Classifying Denial of Service Attacks Using Fast Machine Learning Algorithms

Zhida Li, Ana Laura Gonzalez Rios, and Ljiljana Trajković

Abstract— **Denial of service attacks are harmful cyberattacks that diminish Internet resources and services. Hence, detecting these cyberattacks is a topic of great interest in cybersecurity. Using traditional machine learning approaches in intrusion detection systems requires long training time and has high computational complexity. Thus, we evaluate performance of fast machine learning algorithms for training and generating models to detect denial of service attacks in communication networks. We use synthetically generated datasets that captured Transmission Control Protocol and User Datagram Protocol network flows in a controlled testbed laboratory environment. Evaluated algorithms include broad learning system and its extensions as well as XGBoost, LightGBM, and CatBoost gradient boosting decision tree algorithms. Experiments indicate that boosting algorithms often require shorter training time and have better performance.**

Index Terms— **Network anomalies, denial of service, machine learning, broad learning system, gradient boosting.**

I. Introduction

Cybercriminals exploit vulnerabilities of communication networks and systems to execute denial of service (DoS) and distributed DoS (DDoS) attacks that compromise the availability of resources to legitimate users by flooding the network and by overwhelming servers with a large number of requests. DoS attacks are performed from a single system while DDoS attacks are synchronized and executed from multiple systems. They may be classified as floods, fragmentation, Transport Control Protocol (TCP) state exhaustion, and application-layer [4]:

Floods are attacks that overload a target by a voluminous traffic. A botnet, collection of devices infected with malware under control of a botmaster, floods a victim by consuming its bandwidth with User Datagram Protocol (UDP) or Internet Control Message Protocol (ICMP) packets.

Fragmentation attacks are executed by sending manipulated network packets that cannot be reassembled due to large packet headers.

TCP state exhaustion attacks (protocol attacks) usually target firewalls, load balancers, and servers by sending large Internet Protocol (IP) or TCP Synchronize (SYN) packets.

Application-layer attacks monopolize Simple Mail Transfer Protocol (SMTP), Hypertext Transfer Protocol Secure (HTTPS), and Domain Name System (DNS) services. These types of attacks are the most challenging to detect because the requests seem legitimate.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant R31-611284.

The authors are with the School of Engineering Science, Simon Fraser University, Vancouver, British Columbia, Canada V5A 1S6, {zhidal, anag, ljilja}@sfu.ca

Detection techniques for DoS and DDoS attacks include activity profiling, change-point detection, and wavelet analysis [6]. In activity profiling, headers of network packets are monitored to estimate the average packet rate of inbound and outbound flows by deriving an activity profile based on analysis of consecutive packets and similarity of packet fields. Change-point detection creates a time series by clustering traffic data based on the address, port, or protocol. Network traffic is described using wavelet analysis to calculate spectral components and separate anomalous events from regular network activity. Recent intrusion detection techniques [16], [17] also rely on machine learning algorithms [5], [12], [18] including support vector machine (SVM) and deep neural networks such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders, and multilayer perceptrons.

Detection of DoS and DDoS attacks that rely on machine learning algorithms require updating or retraining generated models in order to capture deviations from regular network activities caused by ever-evolving and zero-day cyberattacks. Therefore, the training time is important for the decision-making process at the onset of anomalies when preventing cyberattacks on servers and avoiding denial of service to legitimate users. Hence, we evaluate fast machine learning algorithms including: broad learning system (BLS) [7] and its extensions [8], [14], [15] as well as the gradient boosting decision tree (GBDT) algorithms: eXtreme gradient boosting (XGBoost) [9], light gradient boosting machine (LightGBM) [13], and categorical boosting (CatBoost) [19]. BLS is a fast machine learning algorithm that relies on pseudo-inverse during the training process and has a single layer feed-forward neural network. Training time of GBDT algorithms is optimized by iteratively constructing an ensemble of decision trees and using functional gradient descent.

Datasets [1] capturing DoS and DDoS attacks have been generated by the Canadian Institute for Cybersecurity (CIC) using a testbed framework [20]. These datasets are synthetically generated by profiling behavior of regular and malicious users. Malicious behavior is modeled based on common techniques used to execute DoS and DDoS attacks. The CICIDS2017 and CSE-CIC-IDS2018 datasets include application-layer DoS attacks while the CICDDoS2019 datasets contain TCP, UDP, and TCP/UDP DDoS attacks. We use these recent synthetic datasets to compare performance of various BLS and GBDT models.

