

Feature-based real-time distributed denial of service detection in SDN using machine learning and Spark

Sama Salam Samaan, Hassan Awheed Jeiad

Department of Computer Engineering, University of Technology, Baghdad, Iraq

Article Info

Article history:

Received Sep 7, 2022

Revised Nov 2, 2022

Accepted Dec 5, 2022

Keywords:

Big data

Distributed denial of service detection

Machine learning

Software defined networking

Spark

ABSTRACT

Recently, software defined networking (SDN) has been deployed extensively in diverse practical domains, providing a new direction in network management by separating the control plane from the data plane. Nevertheless, SDN is vulnerable to distributed denial of service (DDoS) attacks resulting from its centralized controller. Several studies have been suggested to address the DDoS attacks in SDN utilizing machine learning approaches. However, these approaches are resource-intensive and cause performance degradation since they cannot perform effectively in large-scale SDN networks that generate vast traffic statistics. To handle all these challenges, we build a DDoS attack detection model in SDN using Spark as a big data tool to overcome the limitations of conventional data processing methods. Four machine learning algorithms are employed. The decision tree (DT) is elected to be used for real-time deployment based on the performance results, which indicates that it has the best accuracy of 0.936. The model performance is compared with state-of-the-art and shows an overall better performance.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Sama Salam Samaan

Department of Computer Engineering, University of Technology

Baghdad, Iraq

Email: sama.s.samaan@uotechnology.edu.iq

1. INTRODUCTION

With the rapid improvements in the IT infrastructure field, networks have become larger, leading to increased complexity. Consequently, fundamental network characteristics such as confidentiality, integrity, authentication, non-repudiation, and information availability become demanding. In recent years, many industries and researchers have shifted their interest to designing more scalable, robust, and secure networks. Software defined networking (SDN) is considered the latest advancement in networking as a step towards building a centralized and dynamic network as opposed to the distributed and static environment of conventional IP-based networks, which are complicated to manage.

When there is a need to implement a new policy in a traditional network, the network operator should configure every device using vendor-specific commands. As a result, it becomes very demanding to implement the required policy in existing IP-based networks. Another side of the inflexibility of such networks is the vertical integration in which the control plane and the data plane are tightly coupled, adding more rigidity to the overall architecture and hampers the evolution of network infrastructure. Therefore, SDN is considered a suitable solution to the problems of the existing IP networks mentioned above. SDN is an evolving architecture that presents an aspiration for a competent network infrastructure for the following reasons. First, it ends the vertical integration of the traditional networks by separating the control plane that determines packet routes from the data plane that forwards these packets. Second, by this separation, the

control logic of the whole network is laid down in a logically centralized controller. Therefore, network switches turn into simple forwarding devices that simplify network management and add rapid implementation, flexibility, and programmability.

Although the SDN architecture can enhance network security with a global overview of the entire network, a centralized controller, and the creation of forwarding rules in demand [1], SDN still has some challenges and concerns like scalability, supportability, and security. Among all these aspects, security has significant prominence. Because the centralized controller is responsible for network management, the controller failure would hinder the entire network. Therefore, the controller and the communication links between the controller and their assigned switches might be victims of complicated distributed denial of service (DDoS) attacks. Besides, [2] shows that in SDN, the DDoS attacks can be classified based on the targeted plane into data plane DDoS attacks, control plane DDoS attacks, and application plane DDoS attacks.

Presently, there are two basic approaches to detecting DDoS attacks in SDN. The first approach sets a threshold for monitoring traffic indicators like maximum entropy, traffic rate, and packet delay. The network may encounter an attack if these indicators surpass a pre-defined threshold. This approach leads to excessive false-error rates due to the rigid threshold. The second approach depends on traffic flow features using machine learning algorithms to classify traffic into benign or attack. Although this approach is widely adopted since it performs better than the first one, it is considered highly resource-intensive because the amounts of traffic statistics in large-scale networks are huge. In these large-scale networks, for instance, data centers and cloud servers, conventional data processing systems pose some limitations when processing massive amounts of data [3]. For all these reasons, there is a need for scalable, reliable, and efficient big data solutions to process such amounts of data in real time with high accuracy and minimum prediction speed. The main contributions of this work are as follows.

- Design a machine learning pipeline that represents a seamless workflow to combine a set of stages as a single entity. The pipeline design is achieved in the learning stage of the proposed model.
- The analysis of variance (ANOVA) F-test technique is used for feature selection to reduce dimensionality, improve classification effectiveness, and lower computation time.
- Evaluate the produced models in a streaming data environment by incorporating Spark Streaming.
- The model with the highest accuracy is elected and deployed in the online DDoS detection in the SDN network. This step is accomplished in the deployment stage of the proposed model.

Detection and classification mechanisms of traffic anomalies should be flexible and straightforward to detect the fast-growing types of anomalies. The design of such mechanisms is complicated since they need to have a precise and global view of the entire network, the capability to detect new types of attacks, and the necessity to classify these attacks accurately. da Silva *et al.* [4] proposed anomaly detection and machine learning traffic classification for software-defined networking (ATLANTIC), a framework to detect, classify, and mitigate DDoS attacks jointly. The suggested framework consists of two phases. The first phase, which is the lightweight phase, computes the entropy deviations of flow tables. The second phase, which is the heavyweight phase, deployed support vector machine (SVM) to classify traffic into normal and abnormal.

