Machine Learning for Detecting Anomalies and Intrusions in Communication Networks

Zhida Li, *Graduate Student Member, IEEE,* Ana Laura Gonzalez Rios, *Graduate Student Member, IEEE,* and Ljiljana Trajković, *Fellow, IEEE*

***Abstract*—Cyber attacks are becoming more sophisticated and, hence, more difficult to detect. Using efficient and effective machine learning techniques to detect network anomalies and intrusions is an important aspect of cyber security. A variety of machine learning models have been employed to help detect malicious intentions of network users. In this paper, we evaluate performance of recurrent neural networks (Long Short-Term Memory and Gated Recurrent Unit) and Broad Learning System with its extensions to classify known network intrusions. We propose two BLS-based algorithms with and without incremental learning. The algorithms may be used to develop generalized models by using various subsets of input data and expanding the network structure. The models are trained and tested using Border Gateway Protocol routing records as well as network connection records from the NSL-KDD and Canadian Institute of Cybersecurity datasets. Performance of the models is evaluated based on selected features, accuracy, F-Score, and training time.**

***Index Terms*—Intrusion detection, network anomalies, feature selection, machine learning, recurrent neural networks, broad learning system.**

I. Introduction

THE Internet has been highly susceptible to failures and attacks that greatly affect its performance. Over the past years, frequent cases of complex and challenging threats have been encountered. Hence, various machine learning algorithms have been considered to enhance cyber security [1]–[3]. Machine learning algorithms [4] have been used to address a variety of engineering and scientific problems. They classify data using a feature matrix where its rows and columns correspond to data points and feature values, respectively. By providing a sufficient number of relevant features, machine learning approaches help build generalized classification models and improve their performance. Supervised machine learning models used to classify network anomalies and intrusions include logistic regression [4], naïve Bayes [5], support vector machine (SVM) [6], decision tree [7], boosting algorithms [8], [9], deep learning networks [10], [11], and the broad learning system (BLS) [12], [13]. In this study, we use training time as one of the measures affecting the generation of a model. Note that a model's testing time, essential for detecting impending anomalies, is usually rather short. Selecting algorithms that have a short training time is important if they are to be implemented in network intrusion detection systems in order to adequately prevent the onset of malicious attacks. It will enable system administrators to effectively and timely remove affected network elements.

Feature selection algorithms are used to reduce the dimensionality of the dataset matrix by selecting features based on their importance. They enhance performance of machine learning algorithms and classification results. Algorithms such as Fisher, minimum Redundancy Maximum Relevance, mutual information base, and decision tree were used to select relevant features and to achieve better classification performance. In our prior studies [14]–[18], machine learning models were generated using a predefined set of selected features.

Light Gradient Boosting Machine (LightGBM) [9], an optimized gradient boosting decision tree (GBDT) algorithm, utilizes gradient-based one-side sampling and exclusive feature bundling techniques to accelerate the training speed for large datasets. Compared to conventional GBDT models, the LightGBM model achieved comparable performance with shorter training time for ranking and classification using various datasets.

Deep neural networks such as recurrent neural networks (RNNs), including bidirectional RNNs [19], [20], long short-term memory (LSTM) [21], [22], and gated recurrent unit (GRU) [23] are trained to identify important features in the input data by adjusting weights in each iteration. Their significant advantage over logistic regression, naïve Bayes, SVM, and decision tree algorithms is using back-propagation to calculate gradients and update the weights. Furthermore, deep neural networks may achieve desired classification results by adjusting the number of hidden nodes and layers, activation functions, and optimization algorithms. They employ linear or nonlinear activation functions such as rectified linear unit (ReLU), logistic sigmoid, or hyperbolic tangent (*tanh*). The numbers of hidden nodes and layers are chosen depending on the size of the dataset. An important advantage of RNNs is their ability to use contextual information between input and output sequences. The bidirectional recurrent neural networks (Bi-RNNs) [19], [20] were proposed to enhance handwriting and speech recognition. Bi-RNNs utilize both forward and backward information for prediction: an RNN layer calculates the output by using the sequential input data stream starting from its beginning while another RNN layer uses the sequential input backward starting from its end. The outputs of the two RNN layers are then combined. The LSTM neural network was proposed to address certain deficiencies in RNNs. The GRU structure is a special case of LSTM with a simpler structure. RNNs, convolutional neural networks, deep

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

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

belief networks, and autoencoders offer promising results for anomaly detection [24].

BLS [12] and its extensions [13], [25]–[29] are alternatives to deep learning networks. They achieve comparable classification accuracy and require a considerably shorter time for training than the conventional deep learning networks because of a small number of hidden layers. They also use pseudo-inverse or ridge regression approaches rather than back-propagation during the training process. BLS offers desired classification when used for function approximation, time series forecast, and image recognition. Generating a set of mapped features is an important step in BLS and its extensions. Various modifications for creating the set of mapped features have been proposed to improve the generalization of the BLS-based models such as CFBLS, CCFBLS [13], and multi-kernel BLS [28].

Our experimental results indicated that the best BLS models were not often derived by including all features in analyzed datasets. Using a subset of relevant features may enhance the model performance [8], [14], [15], [17], [18], [30], [31]. Reported BLS models that achieved the best performance were trained using a single subset of features extracted from the input data. Existing BLS-based algorithms include a single set of groups of mapped features where each group has a constant number of mapped features. We introduce two new BLS-based algorithms: variable features BLS algorithms without (VFBLS) and with cascades (VCFBLS), which are implemented with and without incremental learning. VFBLS and VCFBLS consist of a variable number of mapped features and groups of mapped features as well as an explicit feature selection algorithm to create subsets of input data thus making the model more generalized compared to BLS algorithms. Models generated using the proposed algorithms with integrated feature selection enable utilizing a variable number of selected features. Each mapped feature in VFBLS is created based on input data while in case of VCFBLS, consequent mapped features are generated based on a previous mapped feature. Training time of the proposed models may be adjusted by allocating a smaller number of mapped features and groups of mapped features to the input data subsets having a larger number of selected features.

Machine learning algorithms have been used to successfully classify network anomalies and intrusions [32]. These algorithms have been evaluated for robustness, high accuracy, and training time when classifying various datasets collected from communication networks. Reliable testing and validation of anomaly and intrusion detection algorithms depend on the quality of datasets such as traffic collected from deployed networks or experimental testbeds. We used the Border Gateway Protocol (BGP) Réseaux IP Européens (RIPE) [33], NSL-KDD [34], CICIDS2017 [35], and CSE-CIC-IDS2018 [36] datasets. CICIDS2017 and CSE-CIC-IDS2018 datasets were collected by the Canadian Institute for Cybersecurity (CIC). (We have not considered in this paper the BGP Route Views datasets [37] because they did not contain a complete record of anomalous events.)

Various intrusion detection systems (IDSs) [2], [38]–[40] are used to identify and classify malicious activities such as anomalies and intrusions in communication networks. They may be host-based or network-based. Host-based systems protect the host (endpoint) by monitoring operating system files and processes. Network-based systems (NIDSs) monitor incoming network traffic (Internet Protocol addresses, service ports, and protocols) by analyzing flows of packets or by inspecting packet headers. Their role is to enhance security by identifying suspicious events in the observed network traffic. Detecting malicious network intrusions may be signature-based or anomaly-based. Signature-based techniques [32] rely on known events that follow certain rules and patterns while anomaly-based intrusion detection techniques [1], [41] rely on detecting deviations from an expected behavior.

This paper is organized as follows: The RNNs (LSTM and GRU), BLS, and proposed BLS-based algorithms are presented in Section II. BGP RIPE, NSL-KDD, CICIDS2017, and CSE-CIC-IDS2018 datasets are described in Section III. The experimental procedure and performance evaluation are given in Section IV. We conclude with Section V.

II. Detecting Network Intrusions

Machine learning has changed approaches to counter cyber risks. Proposed approaches rely on supervised and unsupervised techniques to identify and classify anomalies in network traffic. Performance of proposed methods, evaluated based on accuracy and F-Score, depends on selected features and their combinations. The main disadvantages of conventional machine learning techniques are long training time and computational complexity. Hence, boosting algorithms (LightGBM) [42], deep learning networks (LSTM and GRU) [15], [17], [43] and the BLS were employed to detect network traffic anomalies [44]–[47].

A. Long Short-Term Memory

LSTM networks are RNNs that are capable of learning long-term dependencies by connecting time intervals to form a continuous memory [21], [22]. Traditional RNN networks perform poorly when they need to bridge segments of information with long time gaps. Hence, LSTM networks were introduced to overcome long-term dependency and vanishing gradient problems.

The LSTM cell is composed of: (a) *forget gate* f_t, (b) *input gate* i_t, and (c) *output gate* o_t. The *forget gate* discards irrelevant memories based on the cell state, the *input gate* controls the information that will be updated in the LSTM cell, and the *output gate* functions as a filter that controls the output. The logistic sigmoid σ and $tanh$ are used as cell functions. The output of the LSTM cell is connected to the output layer and the next cell.