In this paper, we compare performance of BLS and its extensions as well as the XGBoost, LightGBM, and CatBoost GBDT algorithms based on the training time, accuracy, F-

Score, precision, sensitivity, and confusion matrix. We introduce the evaluated machine learning algorithms in Section II. The CICIDS2017, CSE-CIC-IDS2018, and CICDDoS2019 datasets are described in Section III while the experimental procedure and performance evaluation are given in Section IV. We conclude with Section V.

II. Machine Learning Algorithms

Matrix $\boldsymbol{X}$ used for training and test of various supervised machine learning algorithms contains N input data points (rows) and F features (columns). In the training phase, output matrix $\boldsymbol{Y}$ contains assigned labels: 0 (regular) and 1 (anomalous) data points.

A. Broad Learning System

Broad learning system [7], [8] is based on the random vector functional-link neural network. It consists of a set of n mapped features ($\boldsymbol{Z}^n$) and m enhancement nodes ($\boldsymbol{H}^m$) that are concatenated in a single layer feed-forward neural network to form a broad instead of a deep network. Each group of mapped features $\boldsymbol{Z}_i$ consists of n_1 nodes generated by first multiplying matrix $\boldsymbol{X}$ by randomly generated weights $\boldsymbol{W}_{e_i}$ and including randomly generated bias $\boldsymbol{\beta}_{e_i}$. A mapping ϕ is then used to generate mapped features:

$$\boldsymbol{Z}_i = \phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}), i = 1, 2, ..., n. \tag{1}$$

Matrix $\boldsymbol{Z}_i$ of dimension $N \times n_1$ corresponds to a single group of mapped features. The dimension of matrix $\boldsymbol{W}_{e_i}$ is $F \times n_1$. Matrix $\boldsymbol{Z}^n$ of dimension $N \times (n_1 \times n)$ is the concatenation of generated mapped features $\boldsymbol{Z}_i$. Each enhancement node $\boldsymbol{H}_j$ is created by first multiplying the concatenated groups of mapped features with random weights $\boldsymbol{W}_{h_j}$ and by adding random bias $\boldsymbol{\beta}_{h_j}$. The dimension of $\boldsymbol{W}_{h_j}$ is $(n_1 \times n) \times m$. A mapping ξ is then applied to generate enhancement nodes:

$$\boldsymbol{H}_j = \xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}), j = 1, 2, ..., m. \tag{2}$$

Mapped features and enhancement nodes are then concatenated to form the state matrix $\boldsymbol{A}_n^m$.

In the training phase, the Moore-Penrose pseudo-inverse or ridge regression is used to invert the state matrix and calculate the output weights $\boldsymbol{W}_n^m$ for the given labels $\boldsymbol{Y}$:

$$\boldsymbol{W}_n^m = (\lambda \boldsymbol{I} + (\boldsymbol{A}_n^m)^T \boldsymbol{A}_n^m)^{-1} (\boldsymbol{A}_n^m)^T \boldsymbol{Y}, \tag{3}$$

where λ is the regularization coefficient and $\boldsymbol{I}$ is the identity matrix whose dimension is $(n_1 \times n) + m$. During testing, the predicted labels are calculated using the output weights.

BLS extensions include incremental learning [7], radial basis function network (RBF-BLS) [15], cascades of mapped features (CFBLS), enhancement nodes (CEBLS), and both mapped features and enhancement nodes (CFEBLS) [8]. Incremental BLS algorithms allow increments of input data, mapped features, and/or enhancement nodes and, thus, enable dynamical updates of BLS models. In case of incremental input data, additional data points are used to recalculate mapped features and enhancement nodes. The calculation is done only for the additional data points and there is no need to include all previously considered data points. One may also create additional mapped features by increasing their number and recalculating/updating the weights of enhancement nodes. A similar approach is applied when creating new enhancement nodes based on the existing mapped features. Output weights have to be recalculated in all incremental cases. The RBF-BLS extension employs the Gaussian radial basis function to generate enhancement nodes. The cascades of mapped features and enhancement nodes add depth to the original architecture of BLS and may improve its performance.

