97.41	95.38	96.25	0.9967	3.59	1.369	33.8
XGBoost	96.13	97.24	95.35	96.16	0.9968	3.67	9.549	61.1
Random Forest	94.88	95.27	94.10	94.60	0.9960	4.74	67.894	229.1
Decision Tree	91.78	92.12	90.30	90.84	0.9725	7.27	1.243	9.0
SVM	85.21	85.94	88.35	86.36	0.9812	9.10	5.741	3045.3

From the efficiency standpoint, the differences among the models were far more striking. As shown in Table 5, LightGBM combined a very low inference latency with a short training time, whereas Random Forest and SVM were far slower in latency and training respectively. Decision Tree reached the lowest latency but at the cost of much lower accuracy. The trade-off between F1-macro and inference latency

further shows that LightGBM occupies the most ideal position, namely the highest F1-macro with the lowest latency among the accurate models.

Based on all these metrics, LightGBM was selected as the final model because it combines the highest F1-macro, the smallest gap, low inference latency, and a short training time, making it the most suitable for real-time attack detection. Further analysis of the advantages of LightGBM is described in the following discussion section.

Detailed Evaluation of LightGBM

An in-depth evaluation of LightGBM is presented through the per-class classification report in Table 6, with an overall accuracy of 96.22% and macro averages of precision 97.41%, recall 95.38%, and F1-score 96.25%. Four attack classes were detected very well with F1-scores above 99%, namely UDP_FLOOD (99.67%), HTTP_FLOOD (99.59%), ICMP_FLOOD (99.49%), and TCP_SYN (99.46%). The normal class was also detected well with an F1-score of 95.71%, supported by a high recall of 98.10% although its precision was slightly lower (93.44%). In contrast, the SLOWLORIS class was the most difficult with an F1-score of 83.58%. Its precision was relatively high (92.53%) but its recall was much lower (76.21%), indicating that many Slowloris samples were not successfully recognized as Slowloris.

Table 6. Per-class classification report of LightGBM

Class	Precision (%)	Recall (%)	F1-score (%)	PR-AUC	Support
HTTP_FLOOD	99.36	99.83	99.59	0.9999	4021
ICMP_FLOOD	99.83	99.15	99.49	0.9997	3647
SLOWLORIS	92.53	76.21	83.58	0.9271	3737
TCP_SYN	99.45	99.47	99.46	0.9996	3611
UDP_FLOOD	99.84	99.49	99.67	0.9999	3745
normal	93.44	98.10	95.71	0.9912	13579
Macro average	97.41	95.38	96.25	0.9862	32340

The source of these errors is clearly seen in the confusion matrix in Figure 5. Of the 3,737 Slowloris samples, 2,848 were correctly recognized while the remaining 889 were classified as normal traffic. In the opposite direction, 221 of the 13,579 normal samples were predicted as Slowloris. This confusion occurred almost entirely between Slowloris and normal, while the other attack classes were hardly ever confused and had only a handful of errors. The same pattern is reflected in the per-class ROC-AUC, namely the four flooding attack classes reached values close to 1.0 and normal 0.9939, while SLOWLORIS obtained the lowest value of 0.9866, although it remained very high. The gap becomes far more visible in the PR-AUC, where SLOWLORIS drops to 0.9271 while the other classes remain above 0.99. This contrast confirms that macro ROC-AUC can be overly optimistic on imbalanced data, since the precision-recall view exposes the weakness of the minority Slowloris class much more clearly than ROC-AUC alone.