The outputs of the *forget gate* f_t, *input gate* i_t, and *output gate* o_t at time t are [48]:

$$\begin{aligned} f_t &= \sigma(W_{if}x_t + b_{if} + U_{hf}h_{t-1} + b_{hf}) \\ i_t &= \sigma(W_{ii}x_t + b_{ii} + U_{hi}h_{t-1} + b_{hi}) \\ o_t &= \sigma(W_{io}x_t + b_{io} + U_{ho}h_{t-1} + b_{ho}), \end{aligned} \tag{1}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, x_t is the current input, h_{t-1} is the previous output, W_{if}, U_{hf}, W_{ii}, U_{hi}, W_{io},

and U_{ho} are weight matrices, and b_{if}, b_{hf}, b_{ii}, b_{hi}, b_{io}, and b_{ho} are bias vectors. The information is stored in the cell state depending on the output i_t of the *input gate*. The sigmoid function is used to update the cell state c_t calculated as:

$$c_t = f_t * c_{t-1} + i_t * tanh(W_{ic}x_t + b_{ic} + U_{hc}h_{t-1} + b_{hc}), \quad (2)$$

where $*$ denotes element-wise multiplications and the $tanh$ function is used to calculate the input to the next cell state. The output of the LSTM cell is:

$$h_t = o_t * tanh(c_t). \quad (3)$$

B. Gated Recurrent Unit

The GRU cell is derived from LSTM and has a simpler structure. To make predictions, it employs gated mechanisms to control input and memory at the current timestep. While an LSTM cell consists of three gates, a GRU cell contains only *reset* r_t and *update* z_t gates [23]. The *reset gate* determines the combination of new input information and previous memory content while the *update gate* defines the content stored at the current timestep.

The outputs of the *reset gate* r_t and the *update gate* z_t at time *t* are [48]:

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + U_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + U_{hz}h_{t-1} + b_{hz}), \end{aligned} \quad (4)$$

where $\sigma(\cdot)$ is the logistic sigmoid function, x_t is the input, h_{t-1} is the previous cell output, W_{ir}, U_{hr}, W_{iz}, and U_{hz} are the weight matrices, and b_{ir}, b_{hr}, b_{iz}, and b_{hz} are the bias vectors. The output of the GRU cell is:

$$h_t = (1 - z_t) * n_t + z_t * h_{t-1}, \quad (5)$$

where n_t is:

$$n_t = tanh(W_{in}x_t + b_{in} + r_t * (U_{hn}h_{t-1} + b_{hn})), \quad (6)$$

W_{in} and U_{hn} are the weight matrices, and b_{in} and b_{hn} are the bias vectors.

C. Broad Learning System

Deep learning neural networks may require long training time because of their high computational complexity and a large number of hidden layers. In contrast, boosting algorithms such as LightGBM [9] and BLS [29] offer comparable performance with shorter training time. BLS is based on a single layer feedforward neural network and pseudo-inverse or ridge regression to calculate outputs. Several BLS extensions exploit the algorithm's flexible structure to include: incremental learning [12], radial basis function network (RBF-BLS) [25] as well as cascades of mapped features (CFBLS), enhancement nodes (CEBLS), and both mapped features and enhancement nodes (CFEBLS) [13].

BLS improves the random vector functional-link neural network [49] by mapping the input data $\boldsymbol{X}$ to a set of groups of mapped features $\boldsymbol{Z}^n \triangleq [\boldsymbol{Z}_1, ..., \boldsymbol{Z}_n]$ that generates enhancement nodes $\boldsymbol{H}^m \triangleq [\boldsymbol{H}_1, ..., \boldsymbol{H}_m]$ using random weights. Groups of mapped features and enhancement nodes are defined as:

$$\boldsymbol{Z}_i = \phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}), i = 1, 2, ..., n \quad (7)$$

$$\boldsymbol{H}_j = \xi(\boldsymbol{Z}_x^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}), j = 1, 2, ..., m, \quad (8)$$

where ϕ $(linear)$ and ξ $(tanh)$ are the feature and enhancement mappings, respectively, $\boldsymbol{W}_{e_i}$ and $\boldsymbol{W}_{h_j}$ are weights, and $\boldsymbol{\beta}_{e_i}$ and $\boldsymbol{\beta}_{h_j}$ are bias parameters. A state matrix $\boldsymbol{A}_n^m$ is constructed by concatenating matrices $\boldsymbol{Z}^n$ and $\boldsymbol{H}^m$ associated with n groups of mapped features and m groups of enhancement nodes, respectively. The Moore-Penrose pseudo-inverse or ridge regression of matrix $\boldsymbol{A}_n^m$ is computed to calculate the weights $\boldsymbol{W}_n^m$ for the given labels $\boldsymbol{Y}$. During testing, data labels are predicted using the calculated weights, mapped features, and enhancement nodes.

The BLS structure may be dynamically expanded by using incremental learning to include additional input data $\boldsymbol{X}_a$, mapped features $\boldsymbol{Z}_{n+1}$, and enhancement nodes $\boldsymbol{H}_{m+1}$. It requires a shorter training time because the weights are updated using only the incremental input data instead of retraining the entire model. RBF-BLS extension employs the Gaussian rather than *tanh* function as the enhancement mapping ξ. The structure of BLS with cascades is defined by the connections within and between the mapped features and enhancement nodes. In the case of CFBLS, the first group of mapped features is based on input data and weights (7) while subsequent groups (k) of mapped features are created by using the previous group $(k-1)$. The groups of mapped features are formulated as:

$$\begin{aligned} \boldsymbol{Z}_k &= \phi(\boldsymbol{Z}_{k-1}\boldsymbol{W}_{e_k} + \boldsymbol{\beta}_{e_k}) \\ &\triangleq \phi^k(\boldsymbol{X}; \{\boldsymbol{W}_{e_i}, \boldsymbol{\beta}_{e_i}\}_{i=1}^k), \text{for } k = 1, ..., n. \end{aligned} \quad (9)$$

The cascades of these groups $\boldsymbol{Z}^n \triangleq [\boldsymbol{Z}_1, ..., \boldsymbol{Z}_n]$ are used to generate the enhancement nodes $\{\boldsymbol{H}_j\}_{j=1}^m$. The first CEBLS enhancement node is generated from mapped features while subsequent nodes are generated from previous nodes creating a cascade:

$$\begin{aligned} \boldsymbol{H}_u &= \xi(\boldsymbol{H}_{u-1}\boldsymbol{W}_{e_u} + \boldsymbol{\beta}_{e_u}) \\ &\triangleq \xi^u(\boldsymbol{Z}^n; \{\boldsymbol{W}_{h_i}, \boldsymbol{\beta}_{h_i}\}_{i=1}^u), \text{for } u = 1, ..., m, \end{aligned} \quad (10)$$

where $\boldsymbol{W}_{h_i}$ and $\boldsymbol{\beta}_{h_i}$ are randomly generated. The CFEBLS architecture is a combination of the two cascading approaches.

D. Variable Features Broad Learning System

Mapping the input data to sets of mapped features is an essential step of BLS algorithms. We propose a variable features system (VFBLS) and system with cascades (VCFBLS) shown in Fig. 1 that consist of a variable number of mapped features and groups of mapped features. They employ feature selection algorithms that enable models to be trained based on a variable number of features extracted from the input data. The two algorithms also offer variants with incremental learning.

The VFBLS and VCFBLS algorithms expand the BLS network by using both original and subsets of input data as well as sets of groups of mapped features. They enable developing more generalized models and, hence, prevent overfitting and enhance the performance. Generating the best

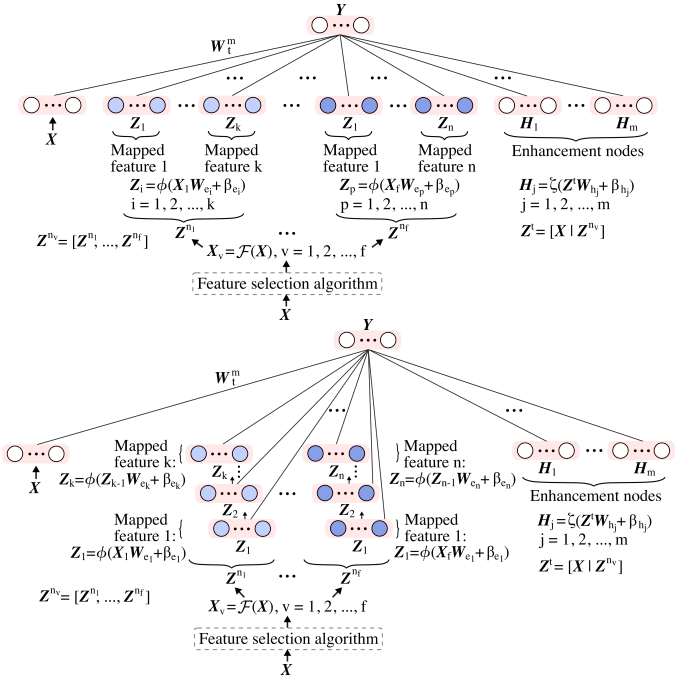


Fig. 1. Modules of the VFBLS and VCFBLS algorithms with input data $\boldsymbol{X}$, sets of groups of mapped features with and without cascades ($\boldsymbol{Z}^{n_1}$, ..., $\boldsymbol{Z}^{n_f}$), and enhancement nodes ($\boldsymbol{H}_1$, ..., $\boldsymbol{H}_m$).