Based on the its broad hidden layer and the use of pseudo-inverse or ridge regression, BLS offers shorter training time with comparable performance to deep learning networks such as multilayer perceptron, deep belief networks, deep Boltzmann machines, and convolutional neural networks [7]. Recent studies also reported comparable performance when applying BLS to data collected from communication networks [11], [14].

B. Gradient Boosting Decision Tree Algorithms

Boosting algorithms, a class of ensemble learning, are greedy algorithms that sequentially include estimators (base learners) to enhance the model performance [18]. The goal is to minimize the loss function by including estimators that are trained based on residuals. Residuals (the difference between the target and predicted values) are calculated in each iteration and are used as the target values in the next iteration. The forward stage-wise additive modeling is used to generate boosting models. The number of training iterations is equivalent to the number of estimators because a new estimator is added to the boosting model in each iteration. Boosting models employ loss functions such as squared error, absolute error, exponential loss, or log-loss.

The gradient boosting machines (GBMs) [10] are boosting algorithms that employ functional gradient descent to minimize the loss function. GBDT is a GBM variant that employs decision trees as estimators. Optimized GBDT algorithms include XGBoost [9], LightGBM [13], and CatBoost [19].

When training a GBDT model [9], [18] with K estimators using N data points, the predicted output is:

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

where f_k is the k^{th} decision tree and $\boldsymbol{x}_i$ is the i^{th} data point. Note that in the experiments, $\boldsymbol{x}_i$ is a row vector of matrix $\boldsymbol{X}$ containing input data and represents one collection sample. In the k^{th} iteration, predicted output is evaluated using the k^{th} decision tree (estimator):

$$\hat{y}_i^{(k)} = \hat{y}_i^{(k-1)} + f_k(\boldsymbol{x}_i), \tag{5}$$

where $\hat{y}_i^{(k)}$ is the predicted output of the i^{th} data point and $\hat{y}_i^{(k-1)}$ is the previously predicted output. The goal of the GBDT models is to minimize the objective function:

$$\mathcal{L}^{(k)} = \sum_{i=1}^{N} l(y_i, \hat{y}_i^{(k)}) + \Omega(f_k), \tag{6}$$

where $l(\cdot)$ is the loss function, y_i is the label of the i^{th} input data point, and $\Omega(f_k)$ (optional) is the regularization term.

1) XGBoost Algorithm: GBDT may be improved by adding an L^2 norm regularization term to avoid over-fitting. XGBoost [9] employs the second-order Taylor series to approximate its objective function and a sparsity-aware algorithm to deal with the sparse data. A cache-aware block structure is used to generate the XGBoost model on parallel and distributed computing and to increase training speed.

The regularization function is:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda ||\omega||^2, \tag{7}$$

where γ and λ are the regularization coefficients, T is the number of leaves in the tree, and ω are the leaf weights.

The second-order Taylor series is used to approximate (6):

$$\mathcal{L}^{(k)} \simeq \sum_{i=1}^{N} \Big[l(y_i, \hat{y}_i^{(k-1)}) + g_i f_k(\boldsymbol{x}_i) + \frac{1}{2} h_i f_k^2(\boldsymbol{x}_i) \Big] + \Omega(f_k), \tag{8}$$

where $g_i = \frac{\partial l(y_i, \hat{y}_i^{(k-1)})}{\partial \hat{y}_i^{(k-1)}}$ and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(k-1)})}{\partial (\hat{y}_i^{(k-1)})^2}$ are known and $l(y_i, \hat{y}_i^{(k-1)})$ is a constant.

For a known tree structure $q(\boldsymbol{X})$, I_t is a set containing the indices of data points in leaf t. Setting the derivative of (8) to zero gives the optimal weight ω_t^* for leaf t:

$$\omega_t^* = -\frac{\sum_{i \in I_t} g_i}{\sum_{i \in I_t} h_i + \lambda}. \tag{9}$$

The optimal solution of the objective function is:

$$\mathcal{L}^{*(k)} = -\frac{1}{2} \sum_{t=1}^{T} \frac{(\sum_{i \in I_t} g_i)^2}{\sum_{i \in I_t} h_i + \lambda} + \gamma T. \tag{10}$$

This optimal value is used to evaluate the quality of a tree structure $q(\boldsymbol{X})$. The tree structure with the lowest optimal value is selected for each iteration.