Although the centralized controller in SDN represents its primary benefit, sometimes it becomes a vital security threat. When an intruder succeeds in attacking the controller, he will gain access to the whole network. Therefore, it is crucial to detect attacks on the SDN controller ahead of time. Meti *et al.* [5] suggested two classifiers utilizing SVM and neural network to detect doubtfully and likely damaging in the SDN controller connections.

The cooperation between the cloud and SDN can facilitate the difficulties in a traditional cloud platform, such as the particular isolation of the cloud user and network flow control. While in an SDN-based cloud, the centralized SDN controller is vulnerable to DDoS attacks, disabling the whole network. Chen *et al.* [6] suggested an improved DDoS detection method named extreme gradient boosting (XGBoost) that is based on the conventional gradient boosting decision tree (GBDT) algorithm. This method attempts to differentiate the benign flow from the attack flow utilizing the greedy GBDT that analyses the traffic flow features.

In traditional machine learning, essential input features are designed manually, and the system automatically learns how to map these features to an output. While in deep learning, there are various levels of features that are discovered automatically and are composed together at different levels to generate outputs. Tang *et al.* [7] suggested a deep learning intrusion detection system in SDN. They built a deep neural network consisting of an input layer with three neurons, three hidden layers with 12, 6, and 3 neurons, respectively, and an output layer with two neurons. They used 6 out of 41 features of the NSL-KDD dataset for training the model. The acquired accuracy was 75.75%.

Stacked autoencoder (SAE) is a deep learning approach consisting of stacked sparse autoencoders and soft-max classifiers for unsupervised learning and classification. Niyaz *et al.* [8] suggested a deep learning approach based on a SAE to detect DDoS attacks in SDN networks. The SAE resulted in an

accuracy equal to 95.65%. Despite the high accuracy, the two-stage process of the SAE unveils exceptional complexity that drains SDN network resources.

Nowadays, SDN is widely used in diverse practical domains, providing a new way to manage networks by separating the control plane from its data plane. Nevertheless, it becomes vulnerable to DDoS attacks because of its centralized control. Research by Dinh and Park [3], utilized big data tools in building a framework to overcome conventional limitations in data processing and detect DDoS attacks in a large-scale SDN network. The suggested framework consists of three sequential stages; data ingestion, data preprocessing, and machine learning model training and deployment. Apache Kafka, Spark streaming, Spark core, hadoop distributed file system (HDFS), and Spark machine learning library (MLlib), are the big data tools used in the proposed framework.

2. THEORETICAL CONCEPTS

Before diving deeply into the proposed SDN real-time DDoS detection model, it is essential to describe the theoretical fundamentals related to the proposed work. This section briefly introduces the concepts and techniques applied in this work. It includes the Spark MLlib, machine learning pipeline, the dataset used, and the ANOVA F-test technique for feature selection.

2.1. Machine learning with spark

Recently, various types of structured and unstructured data are likely generated by humans and machines of huge sizes. As a result, solving machine learning problems using traditional techniques face a big challenge. Here comes the need for a distributed machine learning framework to handle these problems efficiently. Developed on top of Spark, MLlib is a library that provides preprocessing, model training, and making predictions at scale on data [9]. Various machine learning tasks can be performed using MLlib like classification, regression, clustering, deep learning, and dimensionality reduction. MLlib integrates seamlessly with other Spark components like Spark Streaming, Spark SQL, and DataFrames [10]. In Spark, a DataFrame is accumulated data arranged into named columns distributed across multiple nodes.

2.2. Machine learning pipelines

The concept of pipelines facilitates the creation, tuning, and examination of machine learning workflows. It consists of stages chained together to automate a machine learning workflow [11]. Each stage is either an estimator or a transformer. An estimator is an abstraction of an algorithm fitted on a DataFrame to create a transformer; e.g., a learning algorithm is an estimator which trains on a DataFrame and develops a fitted model. A transformer is an algorithm that transforms one DataFrame into another by deleting, adding, or updating existing features in the DataFrame. For example, a machine learning model is a transformer that transforms a DataFrame with features into a DataFrame with predictions appended as columns. Pipeline stages are run consecutively, and the input DataFrame is converted as it goes across each stage. The pipeline design is elaborated in section 4.

2.3. Dataset description

For training purposes, a dataset [12] is used that has been created by executing a transmission control protocol (TCP-SYN) flood attack. In this attack, the targeted machine is flooded with bogus SYN requests by deceived IP addresses. Since these addresses are deceived, the machine will never have additional replies for its SYN/ACK packets. As a result, the corresponding port is kept needlessly open. When the number of bogus SYN requests becomes significant, all the ports of the targeted machine are blocked, and it becomes unable to communicate with authorized users [13].

This dataset consists of 28 feature columns and 1,536,950 records. Traffic flow information is carried in each record, such as source and destination IP addresses, TCP port numbers, and TCP window size. Numeric features are the majority in this dataset. In addition, there are nominal features. The class column contains two values; DDoS and benign.

2.4. Feature selection

It is familiar to have hundreds or even thousands of features in today's datasets. More features might give more information about each record. However, these additional features might introduce complexity without offering valuable information [14]. The biggest challenge in machine learning is building robust predictive models using a minimum set of features. The concept of feature selection is to eliminate the number of input features when building a predictive model to enhance the overall performance. It aims to mitigate problems such as the curse of dimensionality and computational cost.