BLS and incremental BLS models is rather time-consuming because they rely on multiple two-stage experiments: selecting features and generating models. In contrast, VFBLS and VCFBLS models are developed using a single experiment with integrated stages. A variable number of mapped features is used to reduce training time because the proposed algorithms introduce additional complexity by using the entire input dataset and by incorporating a feature selection algorithm and additional sets of mapped features. In case of incremental learning, features are selected in each step and ranked based on their importance. After the last step, all selected features are multiplied with weights proportional to the size of the dataset used in each step. They are then ranked and used for testing.

Data $\boldsymbol{X}$ and features that are selected based on a feature selection algorithm are used as input:

$$\boldsymbol{X}_v = \mathcal{F}(\boldsymbol{X}), v = 1, 2, ..., f, \tag{11}$$

where $\boldsymbol{X}_v$ is a subset of $\boldsymbol{X}$ with a selected set of features. Note that selecting all features in the input data to generate mapped features is a special case where $\boldsymbol{X}_v = \boldsymbol{X}$. Sets of groups of mapped features are generated as:

$$\boldsymbol{Z}^{n_v} \triangleq [\boldsymbol{Z}^{n_1}, ..., \boldsymbol{Z}^{n_f}], \tag{12}$$

where $\boldsymbol{Z}^{n_v}$ contains n_v mapped features. In the case of VF-BLS groups of mapped features $\boldsymbol{Z}^{n_v}$, $\boldsymbol{X}_v$ and n_v correspond to $\boldsymbol{X}$ and n, respectively (7) while VCFBLS groups of mapped features $\boldsymbol{Z}^{n_v}$ are defined based on the previous group (9). Note that the first mapped feature $\boldsymbol{Z}_1$ in each set is created from $\boldsymbol{X}_1$, ..., $\boldsymbol{X}_f$. The number of mapped features and groups of mapped features may vary in sets $\boldsymbol{Z}^{n_1}$, ..., $\boldsymbol{Z}^{n_f}$. The number of mapped features in each group of a set is constant. We improve performance by including properties of the original data and concatenating input data $\boldsymbol{X}$ and sets of mapped features to create $\boldsymbol{Z}^t$ similar to the case of random vector functional-link network [49]:

$$\boldsymbol{Z}^t = [\boldsymbol{X}|\boldsymbol{Z}^{n_v}]. \tag{13}$$

The enhancement nodes are:

$$\boldsymbol{H}_j = \xi(\boldsymbol{Z}^t\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}), j = 1, 2, ..., m. \tag{14}$$

The state matrix $\boldsymbol{A}_t^m$ is the concatenation of $\boldsymbol{Z}^t$ and $\boldsymbol{H}^m$. The ridge regression algorithm is then employed to compute the weights $\boldsymbol{W}_t^m$ based on $\boldsymbol{A}_t^m$ and given labels $\boldsymbol{Y}$. The error function, minimized during the training process, is defined as [12]:

$$\boldsymbol{E}(\boldsymbol{W}_t^m) = ||\boldsymbol{A}_t^m\boldsymbol{W}_t^m - \boldsymbol{Y}||_2^2 + \lambda||\boldsymbol{W}_t^m||_2^2, \tag{15}$$

where λ is the sparse regularization coefficient. The term $\lambda||\boldsymbol{W}_t^m||_2^2$ is used to control over-fitting. The minimum of (15) can be found in closed-form by setting its derivative with respect to $\boldsymbol{W}_t^m$ to 0. The output weights are defined as [4]:

$$\boldsymbol{W}_t^m = (\lambda\boldsymbol{I} + (\boldsymbol{A}_t^m)^T\boldsymbol{A}_t^m)^{-1}(\boldsymbol{A}_t^m)^T\boldsymbol{Y}. \tag{16}$$

Pseudocode for the VFBLS and VCFBLS algorithms and their incremental versions are listed in Algorithm 1 and Algorithm 2, respectively.

A variety of feature selection algorithms may be employed. The extremely randomized trees (extra-trees) feature selection algorithm [50] is used in our experiments to rank features based on importance. The algorithm is an improved version of the decision tree and random forests used to select relevant features for generating subsets of the input data. It introduces additional randomness by randomly splitting nodes in order to avoid over-fitting. Features with higher importance are more relevant for a given dataset and better capture its properties. They may have better spatial separation and, thus, enhance the model's performance. The Gini importance is used to compute feature scores in a given dataset [51]:

$$Importance(\boldsymbol{X}_c) = \frac{1}{N_T}\sum_{T}\sum_{t\in T:v(s_t)=\boldsymbol{X}_c} p(t)\Delta i(s_t, t), \tag{17}$$

where $\boldsymbol{X}_c$ is the subset of $\boldsymbol{X}$ corresponding to one feature, N_T is the number of trees, t is the index of a node in a tree, s_t is the direction of the split, $v(s_t)$ is a randomly generated threshold, $p(t)$ is the weight, and $\Delta i(s_t, t)$ is the decrease of the node impurity equivalent to its importance.

E. Intrusion Detection Systems

An early hybrid intrusion detection system (IDS) was proposed to identify misuse and anomaly intrusions using random forests [52]. Intrusion detection systems [2], [53], [54] have been designed using machine learning and deep neural networks such as stacked non-symmetric deep auto-encoder [55] and recurrent neural networks [43]. Training time is often of concern when employing deep learning algorithms. Hence, a stacked non-symmetric deep auto-encoder (NDAE) that combines deep and shallow learning offered by the random forest classifier has been proposed and implemented using a graphics processing unit (GPU) [55]. Network (NIDS) [56]

Algorithm 1 VFBLS and VCFBLS algorithms: Pseudocode

procedure VFBLS AND VCFBLS (training dataset $\boldsymbol{X}$ with labels $\boldsymbol{Y}$)
  Initialize:
  Number of subsets v of input data
  Number f of features to be selected in each subset
  Subsets of input data $\boldsymbol{X}_v$, $v = (1, ..., f)$
  Sets of groups of mapped features $\boldsymbol{Z}^{n_v}$, $n_v = (n_1, ..., n_f)$
  Groups of mapped features in each set $\boldsymbol{Z}_i$, $i = (k, ..., p)$
  Number of mapped features in each group
  for each f in v **do**
    Calculate feature importance and create $\mathcal{F}(\boldsymbol{X})$ by ranking features using a feature selection algorithm
    Generate subset $\boldsymbol{X}_v = \mathcal{F}(\boldsymbol{X}), v = 1, 2, ..., f$
    for each set $\boldsymbol{Z}^{n_i}$ in $\boldsymbol{Z}^{n_v}$ **do**
      Initialize the set of groups of mapped features $\boldsymbol{Z}^{n_i}$
      for each group $\boldsymbol{Z}_i$ in set $\boldsymbol{Z}^{n_i}$ **do**
        switch Algorithm **do**
          case VFBLS
            Generate $\boldsymbol{Z}_i$ based on $\boldsymbol{X}_v$ and the number of mapped features
          case VCFBLS
            Generate $\boldsymbol{Z}_1$ based on $\boldsymbol{X}_v$ and the number of mapped features
            Generate subsequent groups $\boldsymbol{Z}_i$ based on $\boldsymbol{Z}_{i-1}$
        Insert $\boldsymbol{Z}_i$ into $\boldsymbol{Z}^{n_i}$
      end for
      Insert $\boldsymbol{Z}^{n_i}$ into $\boldsymbol{Z}^{n_v}$
    end for
  end for
  Construct matrix $\boldsymbol{Z}^t = [\boldsymbol{X}|\boldsymbol{Z}^{n_v}]$
  Generate enhancement nodes $\boldsymbol{H}^m = [\boldsymbol{H}_1, ..., \boldsymbol{H}_m]$ based on $\boldsymbol{Z}^t$
  Concatenate $\boldsymbol{Z}^t$ and $\boldsymbol{H}^m$ to create the state matrix $\boldsymbol{A}_t^m$
  Compute weights $\boldsymbol{W}_t^m$ based on $\boldsymbol{A}_t^m$ and labels $\boldsymbol{Y}$ using the ridge regression algorithm
end procedure