2) LightGBM Algorithm: Gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) techniques are employed to significantly accelerate the training speed. LightGBM [13] achieves performance comparable to XGBoost albeit with lower memory usage.

LightGBM employs the histogram-based algorithm that accelerates the process to locate the best splitting point for each feature. Using the training data points, mutually exclusive features are bundled to create feature histograms. GOSS involves sorting the training data points in a descending order based on the absolute value of their gradients. Top N_t data points (subset $\boldsymbol{A}$) with the largest gradients are selected and random sampling of the remaining input data points is performed to create a subset $\boldsymbol{B}$. The dimensions of $\boldsymbol{A}$ and $\boldsymbol{B}$ depend on predefined sampling ratios a and b, respectively. When training a GBDT model with a given dataset, gradients are calculated in each iteration.

In a decision tree, nodes are split based on features with the largest information gain that depends on the variance gain $\tilde{V}_j(d)$ for feature j computed after splitting as [13]:

$$\begin{aligned} \tilde{V}_j(d) = &\frac{1}{N \times N_l^j(d)} \Big(\sum_{\boldsymbol{x}_i \in \boldsymbol{A}_l} g_i + \frac{1-a}{b} \sum_{\boldsymbol{x}_i \in \boldsymbol{B}_l} g_i \Big)^2 \\ &+ \frac{1}{N \times N_r^j(d)} \Big(\sum_{\boldsymbol{x}_i \in \boldsymbol{A}_r} g_i + \frac{1-a}{b} \sum_{\boldsymbol{x}_i \in \boldsymbol{B}_r} g_i \Big)^2, \end{aligned} \tag{11}$$

where d is the splitting point, N is the number of data points, N_l^j and N_r^j are number of input data points related to left and right child nodes, respectively, and g_i is the gradient for input data point $\boldsymbol{x}_i$. The sampling ratios a and b are used to calculate the normalization coefficient $(1-a)/b$. $\boldsymbol{A}_l$ ($\boldsymbol{B}_l$) and $\boldsymbol{A}_r$ ($\boldsymbol{B}_r$) are the subsets of $\boldsymbol{A}$ ($\boldsymbol{B}$) for the left and right child nodes, respectively.

LightGBM is based on the GOSS technique and utilizes leaf-wise growth approach to grow the decision trees instead of level-wise growth used in XGBoost. Level-wise growth splits the leaves of the same layer, which enables easy control of the model complexity. However, it introduces unnecessary overhead because the leaves with low variance gains are also split. Compared to level-wise tree growth, leaf-wise is a more efficient approach because it splits the leaf that has the maximum variance gain thus reducing additional loss after a number of splits. Furthermore, it may lead to deeper decision trees resulting in over-fitting. Hence, the hyper-parameter "max depth" is introduced to limit their depth.

The EFB technique combines dataset features in order to reduce the dimension of the input data and the complexity of building the histogram from $\mathcal{O}(n_d n_f)$ to $\mathcal{O}(n_d n_b)$, where n_d, n_f, and n_b are the number of data points, features, and bundles, respectively.

3) CatBoost Algorithm: The XGBoost algorithm only accepts numerical values and employs one-hot encoding to convert categorical features to numerical values while LightGBM converts these features to gradient statistics. Hence, CatBoost [19] is introduced to deal with categorical features. It employs the ordered boosting algorithm and offers an effective approach (ordered target statistic) when compared to XGBoost and LightGBM. Target statistic was used to convert categorical to numerical features by using the values that estimate the expected labels based on the categories while keeping the dimension of the dataset unchanged.

In the existing GBDT models, residuals are calculated in each iteration and are used as the target values in the next training iteration. This leads to bias increase and prediction shift in subsequent iterations and, thus, model over-fitting. Hence, ordered boosting was proposed to address the prediction shift when building the decision trees during the training process. It performs permutation and trains multiple decision trees in each iteration. Each residual is calculated based on the target and predicted values generated by the previous decision tree. Symmetric (oblivious) decision trees are used to avoid over-fitting and reduce the time required to grow the tree. CatBoost offers plain and ordered boosting modes with target statistics and ordered boosting, respectively. In each iteration, the two boosting modes have the same asymptotic complexity for calculating gradients

$\mathcal{O}(sN)$, updating decision trees $\mathcal{O}(sN)$, and computing ordered target statistic $\mathcal{O}(N_{TS}N)$, where s, N, and N_{TS} are the number of permutations, data points, and features using target statistics, respectively.