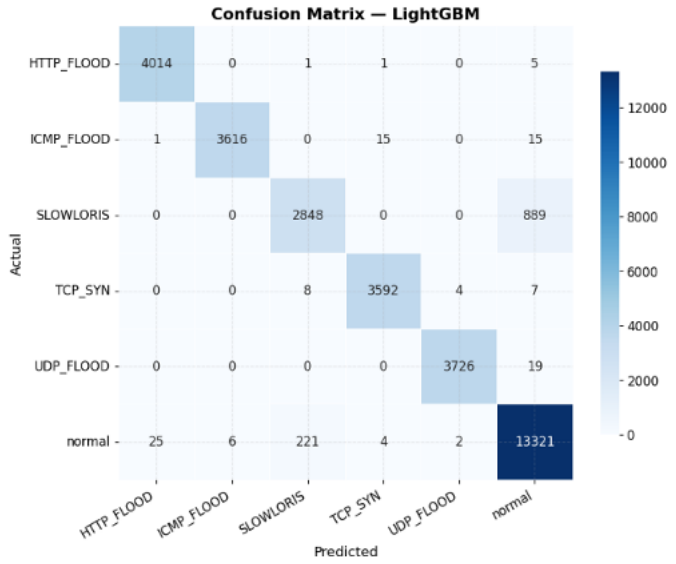

Figure 5. Confusion matrix of LightGBM

Figure 6 shows the 15 most important features according to LightGBM. The rx_tx_ratio feature ranks first, followed by udp_flows, icmp_connections, tx_mbps, and bps_per_conn in the top five. The subsequent ranks are dominated by flow- and connection-based features such as total_bytes_nf, tcp_flows, and tcp_connections, while the source-concentration feature top_src_ip_pct, which is closely related to conn_ratio, is also among the top 15. This composition shows that traffic asymmetry, the flow composition per protocol, and the connection volume are the signals most used by the model to distinguish the six classes.

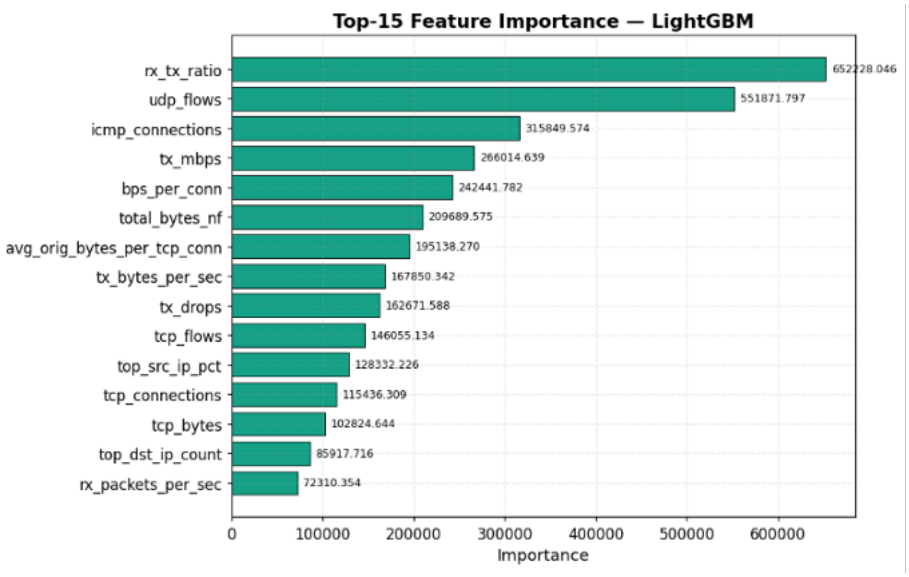

Figure 6. Fifteen most important features of LightGBM

Ablation Study on Feature Reduction

To verify whether all 50 features are necessary and to examine the trade-off between detection performance and inference cost, an ablation study was conducted by retraining LightGBM using only the top 15 features identified in Figure 6. Both models used the same temporal holdout split, the same SMOTE oversampling on the training data, and the same hyperparameters, so that the only difference was the number of input features. The comparison is presented in Table 7. The latency and training time in Table 7 were re-measured during the ablation run, so they differ slightly from the values reported in Table 5.

Table 7. Comparison of the full 50-feature model and the reduced top-15 model

Metric	LightGBM (50 features)	LightGBM (top-15 features)
Accuracy (%)	96.22	95.82
Precision-macro (%)	97.41	96.39
Recall-macro (%)	95.38	95.51
F1-macro (%)	96.25	95.92
ROC-AUC (macro)	0.9967	0.9975
PR-AUC (macro)	0.9862	0.9890
Train-test gap (%)	3.59	3.71
Inference latency (ms)	1.457	1.210
Training time (s)	43.1	18.1

As shown in Table 7, reducing the feature set from 50 to 15 lowers the F1-macro only slightly, from 96.25% to 95.92%, a decrease of 0.33 percentage points, while accuracy decreases from 96.22% to 95.82%. In exchange, the inference latency drops from 1.457 ms to 1.210 ms, about 17% faster, and the training time drops from 43.1 s to 18.1 s, about 58% faster. The ROC-AUC and PR-AUC even increase slightly, which indicates that the removed features contributed little discriminative information and that the reduced model generalizes comparably. This result confirms that the 50-feature model is not excessively redundant, since removing 70% of the features still costs a small amount of macro performance, yet it also shows that a lighter top-15 configuration is an attractive option when computational resources are limited.