Algorithm 2 Incremental VFBLS and VCFBLS algorithms: Pseudocode

procedure INCREMENTAL VFBLS AND VCFBLS (training dataset $\boldsymbol{X}$ with labels $\boldsymbol{Y}$)
  Extract initial input subset $\boldsymbol{X}_0$ from dataset $\boldsymbol{X}$
  Extract initial labels subset $\boldsymbol{Y}_0$ from $\boldsymbol{Y}$
  Initialize:
  Number of incremental learning steps: l
  Number of data points per step: d
  Number of enhancement nodes per step: e
  Calculate feature weight vector $W_i = [w_0, w_1, ..., w_l]$: $w_0 = \boldsymbol{X}_0/\boldsymbol{X}$; $w_1, ..., w_l = (1 - w_0)/l$
  for each step in l **do**
    Generate $\boldsymbol{X}_a$ based on $\boldsymbol{X}$, $\boldsymbol{X}_0$, and d
    Generate $\boldsymbol{Y}_a$ based on $\boldsymbol{Y}$, $\boldsymbol{Y}_0$, and d
    Calculate feature importance and create $\mathcal{F}(\boldsymbol{X}_a)$ by ranking features using a feature selection algorithm
    Generate additional mapped features $\boldsymbol{Z}_{n+1}$ and additional enhancement nodes $\boldsymbol{H}_{m+1}$ using Algorithm 1
    Update $\boldsymbol{A}_t^m$
    Update weights $\boldsymbol{W}_t^m$
  end for
  Rank and select features to be used in testing based on: selected features and their importance in each step and the weight vector W_i
end procedure

and recurrent neural network (RNN-IDS) [43] intrusion detection systems were proposed and compared [57], [58] with various machine learning algorithms such as J48, naïve Bayes, naïve Bayes tree, random forests, random tree, multilayer perceptron, and support vector machine. A hybrid framework was proposed [59] to include binary classifier modules and an aggregation module generating certain and uncertain output classes. The data points from both classes are imported to a k-NN module that refines the class of uncertain data points. SwiftIDS employs LightGBM for generating models. Real-time data acquisition and data processing were used prior to the classification of anomalies using the proposed parallel intrusion detection mechanism [42].

III. Description of Datasets

We have classified network anomalies and intrusions [16], [17], [44], [47] using BGP RIPE [33], NSL-KDD [34], CI-CIDS2017 [35], and CSE-CIC-IDS2018 [36] datasets.

A. Border Gateway Protocol RIPE Datasets

BGP is prone to malicious attacks [60]. Three well-known BGP anomalies are: Slammer [61], [62], Nimda [63], [64], and Code Red I [65], which occurred in January 2003, September 2001, and July 2001, respectively.

BGP RIPE datasets containing anomalies and regular data may be extracted from BGP update messages collected during periods of Internet anomalies [33], [37]. We consider both anomalous (the days of the attack) and regular (two days prior and two days after the attack) data [16], [17], [45], [46]. The duration of anomalies and the number of data points in the BGP RIPE datasets are shown in Table I. We extract 37 numerical features [66] from BGP update messages [33] that originated from AS 513 (route collector rrc04) using a tool written in C# [67]. Slammer and Code Red I datasets contain 7,200 data points consisting of five days of anomalous and regular data while Nimda contains 8,609 data points. Hence, each data point represents one minute of routing records. Training and test datasets for Slammer, Nimda, and Code Red I are shown in Table II.

TABLE I
BGP RIPE DATASETS: INTERNET ANOMALIES

Dataset	Beginning of event GMT	End of event GMT	Duration (min)
Slammer	25.01.2003, 05:31	25.01.2003, 19:59	869
Nimda	18.09.2001, 13:19	19.09.2001, 10:59	1,301
Code Red I	19.07.2001, 13:20	19.07.2001, 23:19	600

TABLE II
BGP RIPE DATASETS: NUMBER OF DATA POINTS

Dataset	Regular (training)	Anomaly (training)	Regular (test)	Anomaly (test)
Slammer	3,210	530	3,121	339
Nimda	3,673	827	3,635	474
Code Red I	3,679	361	2,921	239

For each collected dataset (Slammer, Nimda, Code Red I), we experimented with a number of training and test data points that are selected from periods of anomalies [46]. We partitioned the datasets by selecting 80 %, 70 %, or 60 % of anomalous data for training and the remaining 20 %, 30 %, or 40 % for testing. (We kept the number of data points in the training and test datasets to be divisible by 20 by moving the remaining data points from the training to the test dataset.) Using 60 % of data for training and 40 % for testing generated the best performance results. Another approach was to use concatenations of two datasets for training while the third dataset is used for testing. For example, Slammer and Nimda were concatenated in the training phase and used to test Code Red I dataset. Experiments indicated that the two approaches did not greatly affect performance of the employed algorithms.

B. NSL-KDD Dataset

The NSL-KDD [34] dataset is an improved version of the KDD'99 intrusion dataset based on DARPA 1998 dataset that contains 9 weeks of collected traffic when various intrusions were introduced in a simulated US Air Force base network [68], [69]. Data were captured from an evaluation testbed and included large numbers of virtual hosts and user automata. The KDD'99 dataset [70]–[72] was used in various IDSs [68], [73]–[75]. NSL-KDD is a randomly selected subset of KDD'99 after redundant data were removed [76] and is a widely used benchmark for evaluating anomaly detection techniques. NSL-KDD dataset captures Transport Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP) traffic collected using the *tcpdump* utility. It contains four types of intrusion attacks: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probe.

The NSL-KDD dataset contains one training (KDDTrain^{+}) and two test datasets (KDDTest^{+} and KDDTest^{-21}). KDDTest^{-21} is a subset of the KDDTest^{+} dataset that includes records that could not be correctly classified by 21 models [76]. The number of data points is shown in Table III [34]. Each network connection is represented by 41 features: 38 numerical and 3 categorical ("protocol_type", "service", and "flag") features. Categorical features are converted to numerical features using the dummy coding method to generate 71 additional features.

TABLE III
NSL-KDD DATASET: NUMBER OF DATA POINTS

	Regular	DoS	U2R	R2L	Probe	Total
$KDDTrain^{+}$	67,343	45,927	52	995	11,656	125,973
$KDDTest^{+}$	9,711	7,458	200	2,754	2,421	22,544
$KDDTest^{-21}$	2,152	4,342	200	2,754	2,402	11,850

C. CICIDS2017 and CSE-CIC-IDS2018 Datasets

CIC has developed a testbed framework [77], [78] to generate CICIDS2017 [35] and CSE-CIC-IDS2018 [36] traffic data. The CICIDS2017 dataset includes intrusions that rely on various network vulnerabilities [77] executed using tools for malicious attacks: Patator, Slowloris, Heartleech, Damn Vulnerable Web App, Metasploit, Ares, and Low Orbit Ion Cannon. 84 features including duration, size of packets, number of packets, and number of bytes were extracted using an application for generating and analyzing network traffic flows. We consider application-layer DoS attacks data collected on Wednesday 05.07.2017, labeled GoldenEye, Hulk, SlowHTTPTest, and Slowloris having 10,293, 230,124, 5,499, and 5,796 intrusions, respectively.

The CSE-CIC-IDS2018 dataset was captured over ten days between Wednesday 14.02.2018 and Friday 02.03.2018 [36]. It includes attack scenarios, date, and start and end times of the attack(s). Extracted are 83 features including flow duration, maximum/minimum packet size, and packet flow rate. We consider anomalous instances that include slow-rate low-volume application-layer GoldenEye and Slowloris DoS attacks collected on Thursday 15.02.2018 having 41,508 and 10,990 intrusions, respectively.

IV. EXPERIMENTS AND PERFORMANCE EVALUATION

The experimental procedure consists of four steps: (1) Partitioning datasets for training and testing; (2) Processing data: converting categorical to numerical features, selecting features, and normalizing training and test datasets (features selected during training are applied in the test phase); (3) Using 10-fold cross-validation to train and tune parameters; and (4) Testing and evaluating the generated machine learning models based on accuracy, F-Score, and training time. Performance results were obtained using the same division of datasets for testing in all cases except the NSL-KDD dataset. (In the case of the NSL-KDD dataset, the training and test datasets were predefined.)

The BGP RIPE, NSL-KDD, CICIDS2017, and CSE-CIC-IDS2018 datasets were used to create and evaluate performance of various machine learning models: deep learning RNNs (LSTM and GRU); BLS, RBF-BLS, CFBLS, CE-BLS, and CFEBLS with and without incremental learning; and the newly proposed VFBLS and VCFBLS algorithms with and without incremental learning. We perform two-way classification to identify regular (0) and anomalous (1) data. The datasets are first sorted based on time stamps and then partitioned. The training and test datasets consist of 60 % and 40 % of anomalous data, respectively. While performance of LSTM and GRU depends on the number of hidden layers and nodes [15], [18], BLS classification performance is affected by BLS architecture as well as the number of mapped features, enhancement nodes, and groups of mapped features [44], [45].