Nevertheless, given the sizes of massive datasets, it isn't easy to figure out which feature is important and which isn't. This work uses the ANOVA F-test, which stands for ANOVA. It determines whether the means from three or more data instances come from identical distribution. F-test, also known as F-statistics, is a statistical test that determines the ratio between the explained and unexplained variance using a statistical test like ANOVA. The ANOVA method used in this context is a class of F-statistics known as the ANOVA F-test. Above all, ANOVA is deployed in classification tasks when the input features are numerical and the target feature is categorical [15], which is the case of the employed dataset in this work. The results of the F-test are used in feature selection.

3. THE PROPOSED MODEL FOR THE FEATURE-BASED REAL-TIME DDoS DETECTION IN SDN

This section presents the proposed model for DDoS detection in SDN using Spark. The main contribution of this paper is located in the SDN application plane. In this model, shown in Figure 1, two stages are introduced; learning and deployment. Each stage is explained in the subsections.

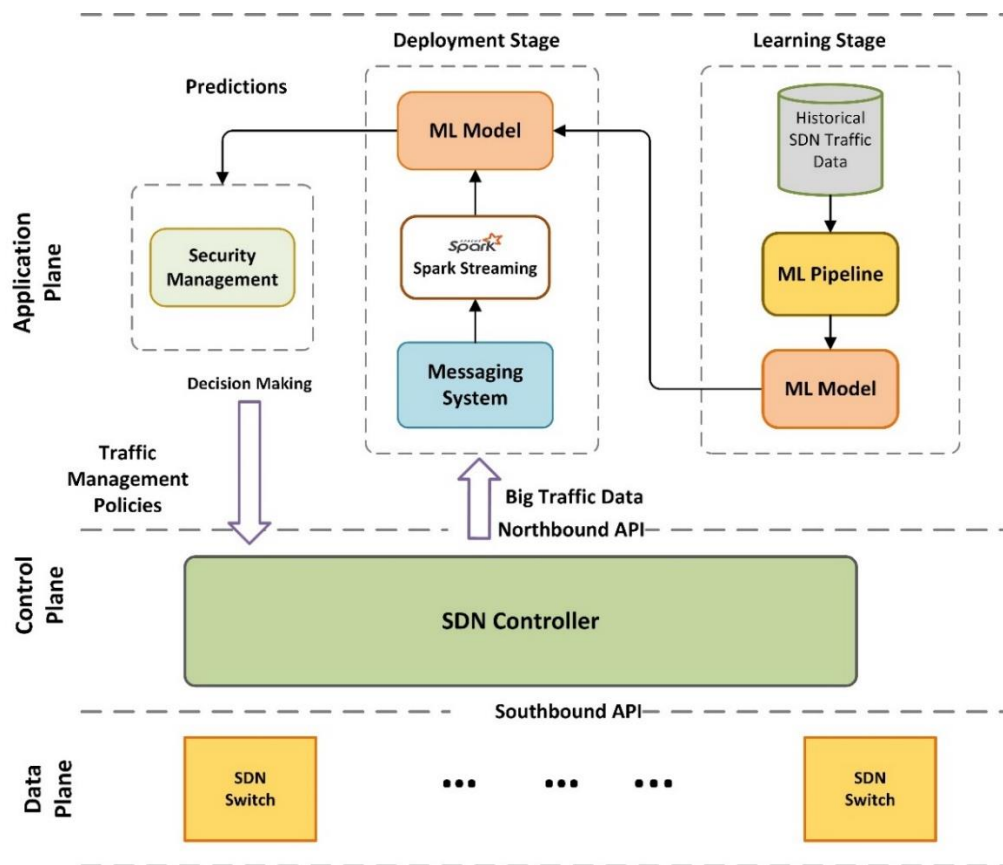


Figure 1. The proposed SDN real-time DDoS detection model using Spark

3.1. Learning stage

This stage (illustrated in Figure 2) highlights the essential contribution of this work. In this stage, a machine learning pipeline is designed as a powerful method to automate complicated machine learning workflows. Before explaining the pipeline design, the nominal typed features are dropped from the dataset since the vector assembler, the second stage in the presented pipeline, accepts only numeric, Boolean, and vector types [15]. The dropped features are the *frame number*, *eth src*, *eth dst*, *eth type*, *ip id*, *ip flags*, *ip src*, *ip dst*, *tcp flags*, and *tcp payload*. In addition, duplicated records are removed since they might be a reason for non-random sampling and could bias the fitted model [16]. The number of removed records equals 109,565. As a result, the number of records becomes 1,427,385.

Afterwards, the resulting DataFrame is randomly divided into 70% for training and 30% for testing. Two subsets are constructed. The first subset (training DataFrame) is used to train the model. The second

subset (testing DataFrame) is used for model evaluation to realize how the model performs on unseen data. The training DataFrame consists of 999,120 records, and the testing DataFrame consists of 428,265 records.

Part (a) of Figure 2 illustrates the machine learning pipeline design consisting of six stages. The first five stages are data preprocessing stages. The third stage is the feature selector, in which the ANOVA F-test method is used. After removing the nominal features and applying the feature selection, the number of the remaining features is five features. The scaled features obtained from the fourth stage and the label column resulting from the string indexer stage are used in the machine learning model building in the final stage. The training DataFrame is used in fitting the pipeline to produce the fitted model, while the testing DataFrame transforms the fitted model and makes the predictions.