A closer look at the Slowloris class reveals a more nuanced effect. With the reduced feature set, the recall of Slowloris increases from 76.21% to 81.24%, because the model relies less on volumetric features that are weak for low-and-slow attacks, so fewer Slowloris windows are misread as normal, with the number

falling from 889 to 701. However, this comes at the cost of precision, which drops from 92.53% to 87.80%, as the number of normal windows misclassified as Slowloris rises from 221 to 413. This trade-off explains why the F1-score of Slowloris changes only marginally and confirms that the difficulty in detecting Slowloris stems from the limited ability of volumetric traffic features to characterize this attack, rather than from the number of features alone.

Real-Time Detection and Mitigation Implementation

The application of the model to a real-time detection system was demonstrated through the monitoring interface for one of the target virtual machines, namely a web server with the address 10.10.60.100. Under the normal condition as shown in Figure 7(a), the LightGBM model classified the traffic as normal with a confidence level of 98.3% and probabilities of all attack classes close to zero. The resource usage under this condition was also low, with a CPU load of only 0.1% and very little network traffic, reflecting an operational state without disruption.

When an attack was launched as shown in Figure 7(b), the status of the virtual machine changed to under attack and the model immediately recognized the traffic as HTTP_FLOOD with a confidence level of 99.1%. This change was accompanied by a surge in the virtual machine CPU load to 58.3% and a sharp increase in traffic volume compared with the normal condition. In addition to classifying the attack type, the system also performed source attribution by identifying the attacker IP address 20.20.20.112 along with the number of packets sent, namely 170,507 packets, so that the response could be directed straight at the attack source.

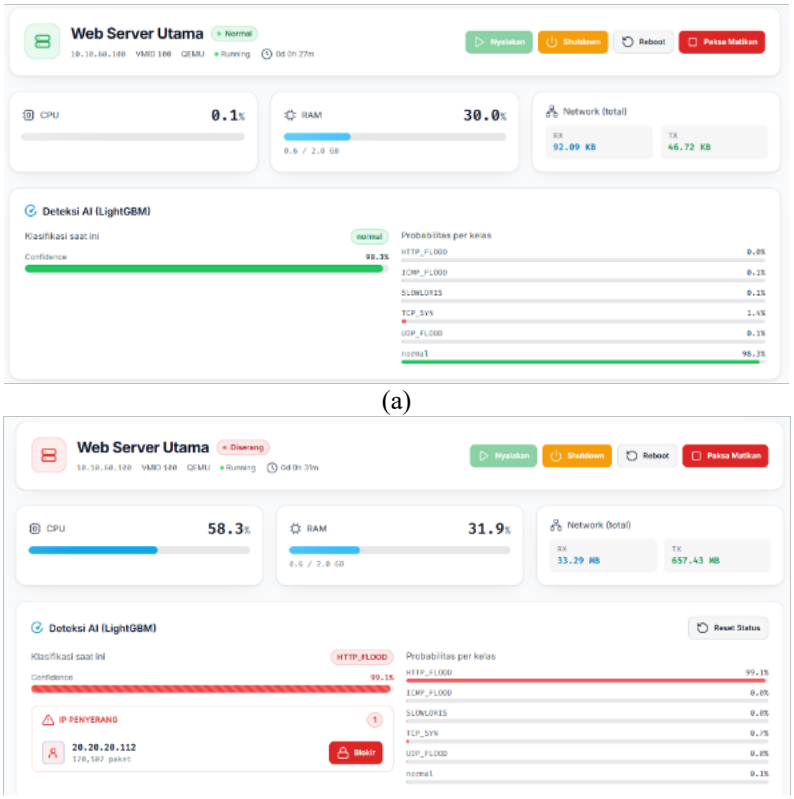

Figure 7. Monitoring interface during (a) normal and (b) HTTP flood conditions

Based on these detection results, the system automatically applied the mitigation action by creating a blocking rule on the MikroTik gateway as shown in Figure 8. The rule operates on the prerouting chain with a drop action against the attacker IP address toward the target machine and is equipped with a note recording the attack type and the model confidence level. The blocking duration can be configured so that access can be automatically restored after a certain period. Thus, the entire process from detection to mitigation runs as a closed loop without manual intervention, which demonstrates the feasibility of the system for real-time network protection.