The number of selected features was used to evaluate their effect on BLS performance. (Note that RNNs do not require feature selection.) Features are ranked based on importance using the function *sklearn.ensemble.ExtraTreesClassifier()* [79] and by tuning and setting parameters $n\_estimators = 100$ and $random\_state = 1$. For cross-validation of LightGBM models, we vary the number of estimators (10–200) and learning rate (0.01–0.1).

We develop deep learning and bidirectional RNN models having various number of hidden layers and input sequence lengths of 10 (Slammer, Code Red I), 100 (Nimda), and 50 (NSL-KDD, CICIDS2017, CSE-CIC-IDS2018) data points. ReLU is used as the RNN activation function. Layers with 0.5 dropout probability are inserted after FC_2 and FC_3 layers. The optimization algorithm "Adam" is selected to train the RNN models using 30 (BGP RIPE) and 50 (NSL-KDD, CICIDS2017, CSE-CIC-IDS2018) epochs with learning rate $lr = 0.001$. The deep learning neural network model with four hidden layers consists of 37 (BGP RIPE)/109 (NSL-KDD)/78 (CICIDS2017, CSE-CIC-IDS2018) RNNs, 64 FC_1, 32 FC_2, and 16 FC_3 fully connected (FC) hidden nodes.

The CFBLS, CEBLS, and CFEBLS models were implemented by modifying the original BLS functions [45]. For the cross-validation of incremental and non-incremental BLS models, we vary the number of mapped features (10–400), groups of mapped features (1–50), and enhancement nodes (20–700). Parameters of the incremental BLS models are: *incremental learning steps* = 2 (Slammer, Nimda, Code Red I, CICIDS2017, CSE-CIC-IDS2018), 3 (NSL-KDD); *enhancement nodes/step* = 50 (Slammer), 10 (Nimda), 60 (Code Red I, NSL-KDD), 20 (CICIDS2017, CSE-CIC-IDS2018); and *data points/step* = 187 (Slammer), 290 (Nimda), 303 (Code Red I), 3,000 (NSL-KDD), 55,680 (CICIDS2017), 49,320 (CSE-CIC-IDS2018). For the cross-validation of VFBLS and VCFBLS models, we vary mapped features (10–200), groups of mapped features (5–30), and enhancement nodes (40–700). In the case of incremental VFBLS and VCFBLS models, *feature weight for initial step* = 0.9 (BGP RIPE), 0.7 (NSL-KDD, CICIDS2017), 0.85 (CSE-CIC-IDS2018) while the remaining parameters are the same as for incremental BLS. Performance of models was controlled by tuning the parameter λ (16) for sparse regularization in the ridge regression algorithm that was used to calculate output weights during the training process.

Training parameters that generate the best performance results are listed in Table IV to Table IX. Performance of LightGBM, RNN and Bi-RNN (LSTM and GRU) with 2 ($LSTM_2$ and GRU_2), 3 ($LSTM_3$ and GRU_3), 4 ($LSTM_4$ and GRU_4) hidden layers, and BLS models are shown in Figs. 2, 3, and 4. Training times for models leading to the best performance are listed in Table X.

TABLE IV
LIGHTGBM PARAMETERS: BGP RIPE, NSL-KDD, CICIDS2017, AND CSE-CIC-IDS2018 DATASETS

Number of features	Dataset	Number of estimators	Learning rate
16	Slammer	30	0.1
16	Nimda	200	0.01
8	Code Red I	30	0.05
64	NSL-KDD	200	0.01
32	CICIDS2017	200	0.01
64	CSE-CIC-IDS2018	100	0.02

Experimental results show that BLS models offer comparable performance to deep learning GRU and LSTM RNN and Bi-RNN models while requiring shorter training time. Introduced VFBLS and VCFBLS models outperform other BLS models in most cases except models based on Slammer dataset. In the case of NSL-KDD datasets, the VFBLS models show 2 %–15 % improvement in accuracy and 4 %–12 % improvement in F-Score. Even though the LightGBM models have the fastest training time as shown in Table X, its accuracy and F-Score are in some cases lower than the proposed VFBLS and VCFBLS models, as illustrated in Figs. 2, 3, and 4.

TABLE V
BLS AND INCREMENTAL BLS PARAMETERS: BGP RIPE DATASETS

Number of features	Dataset	Model	Mapped features	Groups of mapped features	Enhancement nodes
BLS					
	Slammer	BLS	100	50	100
8	Nimda	CFBLS	300	10	50
	Code Red I	RBF-BLS	200	3	300
	Slammer	CFEBLS	200	50	100
16	Nimda	CFBLS	100	10	200
	Code Red I	CEBLS	100	1	700
	Slammer	BLS	200	30	50
37	Nimda	BLS	100	10	200
	Code Red I	CEBLS	100	1	300
Incremental BLS					
	Slammer	BLS	300	50	100
8	Nimda	CEBLS	300	10	100
	Code Red I	BLS	200	3	500
	Slammer	CFEBLS	200	30	200
16	Nimda	CFEBLS	300	50	200
	Code Red I	CFBLS	300	3	300
	Slammer	CEBLS	100	30	50
37	Nimda	CEBLS	100	10	200
	Code Red I	CFBLS	300	1	700

TABLE VI
BLS AND INCREMENTAL BLS PARAMETERS: NSL-KDD DATASET

Number of features	Model	Mapped features	Groups of mapped features	Enhancement nodes
BLS				
32	BLS	100	40	40
64	RBF-BLS	60	20	40
109	CFBLS	100	20	100
Incremental BLS				
32	CFEBLS	60	20	40
64	RBF-BLS	80	20	40
109	CFBLS	80	30	40

Performance of machine learning models heavily depends on the datasets used for training and test. In the case of BGP RIPE datasets, one of the Bi-GRU models has the best accuracy and F-Score. The proposed VFBLS models have the best accuracy and F-Score for $KDDTest^{+}$ and $KDDTest^{-21}$ and the best F-Score for CSE-CIC-IDS2018 datasets. The GRU models have the best accuracy for CICIDS2017 and CSE-CIC-IDS2018 datasets.

Performance results shown in Fig. 2 (top) illustrate that the extra-trees algorithm may not be able to select the top 8 features that better capture properties of the anomalous data. Hence, selecting 8 features is not sufficient for successfully classifying the Slammer dataset. Note that performance of BLS and incremental BLS remains comparable even without feature selection. While performance of incremental VFBLS and VCFBLS is often inferior to other algorithms, their advantage is that the models need not be retrained for new incoming data. These models exhibit poor performance due to inadequate feature selection. Occurrences of long consecutive regular or anomalous data points in the partitioned training datasets lead to low accuracy and F-Score. If a portion of the

TABLE VII
BLS AND INCREMENTAL BLS PARAMETERS: CICIDS2017 AND CSE-CIC-IDS2018 DATASETS

Number of features BLS	Dataset	Model	Mapped features	Groups of mapped features	Enhancement nodes
32	CICIDS2017	CEBLS	10	10	40
	CSE-CIC-IDS2018	CEBLS	15	20	80
64	CICIDS2017	BLS	10	30	20
	CSE-CIC-IDS2018	RBF-BLS	20	10	80
78	CICIDS2017	RBF-BLS	20	30	40
	CSE-CIC-IDS2018	CFBLS	20	10	80
Incremental BLS					
32	CICIDS2017	CFBLS	10	20	40
	CSE-CIC-IDS2018	BLS	10	20	20
64	CICIDS2017	CFEBLS	20	20	20
	CSE-CIC-IDS2018	CEBLS	20	10	40
78	CICIDS2017	CFBLS	10	20	40
	CSE-CIC-IDS2018	BLS	15	30	20

TABLE VIII
VFBLS AND VCFBLS PARAMETERS: BGP RIPE, NSL-KDD, CICIDS2017, AND CSE-CIC-IDS2018 DATASETS

Number of features VFBLS	Dataset	Mapped features	Groups of mapped features	Enhancement nodes
8, 16, 37	Slammer	100, 30, 40	30, 20, 10	100
	Nimda	20, 40, 30	10, 20, 10	50
	Code Red I	20, 50, 30	10, 10, 20	100
32, 64, 109	NSL-KDD	20, 40, 30	20, 20, 20	40
32, 64, 78	CICIDS2017	15, 10, 10	5, 10, 5	40
	CSE-CIC-IDS2018	10, 20, 10	10, 5, 10	40
VCFBLS				
8, 16, 37	Slammer	200, 30, 30	20, 10, 20	100
	Nimda	20, 30, 30	10, 20, 10	100
	Code Red I	30, 40, 40	10, 10, 10	100
32, 64, 109	NSL-KDD	20, 40, 30	10, 20, 30	60
32, 64, 78	CICIDS2017	10, 20, 10	10, 5, 5	40
	CSE-CIC-IDS2018	10, 10, 20	5, 10, 5	40