III. Description of Datasets

The CIC datasets [1] were collected using testbeds that consist of victim and attacker networks. Regular (benign) traffic was generated by implementing B-profiles that replicated the behavior of regular users. M-profiles were used to generate malicious traffic based on common techniques that execute various attacks: botnet, brute force File Transfer Protocol (FTP) and Secure Shell Protocol (SSH), DoS, DDoS, heartbleed, infiltration, and web attack. Dataset features were extracted from collected TCP and UDP network flows with a network traffic flow analyzer. Each dataset has over 80 features including destination IP and port, protocol type, flow duration, and maximum/minimum packet size. Attacks considered in our experiments are listed in Table I.

TABLE I
Application-Layer DoS and TCP/UDP DDoS Attacks

Dataset	Attack	No. of data points
	GoldenEye	10,293
CICIDS2017	Hulk	230,124
July 05, 2017	SlowHTTPTest	5,499
	Slowloris	5,796
CSE-CIC-IDS2018	GoldenEye	41,508
February 15, 2018	Slowloris	10,990
	Domain Name System	5,071,011
CICDDoS2019	Lightweight Directory	2,179,930
December 01, 2018	Access Protocol	
	Network Time Protocol	1,202,642

The CICIDS2017 dataset was generated using a testbed where the victim network consisted of three servers, one firewall, two switches, and ten terminals while the attacker network had one router, one switch, and four terminals. Network traffic flows were collected over five business days and 84 features were extracted. The testbed used to generate the CSE-CIC-IDS2018 dataset consisted of an attacker-network with 50 terminals and a victim-network with five subnets that included 420 terminals and 30 servers. Network traffic flows were collected over ten business days and 83 features were extracted. The CICDDoS2019 dataset was collected on November 03, 2018 and December 01, 2018. The testbed designed to generate this dataset consisted of a victim network with one server, one firewall, two switches, and four terminals while the attacker network was an external (third-party) enterprise. Captured network traffic flows were analyzed to extract 87 features.

Spatial separations of features selected from the CICIDS2017, CSE-CIC-IDS2018, and CICDDoS2019 datasets are visualized in scattered plots shown in Fig. 1. The datasets show visible separation between regular and anomalous classes for various combinations of features. In the case of the CICDDoS2019 dataset, better spatial separation of the two classes captures distinct data patterns leading to better classification performance.

IV. Experiments and Performance Evaluation

Five BLS algorithms with and without incremental learning and three GBDT algorithms are implemented for two-way classification of regular and anomalous data points. The CIC datasets are used to compare performance of algorithms based on training time, accuracy, F-Score, precision, sensitivity, and confusion matrix: true positive (TP), false positive (FP), true negative (TN), and false negative (FN).

We use subsets of the CIC datasets to create training and test datasets with 78, 78, and 79 most relevant features for the CICIDS2017, CSE-CIC-IDS2018, and CICDDoS2019 datasets, respectively. (The CICDDoS2019 dataset includes an additional feature that indicates the traffic direction.) Training and test datasets are partitioned based on the 60 % and 40 % content of the anomalous data points, respectively.

The experiments (cross-validation and testing) are conducted using a supercomputer managed by Compute Canada. The Cedar [3] cluster consists of 94,528 CPU cores. We requested 64 GB memory and an Intel E5-2683 v4 Broadwell (2.1 GHz) processor with 8 cores and used Python 3.6.10 and its libraries [2]: *NumPy*, *pandas*, *scikit-learn*, *XGBoost*, *LightGBM*, and *CatBoost*.

The partitioning process for training and validation datasets for the 10-fold cross-validation based on the time series split is illustrated in Fig. 2. In each fold, 25,000 data points are used as the validation dataset. In the first step (Fold 1), 25,000 data points are used for training. The training dataset in each subsequent fold is the concatenation of the previous training and validation datasets.