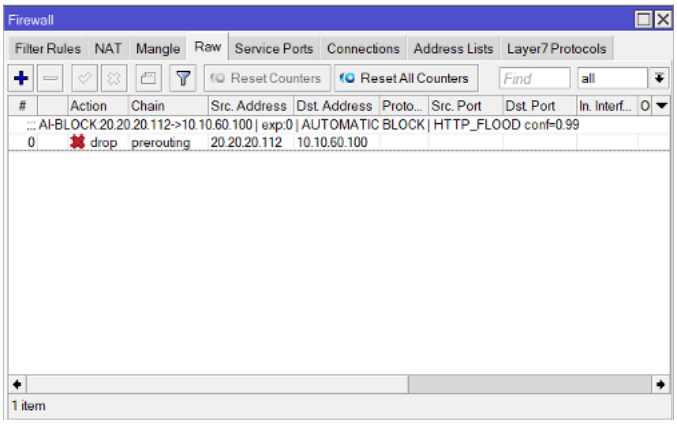

Figure 8. Automatic blocking rule on the MikroTik gateway

To quantify the speed of this closed loop, the mitigation latency was measured as the time from the moment the model triggers a block until the drop rule becomes active on the MikroTik gateway. The measurement was repeated over 25 events across the five attack types. Because the latency distribution is right-skewed, caused by occasional reconnection of the router API session, the median is reported as a more representative value than the mean. The overall median mitigation latency was 56.2 ms, with an interquartile range of 48.6 ms to 84.3 ms. This value is far below the one-second detection window, which means that once an attack is confirmed, the blocking rule is enforced almost immediately relative to the detection interval. Table 8 summarizes the mitigation latency for each attack type.

Table 8. Mitigation latency per attack type

Attack type	N	Median (ms)	Range (ms)
TCP_SYN	5	266.6	55.1 – 521.8
UDP_FLOOD	5	81.2	56.2 – 241.5
ICMP_FLOOD	5	46.0	6.6 – 859.6
HTTP_FLOOD	5	50.4	48.6 – 63.8
SLOWLORIS	5	48.0	6.0 – 64.6
Overall	25	56.2	6.0 – 859.6

To observe the effect of mitigation on the gateway load, the MikroTik CPU utilization was recorded before, during, and after an HTTP flood, as shown in Figure 9. In this scenario the attack was deliberately sustained and the blocking was triggered manually at a chosen moment, so that the rise and fall of the CPU load could be observed clearly, whereas the automatic mitigation speed itself is reported separately as the median latency above. During the baseline period the CPU load stayed near 2%. Once the attack started, the CPU load rose and fluctuated up to about 31%, following the bursty nature of the sustained HTTP flood. After the drop rule was applied on the prerouting chain, the attack traffic was discarded before entering connection tracking, and the CPU load returned to its baseline level even though the attack was still running on the attacker side. This confirms that the mitigation not only installs a rule but effectively removes the attack load from the gateway.

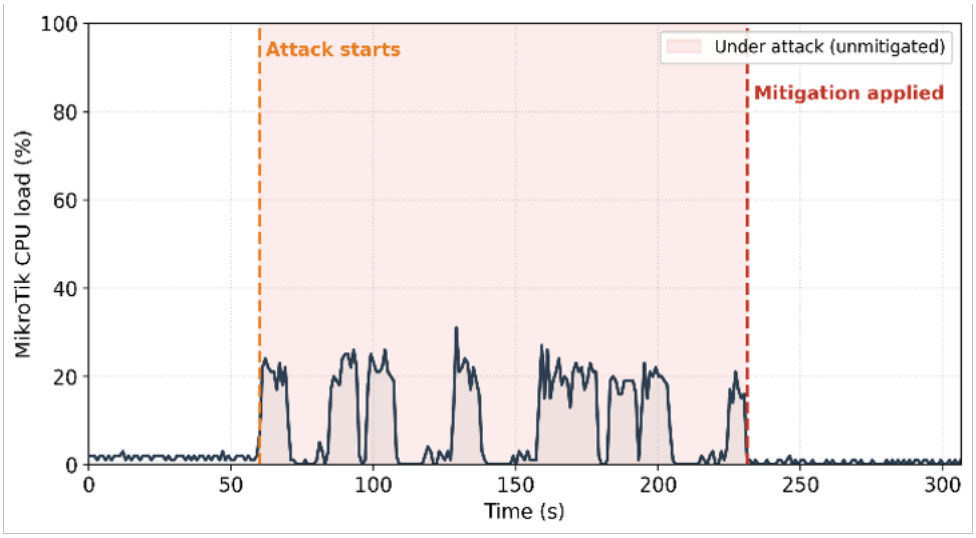

Figure 9. MikroTik CPU load before, during, and after mitigation of an HTTP flood