TABLE IX
INCREMENTAL VFBLS AND VCFBLS PARAMETERS: BGP RIPE, NSL-KDD, CICIDS2017, AND CSE-CIC-IDS2018 DATASETS

Number of features Incremental VFBLS	Dataset	Mapped features	Groups of mapped features	Enhancement nodes
8, 16, 37	Slammer	40, 10, 20	10, 20, 10	100
	Nimda	50, 50, 40	30, 30, 20	30
	Code Red I	30, 40, 30	10, 1, 2	100
32, 64, 109	NSL-KDD	20, 20, 50	10, 10, 10	40
32, 64, 78	CICIDS2017	10, 10, 10	5, 5, 5	40
	CSE-CIC-IDS2018	15, 10, 10	10, 10, 5	40
Incremental VCFBLS				
8, 16, 37	Slammer	40, 20, 20	10, 20, 20	50
	Nimda	50, 30, 20	30, 10, 10	20
	Code Red I	30, 40, 20	5, 5 , 5	100
32, 64, 109	NSL-KDD	30, 40, 30	20, 20, 20	40
32, 64, 78	CICIDS2017	15, 10, 10	5, 5, 5	20
	CSE-CIC-IDS2018	20, 10, 10	10, 5, 5	20

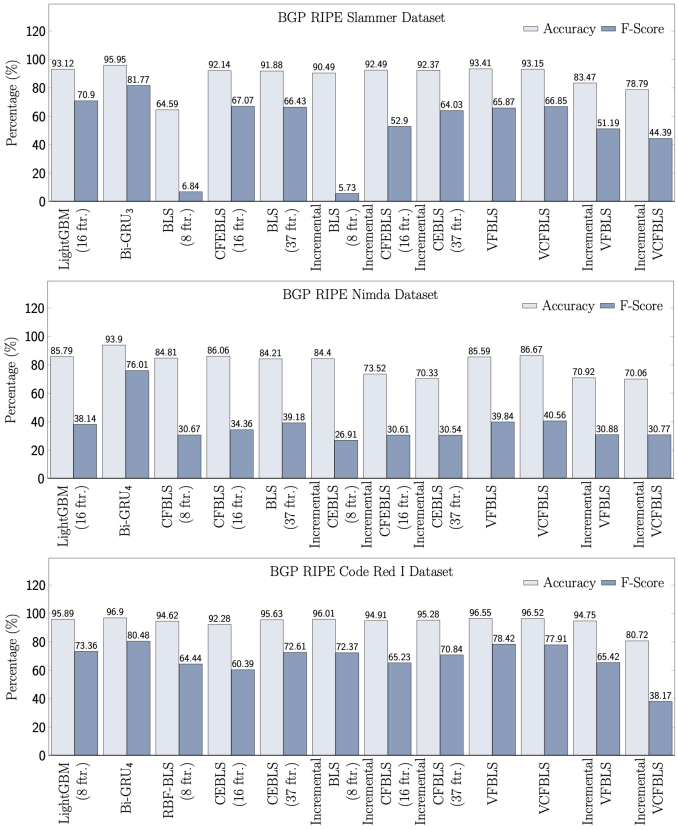


Fig. 2. Best performance results for LightGBM, RNN and Bi-RNN (LSTM and GRU), BLS, Incremental BLS, VFBLS, and VCFBLS models: Slammer (top), Nimda (middle), and Code Red I (bottom) datasets.

training dataset consists of one class only, generated feature importance is 0 and, hence, these features do not contribute to subsequent steps. In the case of BGP RIPE, CICIDS2017, and CSE-CIC-IDS2018 datasets, data points belong to a single class (regular or anomaly) over a long period of time and, hence, features selected during training are not relevant for the test dataset.

BLS and incremental BLS models offer comparable performance to deep learning models and have shorter training time [44] when used with large datasets such as NSL-KDD, CICIDS2017, and CSE-CIC-IDS2018. Introduced VFBLS and VCFBLS models have much shorter training time compared to RNN and Bi-RNN models. The proposed algorithms have an order of magnitude (up to 250 times) shorter training time than Bi-GRU. Their training time is comparable to BLS and its extensions for smaller datasets such as BGP RIPE but it increases for larger datasets because it is mainly spent for selecting features. Incremental variants of VFBLS and VCFBLS require considerably shorter training time. Note that LightGBM models require a much shorter training time than the BLS models.

Note that times reported for training BLS and incremental BLS models do not account for the times spent for selecting features. The overheads for feature selection depend on the size of the training datasets and the number of features. Times spent for selecting features in experiments with the extra-trees algorithm are: 0.13 s (Slammer), 0.22 s (Nimda), 0.24 s (Code Red I), 10.24 s (KDDTrain$^+$), 18.56 s (CICIDS2017), and 13.44 s (CSE-CIC-IDS2018).

Experiments are performed using a Dell Alienware Aurora with 32 GB memory and Intel Core i7 7700K processor. Python 3.6 and libraries [80] *NumPy* (a scientific computing library), *scikit-learn* (a machine learning library), *LightGBM* (a gradient boosting framework), and *PyTorch* (a Python framework for deep learning) were used to create input matrices and to train and test the machine learning models.

TABLE X
TRAINING TIME: BGP RIPE, NSL-KDD, CICIDS2017, AND CSE-CIC-IDS2018 DATASETS

Datasets		LightGBM	RNN	BLS			Incremental BLS			Variable BLS		Incremental Variable BLS	
	Model		Bi-GRU$_3$	BLS	CFEBLS	BLS	BLS	CFEBLS	CEBLS	VFBLS	VCFBLS	VFBLS	VCFBLS
Slammer	(No. of ftr.)	(16)		(8)	(16)	(37)	(8)	(16)	(37)				
	Time (s)	0.02	212.83	6.47	24.09	15.38	216.62	37.83	8.75	9.22	13.86	1.82	1.66
	Model		Bi-GRU$_4$	CFBLS	CFBLS	BLS	CEBLS	CFEBLS	CEBLS	VFBLS	VCFBLS	VFBLS	VCFBLS
Nimda	(No. of ftr.)	(16)		(8)	(16)	(37)	(8)	(16)	(37)				
	Time (s)	0.11	219.50	3.51	1.29	2.01	8.27	43.41	13.35	2.12	1.97	10.37	5.98
	Model		Bi-GRU$_4$	RBF-BLS	CEBLS	CEBLS	BLS	CFBLS	CFBLS	VFBLS	VCFBLS	VFBLS	VCFBLS
Code Red I	(No. of ftr.)	(8)		(8)	(16)	(37)	(8)	(16)	(37)				
	Time (s)	0.02	546.54	5.90	174.21	26.63	1.11	2.00	1.12	1.88	2.55	1.33	1.42
	Model		GRU$_2$	BLS	RBF-BLS	CFBLS	CFEBLS	RBF-BLS	CFBLS	VFBLS	VCFBLS	VFBLS	VCFBLS
NSL-KDD	(No. of ftr.)	(64)		(32)	(64)	(109)	(32)	(64)	(109)				
	Time (s)	0.92	4,831.55	39.77	11.10	24.84	26.05	36.74	83.03	31.21	31.32	28.92	60.43
	Model		GRU$_2$	CEBLS	BLS	RBF-BLS	CFBLS	CFEBLS	CFBLS	VFBLS	VCFBLS	VFBLS	VCFBLS
CICIDS2017	(No. of ftr.)	(32)		(32)	(64)	(78)	(32)	(64)	(78)				
	Time (s)	1.43	15,483.96	39.25	8.97	15.60	6.39	7.39	3.69	25.25	26.05	25.55s	24.19
	Model		GRU$_3$	CEBLS	RBF-BLS	CFBLS	BLS	CEBLS	BLS	VFBLS	VCFBLS	VFBLS	VCFBLS
CSE-CIC-IDS2018	(No. of ftr.)	(64)		(32)	(64)	(78)	(32)	(64)	(78)				
	Time (s)	0.99	26,887.14	33.46	4.65	4.13	5.65	11.59	6.78	21.30	21.38	24.83	14.86

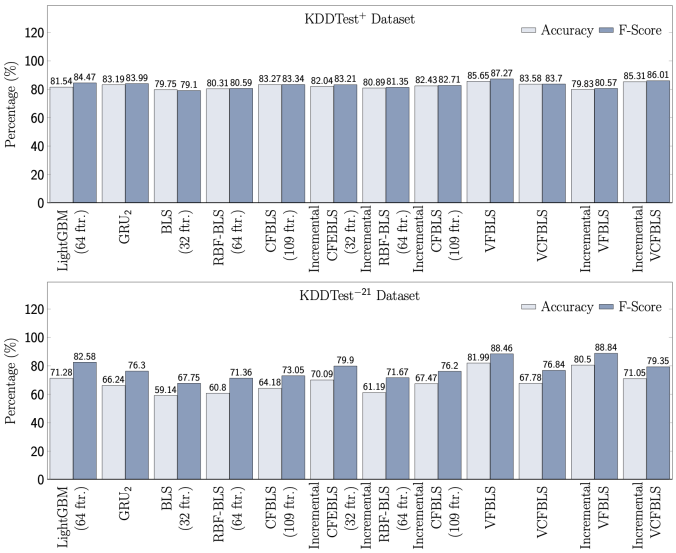


Fig. 3. Best performance results for LightGBM, RNN and Bi-RNN (LSTM and GRU), BLS, Incremental BLS, VFBLS, and VCFBLS models: KDDTest^{+} (top) and KDDTest^{-21} (bottom) datasets.