Training hyper-parameters of the BLS and GBDT models that generate the best performance results are listed in Table II and Table III, respectively. In our experiments, $\phi(\cdot)$ (1) and $\xi(\cdot)$ (2) are $linear$ and $tanh$ mappings, respectively. Additional hyper-parameters for incremental BLS are: *incremental learning steps* = 2, *enhancement nodes/step* = 20 (CICIDS2017, CSE-CIC-IDS2018) and 10 (CICDDoS2019), and *data points/step* = 55,680 (CICIDS2017), 49,320 (CSE-CIC-IDS2018), and 382,929 (CICDDoS2019). Additional hyper-parameters for GBDT algorithms are: *maximum depth in a tree* = 6 (XGBoost, CatBoost), *maximum number of leaves* = 31 (LightGBM, CatBoost), and *loss function* = log-loss. We implement *gbtree* (XGBoost), *gbdt* (LightGBM), and *Plain* (CatBoost) boosting modes.

TABLE II
BLS and Incremental BLS Hyper-Parameters Leading to the Best Performance: CIC Datasets

Model	Dataset	Mapped features	Groups of mapped features	Enhancement nodes
BLS				
RBF-BLS	CICIDS2017	20	30	40
CFBLS	CSE-CIC-IDS2018	20	10	80
BLS	CICDDoS2019	15	5	20
Incremental BLS				
CFBLS	CICIDS2017	10	20	40
BLS	CSE-CIC-IDS2018	15	30	20
CFEBLS	CICDDoS2019	20	5	10

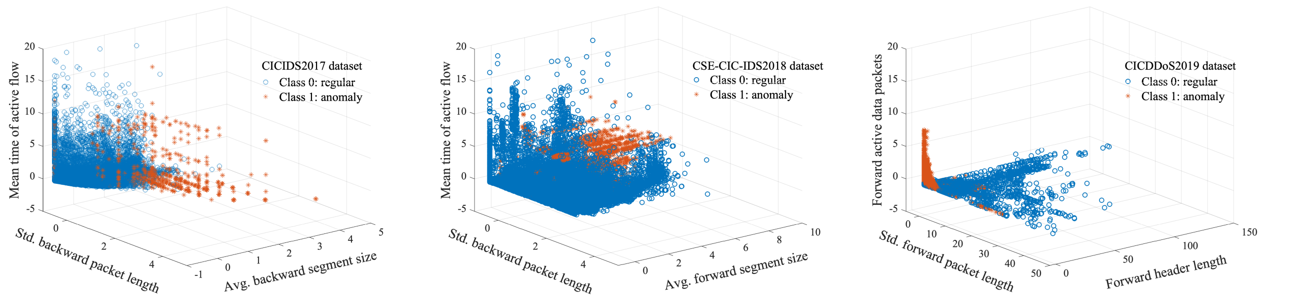

Fig. 1. Scattered plots of the CICIDS2017 (left), CSE-CIC-IDS2018 (middle), and CICDDoS2019 (right). Illustrated are spatial separations of regular (class 0) and anomalous (class 1) data points.

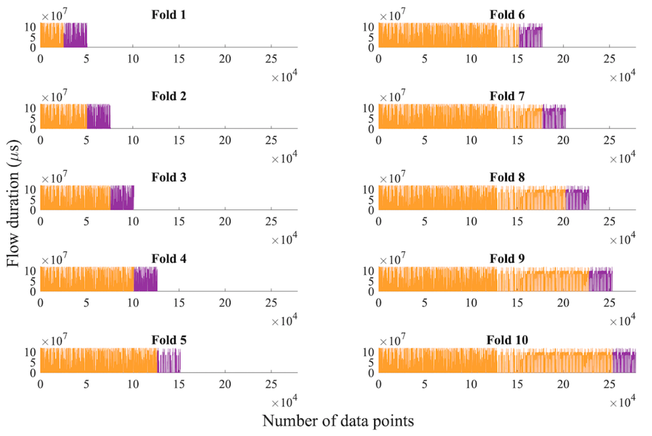

Fig. 2. Time series split for the 10-fold cross-validation of the CICIDS2017 training dataset. Illustrated are the generations of training (orange) and validation (purple) datasets.

TABLE III

XGBoost, LightGBM, and CatBoost Hyper-Parameters Leading to the Best Performance: CIC Datasets

Model	**Dataset**	**Number of estimators**	**Learning rate**
XGBoost	CICIDS2017	100	0.01
	CSE-CIC-IDS2018	100	0.01
	CICDDoS2019	20	0.01
LightGBM	CICIDS2017	200	0.10
	CSE-CIC-IDS2018	150	0.02
	CICDDoS2019	20	0.05
CatBoost	CICIDS2017	150	0.10
	CSE-CIC-IDS2018	150	0.01
	CICDDoS2019	20	0.01

Classification results for the BLS and GBDT models are shown in Table IV and Table V, respectively.