DISCUSSION

Effectiveness of LightGBM for Multi-Class DoS Detection

The performance hierarchy in Table 5 is consistent, namely the two gradient boosting models occupy the top positions, followed by Random Forest, Decision Tree, and SVM. This pattern aligns with the known superiority of ensemble and boosting methods in intrusion detection, which combine many weak learners for better generalization and resistance to overfitting [13], as reflected in the small train-test gap of LightGBM (3.59%). However, LightGBM was selected not on accuracy alone but on the balance between accuracy and efficiency, since it obtained the highest F1-macro together with the lowest inference latency among the accurate models [22]. Although XGBoost can slightly outperform LightGBM depending on the dataset and configuration [13], the boosting group is consistently at the top, and on this real multi-class data LightGBM slightly outperformed XGBoost with much lower latency, making it the most appropriate choice for real-time detection.

This latency advantage is rooted in the architecture of LightGBM described earlier. The leaf-wise tree growth produces accurate trees with fewer nodes, while Gradient-based One-Side Sampling and histogram-based feature binning reduce the amount of computation during both training and inference. These mechanisms explain why LightGBM reaches an accuracy comparable to XGBoost while being several times faster, which is the property that matters most for real-time mitigation.

Challenges in Slowloris Detection and the Role of Features

Among all classes, Slowloris was the most difficult to recognize, with a recall of only 76.21%. The confusion matrix in Figure 5 shows that, of the 3,737 Slowloris samples, 889 were read as normal traffic and 221 normal samples were read as Slowloris. This confusion, concentrated between Slowloris and normal, arises from the low-and-slow nature of the attack, which uses a low traffic volume with connections left open for a long time so that it resembles a legitimate user and penetrates traditional detection [9]. It also creates a precision-recall tension, because recognizing all Slowloris samples risks marking normal traffic as an attack.

The root of this difficulty is evident in the feature ranking in Figure 6, which is dominated by volumetric and flow-based features such as rx_tx_ratio, udp_flows, and tx_mbps. These features capture high-volume flooding well but are less sensitive to the low-volume Slowloris. The dominance of rx_tx_ratio and top_src_ip_pct validates the role of traffic asymmetry and source concentration in Equations (2) and (1). Consistent with the literature, Slowloris is better detected when specific features such as traffic entropy and the number of unique ports are used [9], so the lower recall here is a consequence of relying on common volumetric features and points to an opportunity for features more sensitive to low-and-slow behavior.

The ablation study reinforces this interpretation. When the feature set was reduced to the top 15, the recall of Slowloris rose to 81.24% because the model depended less on volumetric features, yet its precision fell from 92.53% to 87.80% as more normal windows were flagged as Slowloris. The fact that removing features improved recall but hurt precision confirms that the difficulty stems not from the number of features but from the limited ability of volumetric signals to characterize low-and-slow traffic. This weakness is also captured more honestly by the PR-AUC, which dropped to 0.9271 for Slowloris while remaining above 0.99 for the other classes, showing that the precision-recall view is more informative than ROC-AUC for the minority class on imbalanced data.

Real-Time Feasibility and the Integration of Detection and Mitigation

The feasibility of real-time application is determined by the model's ability to complete inference faster than the data arrival rate. With a latency of only 1.369 ms per sample, LightGBM operates well within the one-second detection window, so classification keeps up with the traffic flow without accumulation. This matters because conventional IDS often cannot handle real-time detection on high-speed networks due to the heavy computational load [26], making a lightweight, low-latency model more suitable for latency-sensitive applications [22]. Figure 7(a) and Figure 7(b) reinforce this by showing the model recognizing both normal and attack conditions directly.

Beyond detection, this study unites detection, source attribution, and blocking in a single automatic closed loop. Many previous systems stop at the detection stage without connecting to network devices, even though firewall automation can speed up the response to attacks [21]. As shown in Figure 8, the detection results are directly translated into automatic blocking rules on the MikroTik gateway.