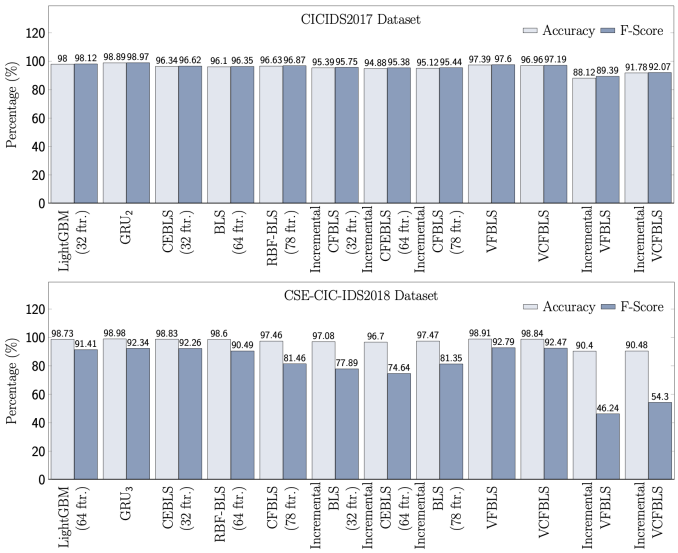


Fig. 4. Best performance results for LightGBM, RNN and Bi-RNN (LSTM and GRU), BLS, Incremental BLS, VFBLS, and VCFBLS models: CICIDS2017 (top) and CSE-CIC-IDS2018 (bottom) datasets.

V. CONCLUSION

In this paper, we evaluated accuracy, F-Score, and training time of LightGBM, RNN, Bi-RNN, and BLS algorithms using BGP RIPE, NSL-KDD, CICIDS2017, and CSE-CIC-IDS2018 datasets. Experiments with BLS models were performed by varying the number of most relevant extracted features. The newly proposed algorithms employed variable number of mapped features and groups of mapped features without (VFBLS) and with (VCFBLS) cascades and a feature selection algorithm. Both algorithms also have incremental learning variants. We also described a procedure for detecting network intrusions.

Performance evaluation indicated that BLS with cascades of enhancement nodes required significantly longer training time. Note that in the case of incremental BLS, the model did not need to be retrained. As expected, larger numbers of mapped features, groups of mapped features, and enhancement nodes required additional memory and longer training time. The advantage of VFBLS and VCFBLS algorithms is their ability to derive generalized models by using various subsets of input data to generate mapped features thus providing an easy process for creating models. Furthermore, VFBLS and VCFBLS models are developed using a single experiment with integrated stages for selecting features and generating models. Training time of these models may be improved by allocating a smaller number of features generated from the input data. In several cases, VFBLS offered the best performance and shorter training time than RNNs and Bi-RNNs and comparable time to other BLS algorithms.

REFERENCES

[1] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1–2, pp. 18–28, Feb.–Mar. 2009.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, July 2009.

[3] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6G wireless networks: carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Veh. Technol. Mag.*, vol. 15, no. 4, pp. 122–134, Dec. 2020.

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

[5] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. Int. Conf. Measurement and Modeling of Comput. Syst.*, Banff, AB, Canada, June 2005, pp. 50–60.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *J. Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sept. 1995.

[7] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

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

[9] 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, pp. 3146–3154.

[10] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[12] 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.

[13] 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.*, vol. 30, no. 4, pp. 1191–1204, Apr. 2019.

[14] Y. Li, H. J. Xing, Q. Hua, X. Z. Wang, P. Batta, S. Haeri, and Lj. Trajković, "Classification of BGP anomalies using decision trees and fuzzy rough sets," in *Proc. IEEE Trans. Syst. Man Cybern.*, San Diego, CA, USA, Oct. 2014, pp. 1331–1336.

[15] Q. Ding, Z. Li, P. Batta, and Lj. Trajković, "Detecting bgp anomalies using machine learning techniques," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Budapest, Hungary, Oct. 2016, pp. 3352–3355.

[16] Q. Ding, Z. Li, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks: datasets and feature selection algorithms," in *Cyber Threat Intelligence*, A. Dehghantanha, M. Conti, and T. Dargahi, Eds. Berlin: Springer, 2018, pp. 47–70.

[17] Z. Li, Q. Ding, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks: classification algorithms," in *Cyber Threat Intelligence*, A. Dehghantanha, M. Conti, and T. Dargahi, Eds. Berlin: Springer, 2018, pp. 71–92.

[18] P. Batta, Z. Li, and Lj. Trajković, "Evaluation of support vector machine kernels for detecting network anomalies," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, May 2018, pp. 1–4.

[19] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[20] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Netw.*, vol. 18, no. 5–6, pp. 602–610, July/Aug. 2005.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Oct. 1997.

[22] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[23] K. Cho, B. van Merriënboer, C. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translations," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1724–1734.

[24] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48 231–48 246, Aug. 2018.

[25] 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.

[26] M. Xu, M. Han, C. L. P. Chen, and T. Qiu, "Recurrent broad learning system for time series prediction," *IEEE Trans. Cybern.*, pp. 1–13, Sept. 2018.

[27] L. Yang, S. Song, and C. L. P. Chen, "Transductive transfer learning based on broad learning system," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Miyazaki, Japan, Oct. 2018, pp. 912–917.

[28] Z. Liu, C. L. P. Chen, T. Zhang, and J. Zhou, "Multi-kernel broad learning systems based on random features: a novel expansion for nonlinear feature nodes," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Bari, Italy, Oct. 2019, pp. 193–197.

[29] (2021, Jan.) Broadlearning. [Online]. Available: http://www.broadlearning.ai.

[30] H. Liu and H. Motoda, *Computational Methods of Feature Selection*. Boca Raton, FL, USA: Chapman and Hall/CRC Press, 2007.

[31] G. Ditzler, J. LaBarck, J. Ritchie, G. Rosen, and R. Polikar, "Extensions to online feature selection using bagging and boosting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 9, pp. 4504–4509, Sept. 2018.

[32] 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. Surv. Tut.*, vol. 21, no. 1, pp. 686–728, First Quarter 2019.

[33] (2021, Jan.) RIPE NCC. [Online]. Available: https://www.ripe.net/analyse.

[34] (2021, Jan.) NSL-KDD Data Set. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html.

[35] (2021, Jan.) Intrusion Detection Evaluation Dataset (CICIDS2017). [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html.

[36] (2021, Jan.) A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018.

[37] (2021, Jan.) University of Oregon Route Views projects. [Online]. Available: http://www.routeviews.org.

[38] B. Al-Musawi, P. Branch, and G. Armitage, "BGP anomaly detection techniques: a survey," *IEEE Commun. Surv. Tut.*, vol. 19, no. 1, pp. 377–396, 2017.

[39] M. C. Libicki, L. Ablon, and T. Webb, *The Defender's Dilemma: Charting a Course Toward Cybersecurity*. Santa Monica, CA, USA: RAND Corporation, June 2015.

[40] J. Zhang, J. Rexford, and J. Feigenbaum, "Learning-based anomaly detection in BGP updates," in *Proc. Workshop Mining Netw. Data*, Philadelphia, PA, USA, Aug. 2005, pp. 219–220.

[41] R. Samrin and D. Vasumathi, "Review on anomaly based network intrusion detection system," in *Proc. Int. Conf. Elect., Electron., Commun., Comput., Optim. Techn.*, Mysuru, India, Dec. 2017, pp. 141–147.

[42] D. Jin, Y. Lu, J. Qin, Z. Cheng, and Z. Mao, "SwiftIDS: real-time intrusion detection system based on LightGBM and parallel intrusion detection mechanism," *Computers & Security*, vol. 97, no. 101984, pp. 1–12, Oct. 2020.