A. *Effect of Hyper-Parameters on Algorithm Performance*

Performance of BLS models depends on the number of mapped features, groups of mapped features, and enhancement nodes. As the value of these hyper-parameters increases, the models require additional memory and longer training time. Selecting more than 30 mapped features, 30 groups of mapped features, and 100 enhancement nodes requires over 64 GB of the supercomputer memory. Calculation of ridge regression further exacerbates the memory requirements. Hence, the range of hyper-parameters has been selected based on memory consumption. Performance (accuracy and F-Score) of BLS models does not monotonically depend on the choice of hyper-parameters. Thus, we rely on the 10-fold cross-validation to select hyper-parameters leading to the highest average accuracy of the validated models. The BLS models trained with the CICDDoS2019 dataset consist of fewer number of mapped features (5) and enhancement nodes (up to 20) compared to models for the CICIDS2017 and CSE-CIC-IDS2018 datasets, without affecting their performance.

The number of estimators and the learning rate are the main hyper-parameters for the GBDT algorithms. The log-loss function was monitored for various hyper-parameters in order to observe its behavior. A range of hyper-parameters was selected around small values of log-loss function to proceed with cross-validation. For each combination of parameters, the 10-fold cross-validation is used to select hyper-parameters leading to the highest average accuracy of the models. Generated GBDT models using the CICDDoS2019 dataset have fewer number of estimators compared to models used with the CICIDS2017 and CSE-CIC-IDS2018 datasets. The largest number of estimators is employed to generate the LightGBM model using the CICIDS2017 dataset.

LightGBM models offer the shortest training time for all considered datasets as shown in Tables IV and V. Their training time is approximately 20 times shorter than the BLS, XGBoost, and CatBoost models that have comparable training times. The GBDT models outperform original and incremental BLS models using the CICIDS2017 and CSE-CIC-IDS2018 datasets. The best accuracy and F-Score for XGBoost and CatBoost models are obtained for the CICIDS2017 and CSE-CIC-IDS2018 datasets, respectively. The XGBoost and CatBoost models generate the lowest number of FNs for the CICIDS2017 and CSE-CIC-IDS2018 datasets, respectively. The BLS and GBDT models using the CICDDoS2019 dataset have similar (two decimal points) and very high accuracy, F-Score, precision, and sensitivity because the regular class has few data points. Hence, the confusion matrix is used for comparison. The XGBoost and LightGBM models generate the lowest number of misclassified anomalous (FN) and regular (FP) data points when using the CICDDoS2019 dataset.

TABLE IV

THE BEST PERFORMANCE OF BLS AND INCREMENTAL BLS MODELS: CIC DATASETS

Model BLS	Dataset	Training time (s)	Accuracy (%)	F-Score (%)	Precision (%)	Sensitivity (%)	TP	FP	TN	FN
RBF-BLS	CICIDS2017	37.72	96.63	96.87	97.57	96.18	96,832	2,416	82,511	3,841
CFBLS	CSE-CIC-IDS2018	17.04	97.46	81.46	98.26	69.56	14,597	258	240,057	6,388
BLS	CICDDoS2019	46.64	99.98	99.99	99.99	99.99	2,541,553	204	954	220
Incremental BLS										
CFBLS	CICIDS2017	17.60	95.12	95.44	96.73	94.19	94,827	3,206	81,721	5,846
BLS	CSE-CIC-IDS2018	38.09	97.47	81.35	99.51	68.80	14,437	71	240,244	6,548
CFEBLS	CICDDoS2019	79.01	99.97	99.99	99.97	99.99	2,541,764	646	512	9