The speed of this loop was also quantified. The mitigation latency, measured from the moment an attack is confirmed until the drop rule becomes active on the gateway, had a median of 56.2 ms, which is far below the one-second detection window. This means the response is not only automatic but also fast relative to the detection interval. The CPU measurement in Figure 9 further shows that once the rule is enforced on the prerouting chain, the attack load on the gateway returns to its baseline level, confirming that the mitigation is effective in practice and not merely a rule insertion.

Comparison with Previous Studies and Limitations

The accuracy of LightGBM in this study, namely 96.22%, is within a range comparable to previous studies, such as Random Forest 97.20% on CIC-IDS 2017 [14], a LightGBM-based model 99.97% on TON-IoT [14], and XGBoost 99.67% on UNSW-NB15 [15]. However, this comparison is not fully equivalent because of differences in datasets and classification tasks. Those studies generally use public datasets and optimize a single algorithm, whereas this study evaluates LightGBM as the proposed detection model against four baseline algorithms on a dataset from real infrastructure, and at the same time unites real-time detection with automatic mitigation. Its main contribution is therefore not the highest accuracy but the balance between accuracy and efficiency on real data and the integration with mitigation. This is consistent with studies that emphasize both accuracy and low latency [16], with the difference that that study uses a CNN-based feature extraction stage to reduce the prediction time whereas this study retains all features to avoid data leakage. The difficulty in recognizing minority classes due to data imbalance reported previously [14], [16] is also consistent with the challenge in the Slowloris class here.

This study has several limitations. The attacks examined are single-source (DoS) and do not yet include DDoS, the testing was conducted on a single testbed with a single topology, the recall of the Slowloris class can still be improved, and the data were collected in a controlled environment. Therefore, future development can be directed toward extension to multi-source DDoS scenarios, the addition of temporal features or features more sensitive to low-and-slow characteristics to strengthen Slowloris detection, and testing on actual production traffic.

CONCLUSION

This study aimed to build and evaluate a real-time multi-class DoS attack detection system on Proxmox virtual machines integrated with MikroTik, with LightGBM as the proposed model and four other algorithms as baselines. From the fair comparison, LightGBM was selected as the best model with an F1-macro of 96.25% and an inference latency of 1.369 ms, offering the best balance between accuracy and efficiency for real-time application. The per-class evaluation showed that the four flooding attacks were detected almost perfectly, while Slowloris remained the greatest challenge because its low-and-slow

behavior resembles normal traffic. Beyond detection, the system ran a closed loop from attack classification and source identification to automatic blocking on the gateway, with a median mitigation latency of 56.2 ms and a measurable reduction of the gateway load after mitigation.

Theoretically, these findings indicate that leaf-wise gradient boosting is well suited to real network telemetry, since it attains accuracy comparable to level-wise boosting at a much lower inference cost, and that the precision-recall view is more reliable than ROC-AUC for assessing minority attack classes on imbalanced data. Practically, the study shows that a lightweight model can detect and mitigate attacks in a single automated loop on commodity hardware, and the ablation study offers a concrete deployment option, since a reduced set of 15 features lowers the inference latency and training time substantially at the cost of only a small decrease in macro performance.

This study also has several limitations. The attacks examined are single-source DoS and do not yet cover distributed DDoS, the data were collected in a controlled testbed rather than production traffic, and the detection of Slowloris still leaves room for improvement because the extracted features are predominantly volumetric. Future work can therefore be directed toward extension to multi-source DDoS scenarios, evaluation on real production traffic, and the addition of temporal or entropy-based features that are more sensitive to low-and-slow attacks, so that the detection of classes such as Slowloris can be strengthened further.

REFERENCES

[1] O. Abied, O. Ibrahim, and S. N.-I. Mat Kamal, "Adoption of Cloud Computing in E-Government: A Systematic Literature Review," *Pertanika J. Sci. Technol.*, vol. 30, no. 1, pp. 655–689, 2022, doi: 10.47836/pjst.30.1.36.

[2] S. Lozano, T. Lugo, and J. Carretero, "A Comprehensive Survey on the Use of Hypervisors in Safety-Critical Systems," *IEEE Access*, vol. 11, pp. 36244–36263, 2023, doi: 10.1109/access.2023.3264825.

[3] A. Coscia, V. Dentamaro, S. Galantucci, A. Maci, and G. Pirlo, "Automatic decision tree-based NIDPS ruleset generation for DoS/DDoS attacks," *Journal of Information Security and Applications*, vol. 82, p. 103736, 2024, doi: 10.1016/j.jisa.2024.103736.