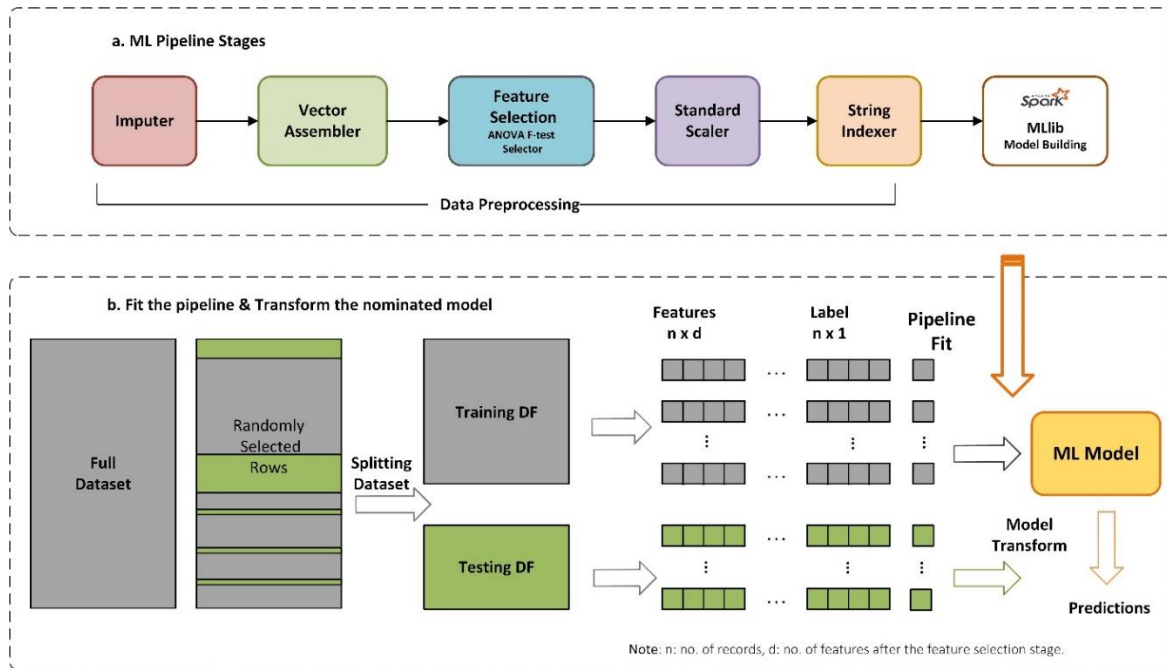


Figure 2. The learning stage

The pipeline consists of the following components:

- **Imputer:** handling missing values is an essential step since several machine learning algorithms do not allow such values [17]. The imputer is an estimator used to complete the missing values by mean, median, or mode of numerical columns. In this case, the mean is used, which is calculated from the remaining values in the related column. The pseudocodes for feature mean computation and imputation process are listed.

Step 1-Pseudocode for feature mean computation

```

For i = 1 to total_no_of_features
  For j = 1 to total_no_of_instances
    feature_sumi =  $\sum_{j=1}^{total\_no\_of\_instances} feature\_value_j$ 
  EndFor
  feature_meani =  $\frac{feature\_sum_i}{feature\_count_i}$ 
EndFor

```

Step 2-Pseudocode for feature imputation

```

For i = 1 to total_no_of_features
  For j = 1 to total_no_of_instances
    IF feature_valueij = null || none || nan || ' '
      feature_valueij = feature_meani
    EndIF
  EndFor
EndFor

```

- Vector assembler: Spark machine learning works differently from other systems. It operates on a single column rather than an array of different columns. The raw features are combined into a single vector to select the features in the next stage. The vector assembler is a transformer that combines multiple columns into a single vector column. Figure 3 shows the result of this stage for the first five records. The vector length equals 17, which refers to the number of the remaining features after removing the nominal features.

| features |
|---|
| ▶ {"vectorType": "dense", "length": 17, "values": [20, 67, 64, 6, 37140, 5001, 406, 27, 0, 28, 0, 20, 8192, 8192, 28, 27, 81]} |
| ▶ {"vectorType": "dense", "length": 17, "values": [20, 67, 64, 6, 43038, 5001, 471, 27, 0, 28, 0, 20, 8192, 8192, 28, 27, 81]} |
| ▶ {"vectorType": "dense", "length": 17, "values": [20, 67, 64, 6, 24099, 5001, 577, 27, 0, 28, 0, 20, 8192, 8192, 28, 27, 81]} |
| ▶ {"vectorType": "dense", "length": 17, "values": [20, 67, 64, 6, 13862, 5001, 805, 27, 0, 28, 0, 20, 8192, 8192, 28, 27, 81]} |
| ▶ {"vectorType": "dense", "length": 17, "values": [20, 67, 64, 6, 24628, 5001, 1306, 27, 0, 28, 0, 20, 8192, 8192, 28, 27, 81]} |

Figure 3. The vector resulted from the vector assembler stage for the first five records

- Feature selection: this work employed the ANOVA F-test as a feature selection technique to remove features independent of the target. As a result, the following features are removed:

ip_hdr_len, ip_ttl, ip_proto, tcp_seq, tcp_nextseq, tcp_ack, tcp_hdr_len, frame len, tcp_window_size_value, tcp_window_size, bytes_in_flight, and push_bytes_sent.