[43] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, Nov. 2017.

[44] Z. Li, P. Batta, and Lj. Trajković, "Comparison of machine learning algorithms for detection of network intrusions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Miyazaki, Japan, Oct. 2018, pp. 4248–4253.

[45] Z. Li, A. L. Gonzalez Rios, G. Xu, and Lj. Trajković, "Machine learning techniques for classifying network anomalies and intrusions," in *Proc. IEEE Int. Symp. Circuits Syst.*, Sapporo, Japan, May 2019, pp. 1–5.

[46] A. L. Gonzalez Rios, G. X. Z. Li, A. D. Alonso, and Lj. Trajković, "Detecting network anomalies and intrusions in communication networks," in *Proc. 23rd IEEE Int. Conf. Intell. Eng. Syst.*, Gödöllő, Hungary, Apr. 2019, pp. 29–34.

[47] 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.

[48] (2021, Jan.) PyTorch. [Online]. Available: https://pytorch.org/docs/stable/nn.html.

[49] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, Apr. 1994.

[50] P. Geurts, D. Ernst, and L.Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, Apr. 2006.

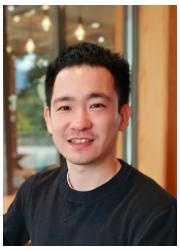
[51] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Proc. Int. Conf. Neural Inform. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2013, pp. 431–439.

[52] J. Zhang and M. Zulkernine, "A hybrid network intrusion detection technique using random forests," in *Proc. First Int. Conf. Availability, Rel. Secur.*, Vienna, Austria, Apr. 2006, pp. 262–269.

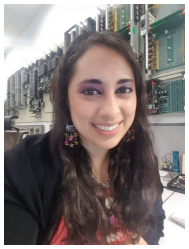
[53] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 303–336, Mar. 2014.

[54] Y. Jia, M. Wang, and Y. Wang, "Network intrusion detection algorithm based on deep neural network," *IET Inf. Secur.*, vol. 13, no. 1, pp. 48–53, Jan. 2019.

[55] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.
[56] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Wireless Netw. Mobile Commun.*, Fez, Morocco, Oct. 2016, pp. 258–263.
[57] D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Commun. Surv. Tut.*, vol. 17, no. 1, pp. 70–91, 2015.
[58] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surv. Tut.*, vol. 18, no. 2, pp. 1153–1176, 2016.
[59] L. Li, Y. Yu, S. Bai, Y. Hou, and X. Chen, "An effective two-step intrusion detection approach based on binary classification and k-NN," *IEEE Access*, vol. 6, pp. 12 060–12 073, Mar. 2018.
[60] Y. Song, A. Venkataramani, and L. Gao, "Identifying and addressing reachability and policy attacks in 'secure' BGP," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2969–2982, Oct. 2016.
[61] (2021, Jan.) MS SQL Slammer/Sapphire worm, SANS Institute GIAC Certifications. [Online]. Available: https://www.giac.org/paper/gsec/3091/ms-sql-slammer-sapphire-worm/105136.
[62] (2021, Jan.) Attack of Slammer worm - a practical case study, SANS Institute GIAC Certifications. [Online]. Available: https://www.giac.org/paper/gcih/414/attack-slammer-worm-practical-case-study/103632.
[63] (2021, Jan.) Responding to the Nimda worm: recommendations for addressing blended threats, Symantec, Cupertino, CA, USA. [Online]. Available: https://vx-underground.org/archive/Symantec/nimda-worm-recommendations-blended-threats-01-en.pdf.
[64] (2021, Jan.) A challenging response to Nimda, SANS Institute GIAC Certifications. [Online]. Available: https://www.giac.org/paper/gcih/273/challenging-response-nimda/102847.
[65] (2021, Jan.) The Code Red worm, SANS Institute Information Security Reading Room. [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/code-red-worm-85.
[66] (2021, Jan.) Border Gateway Protocol Routing Records from Réseaux IP Européens (RIPE) and BCNET. [Online]. Available: http://ieee-dataport.org/1977.
[67] (2021, Jan.) BGP C sharp tool. [Online]. Available: https://github.com/communication-networks-laboratory/BGP_c_sharp_tool.
[68] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation," in *Proc. DARPA Inform. Survivability Conf. Expo.*, Hilton Head, SC, USA, Jan. 2000, pp. 12–26.
[69] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project," in *Proc. USENIX Workshop Tackling Comput. Syst. Problems Mach. Learn. Techn.*, Hilton Head, SC, USA, Jan. 2000, pp. 130–144.
[70] (2021, Jan.) KDD Cup 1999 Data. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[71] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory," *ACM Trans. Inform. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000.
[72] A. A. Olusola, A. S. Oladele, and D. O. Abosede, "Analysis of KDD'99 intrusion detection dataset for selection of relevance features," in *Proc. World Congress Eng. Comput. Sci.*, San Francisco, CA, USA, Oct. 2010, pp. 162–168.
[73] W. Heyi, H. Aiqun, S. Yubo, B. Ning, and J. Xuefei, "A new intrusion detection feature extraction method based on complex network theory," in *Proc. 4^{th} Int. Conf. Multimedia Inf. Netw. Secur.*, Nanjing, China, Nov. 2012, pp. 852–856.
[74] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in a data-flow environment: experience in network intrusion detection," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery and Data Mining*, San Diego, CA, USA, Aug. 1999, pp. 114–124.
[75] H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: a feature relevance analysis on KDD 99 intrusion detection datasets," in *Proc. 3^{rd} Annu. Conf. Privacy Secur. Trust*, St. Andrews, NB, Canada, Oct. 2005, pp. 1–6.
[76] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Ottawa, ON, Canada, July 2009, pp. 1–6.
[77] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Softw. Netw.*, vol. 2017, no. 1, pp. 177–200, July 2017.
[78] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4^{th} Int. Conf. Inform. Syst. Secur. Privacy*, Funchal, Portugal, Jan. 2018, pp. 108–116.
[79] (2021, Jan.) Sklearn.ensemble.ExtraTreesClassifier. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html.
[80] (2021, Jan.) Python package index. [Online]. Available: https://pypi.org.



Zhida Li received the B.E. and M.Eng.Sc. degrees in Electrical Engineering and Microelectronic Design from the University College Cork, Ireland, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree at Simon Fraser University. From 2011 to 2014, he was a Research Assistant at the Tyndall National Institute, Cork, Ireland. His current research project is related to machine learning techniques for classifying network anomalies and intrusions.



Ana Laura Gonzalez Rios received her B.Sc. degree in Electronic and Computer Engineering from the Technology Institute of Monterrey, Puebla, in 2013. She is currently pursuing an M.Sc. degree at the School of Engineering Science at Simon Fraser University. She joined the Communication Networks Laboratory in 2018. Her research interests include application of machine learning algorithms to detecting intrusions and anomalies in communication networks as well as virtual network embedding algorithms.

Ljiljana Trajković received the Dipl. Ing. degree from University of Pristina, Yugoslavia, in 1974, the M.Sc. degrees in electrical engineering and computer engineering from Syracuse University, Syracuse, NY, in 1979 and 1981, respectively, and the Ph.D. degree in electrical engineering from University of California at Los Angeles, in 1986.

She is currently a Professor in the School of Engineering Science at Simon Fraser University, Burnaby, British Columbia, Canada. From 1995 to 1997, she was a National Science Foundation (NSF) Visiting Professor in the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. She was a Research Scientist at Bell Communications Research, Morristown, NJ, from 1990 to 1997, and a Member of the Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ, from 1988 to 1990. Her research interests include communication networks and theory of nonlinear circuits and dynamical systems.

Dr. Trajković served as IEEE Division X Delegate/Director (2019–2020) and IEEE Division X Delegate-Elect/Director-Elect (2018). She served as President (2014–2015), Vice President Publications (2012–2013, 2010–2011), Vice President Long-Range Planning and Finance (2008–2009), and a Member at Large of the Board of Governors (2004–2006) of the IEEE Systems, Man, and Cybernetics Society. She served as 2007 President of the IEEE Circuits and Systems Society and a member of its Board of Governors (2004–2005, 2001–2003). She was General Co-Chair of SMC 2020 and SMC 2020 Workshop on BMI Systems. She serves as Editor-in-Chief of the IEEE Transactions on Human-Machine Systems (2021–2023) and served as an Associate Editor of the IEEE Transactions on Circuits and Systems (Part I) (2004–2005, 1993–1995), (Part II) (2018, 2002–2003, 1999–2001), and the IEEE Circuits and Systems Magazine (2001–2003). She is a Distinguished Lecturer of the IEEE Systems, Man, and Cybernetics Society (2020–2021) and the IEEE Circuits and Systems Society (2020–2021, 2010–2011, 2002–2003). She is a Professional Member of IEEE-HKN and a Life Fellow of the IEEE.