[4] T. H. H. Aldhyani and H. Alkahtani, "Artificial Intelligence Algorithm-Based Economic Denial of Sustainability Attack Detection Systems: Cloud Computing Environments," *Sensors*, vol. 22, no. 13, p. 4685, 2022, doi: 10.3390/s22134685.

[5] Z. R. Alashhab, M. Anbar, M. M. Singh, I. H. Hasbullah, P. Jain, and T. A. Al-Amiedy, "Distributed Denial of Service Attacks against Cloud Computing Environment: Survey, Issues, Challenges and Coherent Taxonomy," *Applied Sciences*, vol. 12, no. 23, p. 12441, 2022, doi: 10.3390/app122312441.

[6] M. Zadnik and E. Carasec, "AI infers DoS mitigation rules," *J. Intell. Inf. Syst.*, vol. 60, no. 2, pp. 305–324, 2022, doi: 10.1007/s10844-022-00728-2.

[7] M. Roshani and M. Nobakht, "HybridDAD: Detecting DDoS Flooding Attack using Machine Learning with Programmable Switches," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, in ARES 2022. ACM, Aug. 2022, pp. 1–11. doi: 10.1145/3538969.3538991.

[8] H. Tang, S. S. Kolahi, and L. Thompson, "Evaluation of HTTP Flood DDoS Cyber Attack on Apache2 Web Server with Linux Ubuntu 22.04," in *2023 IEEE International Conference on Computing (ICOCO)*, IEEE, Oct. 2023, pp. 53–58. doi: 10.1109/icoco59262.2023.10398152.

[9] V. Rios, P. Inacio, D. Magoni, and M. Freire, "Detection of Slowloris Attacks using Machine Learning Algorithms," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, in SAC '24. ACM, Apr. 2024, pp. 1321–1330. doi: 10.1145/3605098.3635919.

[10] S. Ang, S. Huy, and M. Janarthanan, "Utilizing IDS and IPS to Improve Cybersecurity Monitoring Process," *Journal of Cyber Security and Risk Auditing*, vol. 2025, no. 3, pp. 77–88, 2025, doi: 10.63180/jcsra.thestap.2025.3.10.

[11] A. Pinto, L.-C. Herrera, Y. Donoso, and J. A. Gutierrez, "Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure," *Sensors*, vol. 23, no. 5, p. 2415, 2023, doi: 10.3390/s23052415.