And the remaining features are *ip len, tcp src port, tcp dst port, tcp len, and tcp stream*. Figure 4 shows the result of this stage for the first five records.

| selectedFeatures |
|---|
| ▶ {"vectorType": "dense", "length": 5, "values": [67, 37140, 5001, 406, 27]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [67, 43038, 5001, 471, 27]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [67, 24099, 5001, 577, 27]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [67, 13862, 5001, 805, 27]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [67, 24628, 5001, 1306, 27]} |

Figure 4. The vector resulted from the feature selection stage for the first five records

- Standard scaler: values of some features might range from small to vast numbers. This stage transforms a vector row by normalizing each feature with zero mean. While it is an optional stage, it helps in reducing the convergence time. Figure 5 shows the result of this stage for the first five records. The pseudocode for feature standard deviation computation and standard scaler computation are listed below.

| Scaled_features |
|--|
| ▶ {"vectorType": "dense", "length": 5, "values": [5.5451441286394685, 1.8383855183079825, 0.2881878865687194, 0.003282293937290253, 2.0666288657892005]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [5.5451441286394685, 2.1303294544140803, 0.2881878865687194, 0.003807784345969727, 2.0666288657892005]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [5.5451441286394685, 1.1928716372025867, 0.2881878865687194, 0.004664737935508562, 2.0666288657892005]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [5.5451441286394685, 0.6861523978132809, 0.2881878865687194, 0.00650799659979964, 2.0666288657892005]} |
| ▶ {"vectorType": "dense", "length": 5, "values": [5.5451441286394685, 1.219056503631906, 0.2881878865687194, 0.01055831498054451, 2.0666288657892005]} |

Figure 5. The vector resulted from the standard scaler stage for the first five records

Step 1-Pseudocode for feature standard deviation computation

```

-----
For i = 1 to total_no_of_features
  For j = 1 to total_no_of_instances
    
$$feature\_stddev_i = \sqrt{\frac{1}{feature\_count_i} \sum_{i=1}^{feature\_count_i} (feature\_value_j - feature\_mean_i)^2}$$

  EndFor
EndFor

```

Step 2-Pseudocode for feature standard scaler computation

```

-----
For i = 1 to total_no_of_features
  For j = 1 to total_no_of_instances
    
$$feature\_standardization_{ij} = \frac{feature\_value_j - feature\_mean_i}{feature\_stddev_i}$$

  EndFor
EndFor

```

- String Indexer: in this stage, the class feature is mapped from string labels to a column of label indices (zero or one). As a result, benign is indexed as 0, and DDoS is indexed as 1.
- Machine learning model building: this is the final stage of the pipeline in which the machine learning models are built using the outcomes from the previous stages. Four machine learning algorithms available in Spark MLlib are utilized: decision tree (DT), random forest (RF), logistic regression (LR), and gradient boosted trees (GBT). The deployed machine learning algorithms with their tuned parameters are presented as follows:

a. Decision tree

The DT is a supervised learning algorithm that handles continuous and discrete data. Data in DT is split continuously according to a specific parameter. It is used to represent decisions and decision-making explicitly [18]. As the name suggests, DT is a tree-based model characterized by its simplicity in understanding decisions and the ability to select the most preferential features [19]. In addition, it can classify data without vast calculations [20].

b. Random forest

The RF comes under the supervised learning algorithms used in classification problems. It depends on ensemble learning that unites multiple classifiers to solve complicated problems and enhance the model performance. One of RF's strengths is its efficiency in handling massive training datasets [21].

c. Logistic regression

The LR is a predictive analysis, supervised learning algorithm for classifying categorical variables. It is built on the concept of probability. In LR, the output is transformed using the logistic sigmoid function to return a probability value.

d. Gradient boosted trees

Gradient boosting is a machine learning algorithm used in classification and regression. It produces a prediction model in the shape of an ensemble of weak prediction models, predominantly decision trees [22]. Hyperparameters are configurations that specify the main structure of the model and influence the training process, namely model architecture and regularization. The hyperparameters of all the above models are set according to Table 1. After all the stages are prepared, they are placed in the pipeline. Using the training DataFrame, the pipeline is fitted to produce the machine learning models to be evaluated and used in the deployment stage to make predictions. This is illustrated in part (b) of Figure 2.

3.2. Deployment stage

The deployment stage consists of the messaging system, Spark streaming, and the machine learning model. The messaging system is in charge of transmitting traffic statistics from the SDN controller to Spark streaming to perform the required analysis. The chosen messaging system should be scalable, fault-tolerant, elastic, and can transfer high volumes of data in real time with low latency. Apache Kafka has all these capabilities [23]. In addition, it can be integrated conveniently with the OpenDayLight (ODL) since this controller has a northbound plugin that allows real-time event streaming into Kafka [24], [25]. The ODL controller publishes traffic flow data as messages on Kafka using a common topic. Next, Spark streaming subscribes to that topic and gets the message streams from Kafka.

Spark streaming represents the analytics point that performs data cleaning and preprocessing to generate the required information for the machine learning model [26]. The machine learning models built in the learning stage are evaluated in terms of accuracy and speed. Based on the evaluation results, the machine learning model with the best performance is used in the deployment stage to predict the traffic type

(benign or DDoS). These predictions are utilized in security decisions to lower the required time to detect security threats. The deployment stage implementation is currently beyond the scope of this work.