TABLE V

THE BEST PERFORMANCE OF XGBOOST, LIGHTGBM, AND CATBOOST MODELS: CIC DATASETS

Model	Dataset	Training time (s)	Accuracy (%)	F-Score (%)	Precision (%)	Sensitivity (%)	TP	FP	TN	FN
XGBoost	CICIDS2017	24.49	98.62	98.72	99.43	98.02	98,684	568	84,359	1,989
	CSE-CIC-IDS2018	14.43	99.90	99.39	99.99	98.79	20,731	1	240,314	254
	CICDDoS2019	62.99	99.99	99.99	99.99	99.99	2,541,767	7	1,151	6
LightGBM	CICIDS2017	3.35	97.93	98.06	99.94	96.25	96,896	60	84,867	3,777
	CSE-CIC-IDS2018	1.73	98.73	91.44	99.99	84.23	17,675	1	240,314	3,310
	CICDDoS2019	8.12	99.99	99.99	99.99	99.99	2,541,767	8	1,150	6
CatBoost	CICIDS2017	20.27	98.01	98.13	99.91	96.41	97,056	83	84,844	3,617
	CSE-CIC-IDS2018	19.03	99.95	99.72	99.97	99.46	20,872	6	240,309	113
	CICDDoS2019	17.38	99.99	99.99	99.99	99.99	2,541,762	19	1,139	11

V. CONCLUSION

We compared performance of BLS and GBDT supervised machine learning algorithms using three CIC datasets. Performance metrics such as training time, accuracy, F-Score, precision, sensitivity, and confusion matrix were used. Training time for the BLS models depended on the number of mapped features, groups of mapped features, and enhancement nodes while time for GBDT models was affected by the number of estimators, learning rate as well as the maximum depth and number of leaves in the decision trees. XGBoost, LightGBM, and CatBoost offered better accuracy and F-Score than BLS models. The shortest training time was required for LightGBM models. In case of similar results (accuracy, F-Score, precision, and sensitivity), we used the confusion matrix to further compare the classification performance. The experiments illustrated advantages of GBDT algorithms when detecting DoS and DDoS attacks.

REFERENCES

[1] Canadian Institute for Cybersecurity Datasets [Online]. Available: https://www.unb.ca/cic/datasets/index.html. Accessed: August 31, 2021.

[2] Python Package Index [Online]. Available: https://pypi.org. Accessed: August 31, 2021.

[3] Cedar [Online]. Available: https://docs.computecanada.ca/wiki/Cedar. Accessed: August 31, 2021.

[4] A. Bhardwaj, V. Mangat, R. Vig, S. Halder, and M. Conti, "Distributed denial of service attacks in cloud: state-of-the-art of scientific and commercial solutions," *Computer Science Review*, vol. 39, no. 100332, Feb. 2021.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag, 2006.

[6] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "Denial-of-service attack-detection techniques," *IEEE Internet. Comput.*, vol. 10, no. 1, pp. 82–89, Jan.–Feb. 2006.

[7] C. L. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[8] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–14, Sept. 2018.

[9] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 785–794.

[10] J. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–01232, Apr. 2001.

[11] A. L. Gonzalez Rios, Z. Li, K. Bekshentayeva, and Lj. Trajković, "Detection of denial of service attacks in communication networks," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seville, Spain, Oct. 2020.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: The MIT Press, 2016.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: a highly efficient gradient boosting decision tree," in *Proc. Int. Conf. Neural Inform. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, 3146–3154.

[14] Z. Li, A. L. Gonzalez Rios, and Lj. Trajković, "Machine learning for detecting anomalies and intrusions in communication networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2254-2264, July 2021.

[15] Z. Liu and C. L. P. Chen, "Broad learning system: structural extensions on single-layer and multi-layer neural networks," in *Proc. Int. Conf. Secur., Pattern Anal., Cybern.*, Shenzhen, China, Dec. 2017, pp. 136–141.

[16] J. P. A. Maranhão, J. P. C. L. da Costa, E. P. de Freitas, E. Javidi, and R. T. de Sousa, Jr., "Noise-robust multilayer perceptron architecture for distributed denial of service attack detection," *IEEE Commun. Lett.*, vol. 25, no. 2, pp. 402–406, Feb. 2021.

[17] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Commun. Surveys Tut.*, vol. 21, no. 1, pp. 686–728, First quarter 2019.

[18] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: The MIT Press, 2012.

[19] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," in *Proc. Int. Conf. Neural Inform. Process. Syst.*, Montreal, Québec, Canada, Dec. 2018, 6639–6649.

[20] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *J. Softw. Netw.*, vol. 2017, no. 1, pp. 177–200, July 2017.