[12] T.-H. Chua and I. Salam, "Evaluation of Machine Learning Algorithms in Network-Based Intrusion Detection Using Progressive Dataset," *Symmetry (Basel).*, vol. 15, no. 6, p. 1251, 2023, doi: 10.3390/sym15061251.
[13] A. Özçelebi and V. Marttin, "Comparative Evaluation of Ensemble and Tree-Based Machine Learning Algorithms for Network Intrusion Detection," *Journal of Electronic & Information Systems*, vol. 7, no. 2, pp. 116–131, 2025, doi: 10.30564/jeis.v7i2.12299.
[14] P. V Chavan and N. V Alone, "Optimizing Intrusion Detection with Random Forest: A High-Accuracy Approach Using CIC-IDS 2017," *Int. J. Comput. Appl.*, vol. 187, no. 3, pp. 17–22, 2025, doi: 10.5120/ijca2025924816.
[15] H. Yulianton, F. A. Sutanto, and R. C. Noor Santi, "Optimized Network Intrusion Detection Using XGBoost with Hyperparameter Tuning: An Empirical Study on UNSW-NB15 Dataset," *Journal of Software Engineering and Simulation*, vol. 11, no. 8, pp. 01–07, 2025, doi: 10.35629/3795-11080107.
[16] G. ZHAO, Y. WANG, and J. WANG, "Intrusion Detection Model of Internet of Things Based on LightGBM," *IEICE Transactions on Communications*, vol. E106.B, no. 8, pp. 622–634, 2023, doi: 10.1587/transcom.2022ebp3169.
[17] P. Dini, A. Elhanashi, A. Begni, S. Saponara, Q. Zheng, and K. Gasmi, "Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity," *Applied Sciences*, vol. 13, no. 13, p. 7507, 2023, doi: 10.3390/app13137507.
[18] Y. Sahli, "comparison of the NSL-KDD dataset and its predecessor the KDD Cup '99 dataset," *International Journal of Scientific Research and Management*, vol. 10, no. 04, pp. 832–839, 2022, doi: 10.18535/ijsrm/v10i4.ec05.
[19] D. Pinto, I. Amorim, E. Maia, and I. Praça, "A review on intrusion detection datasets: tools, processes, and features," *Computer Networks*, vol. 262, p. 111177, 2025, doi: 10.1016/j.comnet.2025.111177.
[20] E. K. Viegas, A. O. Santin, and P. Tedeschi, "Toward a Reliable Evaluation of Machine Learning Schemes for Network-Based Intrusion Detection," *IEEE Internet of Things Magazine*, vol. 6, no. 2, pp. 70–75, 2023, doi: 10.1109/iotm.001.2300106.
[21] H. A. Damanik and M. Anggraeni, "Sistem Deteksi dan Pencegahan System Attack pada Infrastruktur Jaringan Menggunakan Realtime Honeypot dan Automatic Iptables," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 12, no. 5, pp. 1173–1184, 2025, doi: 10.25126/jtiik.2025126.
[22] V. R. Joshi, K. Assa-Agyei, T. Al-Hadhrami, and S. N. Qasem, "Hybrid AI Intrusion Detection: Balancing Accuracy and Efficiency," *Sensors*, vol. 25, no. 24, p. 7564, 2025, doi: 10.3390/s25247564.
[23] R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, "A Deep Learning Ensemble Approach to Detecting Unknown Network Attacks," *Journal of Information Security and Applications*, vol. 67, p. 103196, 2022, doi: 10.1016/j.jisa.2022.103196.
[24] V. Z. Mohale and I. C. Obagbuwa, "Evaluating machine learning-based intrusion detection systems with explainable AI: enhancing transparency and interpretability," *Front. Comput. Sci.*, vol. 7, 2025, doi: 10.3389/fcomp.2025.1520741.
[25] S. P. Thirimanne, L. Jayawardana, L. Yasakethu, P. Liyanaarachchi, and C. Hewage, "Deep Neural Network Based Real-Time Intrusion Detection System," *SN Comput. Sci.*, vol. 3, no. 2, 2022, doi: 10.1007/s42979-022-01031-1.
[26] A. Singh, "Real Time Intrusion Detection In Edge Computing Using Machine Learning Techniques," *Turkish Journal of Engineering*, vol. 9, no. 2, pp. 385–393, 2025, doi: 10.31127/tuje.1516046.
[27] M. A. Muslim *et al.*, "New model combination meta-learner to improve accuracy prediction P2P lending with stacking ensemble learning," *Intelligent Systems with Applications*, vol. 18, p. 200204, 2023, doi: 10.1016/j.iswa.2023.200204.
[28] D. Li, Z. Yang, S. Yu, M. Duan, and S. Yang, "A Micro-Segmentation Method Based on VLAN-VxLAN Mapping Technology," *Future Internet*, vol. 16, no. 9, p. 320, 2024, doi: 10.3390/fi16090320.
[29] M. Najafimehr, S. Zarifzadeh, and S. Mostafavi, "DDoS attacks and machine-learning-based detection methods: A survey and taxonomy," *Engineering Reports*, vol. 5, no. 12, 2023, doi: 10.1002/eng2.12697.
[30] A. A. Bahashwan *et al.*, "HLD-DDoSDN: High and low-rates dataset-based DDoS attacks against SDN," *PLoS One*, vol. 19, no. 2, p. e0297548, 2024, doi: 10.1371/journal.pone.0297548.

[31] D. Gusak, A. Volodkevich, A. Klenitskiy, A. Vasilev, and E. Frolov, "Time to Split: Exploring Data Splitting Strategies for Offline Evaluation of Sequential Recommenders," in *RecSys2025 - Proceedings of the 19th ACM Conference on Recommender Systems*, Association for Computing Machinery, Inc, Aug. 2025, pp. 874–883. doi: 10.1145/3705328.3748164.
[32] D. Elreedy, A. F. Atiya, and F. Kamalov, "A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning," *Mach. Learn.*, vol. 113, no. 7, pp. 4903–4923, 2023, doi: 10.1007/s10994-022-06296-4.
[33] M. K. Suryadewiansyah and T. E. E. Tju, "Naïve Bayes dan Confusion Matrix untuk Efisiensi Analisa Intrusion Detection System Alert," *Jurnal Nasional Teknologi dan Sistem Informasi*, vol. 8, no. 2, pp. 81–88, 2022, doi: 10.25077/teknosi.v8i2.2022.81-88.