Table 1. The hyperparameter tuning for the machine learning algorithms

| Model | Parameter | Explanation | Value |
|-------|-----------------|--|--------|
| DT | maxDepth | The maximum tree depth, the default equals 5 | 10 |
| | impurity | The required criterion for the selection of information gain (gini or entropy) | gini |
| LR | maxIter | max number of iterations | 150 |
| | family | Can be multinomial or binary | binary |
| | standardization | Whether to standardize the training features before fitting the model | TRUE |
| | elasticNetParam | A floating-point value from 0 to 1. This parameter specifies the mix of L1 and L2 regularization according to elastic net regularization | 0.8 |
| RF | numTrees | The total number of trees to train | 200 |
| | maxDepth | Maximum depth of the tree, must be in range [0, 30] | 8 |
| | maxBins | Max number of bins for discretizing continuous features, must be ≥ 2 | 32 |
| GBT | maxIter | Total number of iterations over the data before stopping | 10 |

4. MACHINE LEARNING MODELS EVALUATION AND TESTING

This section evaluates and tests the machine learning models built in the learning stage. The pipeline implementation, evaluation, and testing are done on a laptop with CPU Intel Core (TM) i7 and installed memory (RAM) of 16 GB. The software tools are Apache Spark 3.3.0 and Python 3.10. The amount of memory needed for the driver process is configured to 15 GB to prevent out of memory errors.

4.1. Machine learning models evaluation

In this subsection, the accuracy of each machine learning model is computed. The model with the highest performance will be applied in the deployment stage. Accuracy is calculated according to the following (1) [27]:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{All\ Samples} \quad (1)$$

True positives, also known as sensitivity, are results where the model predicts the positive classes correctly. True negatives, also known as specificity, are results where the model predicts the negative classes correctly.

Besides accuracy, precision, recall, and F1-score metrics are computed because accuracy alone is inadequate to evaluate the performance of the models. The precision ratio outlines the model performance in predicting the positive classes. It is calculated as (2):

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (2)$$

False positives are results where the model mispredicts the positive classes. The recall ratio is computed as (3):

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (3)$$

False negatives are results where the model mispredicts the negative classes. F1-score is computed according to the following (4):

$$F1 - score = 2 \times \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

Figure 6 shows the overall statistics for the machine learning models built in the learning stage. DT and GBT models have comparable metrics. They exhibit the best accuracy, F1-score, recall, and precision among the other candidates. As part of the evaluation process, we measure the training time when employing the machine learning pipeline in the learning stage. This is illustrated in Figure 7.

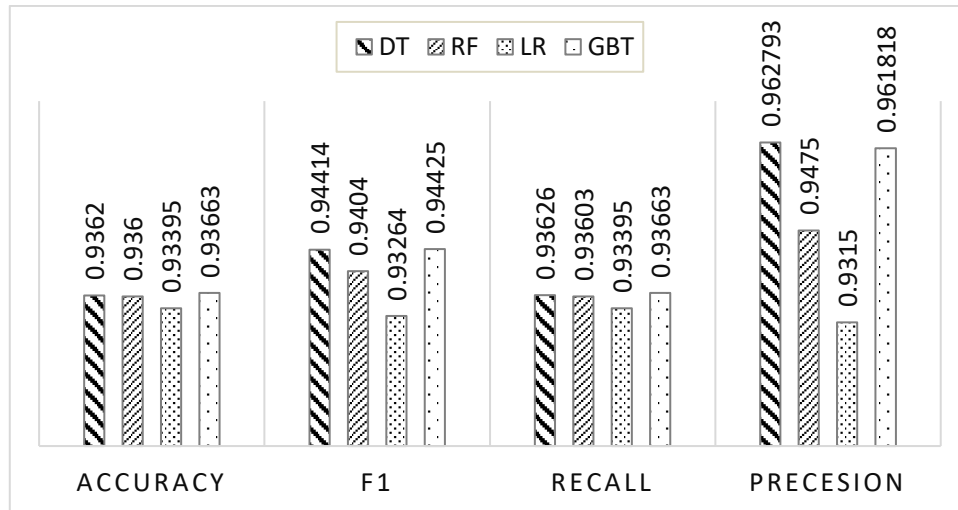


Figure 6. Accuracy, F1, recall, and precision for the four machine learning algorithms



Figure 7. Training time in seconds using machine learning pipeline

4.2. The prototype testing

To test the machine learning models produced from the learning stage, there is a need to replicate the online data streaming. Therefore, a prototype is implemented in which the testing DataFrame is repartitioned into 100 different files; each file has approximately 4,282 records. Spark streaming accepts data from various sources (e.g., file source, Flume, and Kafka) and processes it in real time. Figure 8 illustrates the streaming process from the file directory as a data source for Spark streaming. In the implemented prototype, Spark Streaming listens to the file directory where the testing files are stored. Since the DT model has the best accuracy, it is used to make predictions.

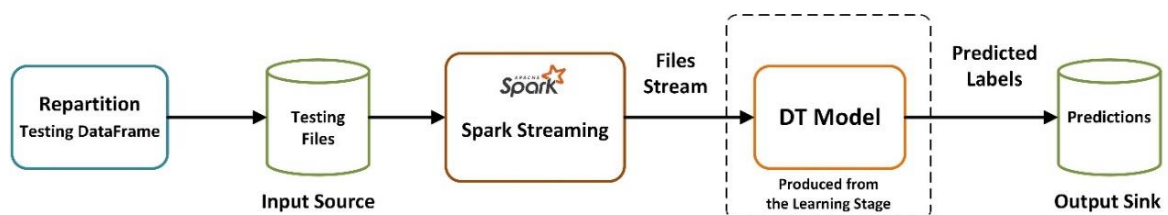


Figure 8. File source for streaming data to the machine learning model

Figure 9 shows a sample snapshot of the streaming, including the actual label, the probability, and the model prediction. As seen in the first row, the vector in the probability column is [1, 0]. The first value in the vector is the probability of class 0 (benign), and the second value is the probability of class 1 (DDoS). The model chooses the largest probability and designates the streamed data to the class with that probability. In the case of the first row, the model selects the larger probability, i.e., 1, and designates the streamed data to class 0 (benign), and it is correct compared with the actual label. In the second row, the model chooses the largest probability, i.e., 0.5384760349268171, and designates the streamed data to class 1 (DDoS), which is a false prediction compared with the actual label.

| | label ▲ | probability ▲ | prediction ▲ |
|---|---------|---|--------------|
| 1 | 0 | ▶ {"vectorType": "dense", "length": 2, "values": [1, 0]} | 0 |
| 2 | 0 | ▶ {"vectorType": "dense", "length": 2, "values": [0.461523965073183, 0.5384760349268171]} | 1 |
| 3 | 0 | ▶ {"vectorType": "dense", "length": 2, "values": [1, 0]} | 0 |
| 4 | 0 | ▶ {"vectorType": "dense", "length": 2, "values": [1, 0]} | 0 |
| 5 | 0 | ▶ {"vectorType": "dense", "length": 2, "values": [1, 0]} | 0 |

Figure 9. Sample output of the label, probability, and prediction on the unseen data

5. CONCLUSION

This paper presented the architecture for a feature-based, real-time DDoS detection model for SDN using Spark. The proposed model consists of two stages; learning and deployment. A pipeline is designed to streamline and automate the machine learning processes in the learning phase. Four machine learning models are built using the Spark machine learning library; decision tree, random forest, logistic regression, and GBT. These models are evaluated in four terms of evaluation metrics; accuracy, precision, recall, and f1-score. In addition, the training time of each model is measured. Results show that the decision tree and the GBT models have the best accuracy with 0.9362 and 0.9366, and training time equals 54 sec and 72 sec, respectively, when using the machine learning pipeline. Spark streaming is incorporated to stream the data to the elected machine learning model to replicate the online network traffic flow of data. In future work, the second stage of the proposed model, i.e., deployment, is intended to be implemented to utilize the valuable information acquired from the learning stage in diverse network security and management aspects.




REFERENCES

- [1] S. Ghaly and M. Z. Abdullah, "Design and implementation of a secured SDN system based on hybrid encrypted algorithms," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 19, no. 4, pp. 1118–1125, Aug. 2021, doi: 10.12928/telkomnika.v19i4.18721.
- [2] J. Singh and S. Behal, "Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions," *Computer Science Review*, vol. 37, pp. 1–25, Aug. 2020, doi: 10.1016/j.cosrev.2020.100279.
- [3] P. T. Dinh and M. Park, "BDF-SDN: A big data framework for DDoS attack detection in large-scale SDN-based cloud," in *2021 IEEE Conference on Dependable and Secure Computing (DSC)*, Jan. 2021, pp. 1–8, doi: 10.1109/DSC49826.2021.9346269.
- [4] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2016, pp. 27–35, doi: 10.1109/NOMS.2016.7502793.
- [5] N. Meti, D. G. Narayan, and V. P. Baligar, "Detection of distributed denial of service attacks using machine learning algorithms in software defined networks," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2017, pp. 1366–1371, doi: 10.1109/ICACCI.2017.8126031.
- [6] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, and J. Peng, "XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jan. 2018, pp. 251–256, doi: 10.1109/BigComp.2018.00044.
- [7] 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 *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Oct. 2016, pp. 258–263, doi: 10.1109/WINCOM.2016.7777224.
- [8] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based DDoS detection system in software-defined networking (SDN)," *ICST Transactions on Security and Safety*, vol. 4, no. 12, pp. 1–18, Dec. 2017, doi: 10.4108/eai.28-12-2017.153515.
- [9] A. Mahmood and A. Kareem, "A new processing approach for scheduling time minimization in 5G-IoT networks," *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 3, pp. 481–492, Jun. 2021, doi: 10.22266/ijies2021.0630.40.
- [10] Apache Spark, "Machine learning library (MLlib) guide," *Apache Spark*, 2015. [https://spark.apache.org/docs/latest/ml-lib-guide.html%5Cn\[29.07.2015\]](https://spark.apache.org/docs/latest/ml-lib-guide.html%5Cn[29.07.2015]) (accessed Aug. 08, 2022).
- [11] S. Wang, J. Luo, and L. Luo, "Large-scale text multiclass classification using spark ML packages," *Journal of Physics: Conference Series*, vol. 2171, no. 1, pp. 1–6, Jan. 2022, doi: 10.1088/1742-6596/2171/1/012022.
- [12] K. Gupta, S. Sharma, and S. Kumar, "SDN-DDOS-TCP-SYN dataset," *Mendeley Data*, 2021. <https://data.mendeley.com/datasets/8nb4cdwc9h> (accessed Aug. 08, 2022).




- [13] H. S. Yaseen and A. Al-Saadi, "Load balancing and detection of distributed denial of service attacks using entropy detection," *Iraqi Journal of Computer, Communication, Control and System Engineering (IJCCCE)*, vol. 21, no. 4, pp. 60–73, Dec. 2021, doi: 10.33103/uot.ijccce.21.4.6.
- [14] S. M. Najem and S. Kadhem, "An efficient feature engineering method for fraud detection in e-commerce," *Iraqi Journal of Computer, Communication, Control and System Engineering (IJCCCE)*, vol. 21, no. 3, pp. 40–52, Sep. 2021, doi: 10.33103/uot.ijccce.21.3.4.
- [15] A. Spark, "Spark documentation," 2018. <https://spark.apache.org/docs/3.2.1/index.html> (accessed Jul. 10, 2022).
- [16] M. P. J. Kuranage, K. Piamrat, and S. Hamma, "Network traffic classification using machine learning for software defined networks," in *International Conference on Machine Learning for Networking*, 2020, pp. 28–39, doi: 10.1007/978-3-030-45778-5_3.
- [17] S. D. Khudhur and H. A. Jeiad, "A content-based file identification dataset: Collection, construction, and evaluation," *Karbala International Journal of Modern Science*, vol. 8, no. 2, pp. 63–70, May 2022, doi: 10.33640/2405-609X.3222.
- [18] J. Xie *et al.*, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2019, doi: 10.1109/COMST.2018.2866942.
- [19] N. F. Idris, M. A. Ismail, M. S. Mohamad, S. Kasim, Z. Zakaria, and T. Sutikno, "Breast cancer disease classification using fuzzy-ID3 algorithm based on association function," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 11, no. 2, pp. 448–461, Jun. 2022, doi: 10.11591/ijai.v11.i2.pp448-461.
- [20] H. Kamel and M. Z. Abdullah, "Distributed denial of service attacks detection for software defined networks based on evolutionary decision tree model," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 4, pp. 2322–2330, Aug. 2022, doi: 10.11591/eei.v11i4.3835.
- [21] Z. A. Mohammed, M. N. Abdullah, and I. H. Al Hussaini, "Predicting incident duration based on machine learning methods," *Iraqi Journal of Computer, Communication, Control and System Engineering (IJCCCE)*, vol. 21, no. 1, pp. 1–15, Feb. 2021, doi: 10.33103/uot.ijccce.21.1.1.
- [22] S. Molla *et al.*, "A predictive analysis framework of heart disease using machine learning approaches," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 5, pp. 2705–2716, Oct. 2022, doi: 10.11591/eei.v11i5.3942.
- [23] B. Zhou *et al.*, "Online internet traffic monitoring system using spark streaming," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 47–56, Mar. 2018, doi: 10.26599/BDMA.2018.9020005.
- [24] OpenDaylight, "OpenDaylight (ODL)," *OpenDaylight*. <https://www.opendaylight.org/> (accessed Jan. 03, 2022).
- [25] ONOS, "Open network operating system (ONOS)," *ONOS*. <https://wiki.onosproject.org/> (accessed Jan. 03, 2022).
- [26] B. Zhou, J. Li, J. Wu, S. Guo, Y. Gu, and Z. Li, "Machine-learning-based online distributed denial-of-service attack detection using spark streaming," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6, doi: 10.1109/ICC.2018.8422327.
- [27] R. R. Sarra, A. M. Dinar, M. A. Mohammed, and K. H. Abdulkareem, "Enhanced heart disease prediction based on machine learning and χ^2 statistical optimal feature selection model," *Designs*, vol. 6, no. 5, pp. 1–12, Sep. 2022, doi: 10.3390/designs6050087.

BIOGRAPHIES OF AUTHORS



Sama Salam Samaan    is currently a Ph.D student and lecturer in the Department of Computer Engineering at the University of Technology in Baghdad, Iraq. She received her undergraduate degree in Information Engineering from this Department in 2005. In 2011, she earned her Master's Degree from Al-Nahrain University in Baghdad, Iraq. She published several papers in national journals, including Al-Nahrain Journal for Engineering Sciences, Al-Khwarizmi Engineering Journal, The Journal of Engineering, and Journal of Engineering and Sustainable Development. Her research activities concern big data, software defined networks (SDN), and machine learning. She can be contacted at email: sama.s.samaan@uotechnology.edu.iq.



Dr. Hassan Awheed Jeiad    received a B.Sc. degree in 1989 in Electronics and Communications Engineering from the University of Technology, Baghdad, Iraq. He received his Master's Degree in Communication Engineering from the University of Technology, Baghdad, Iraq, in 1999. Dr Hassan received his Ph.D in Computer Engineering in 2006 from the University of Technology, Baghdad, Iraq. Currently, he is an assistant professor in the Department of Computer Engineering at the University of Technology, Baghdad, Iraq. His research interests include computer architecture, microprocessors, computer networks, multimedia, adaptive systems, and information systems. He can be contacted at email: hassan.a.jeiad@uotechnology.edu.iq.