An Intelligent and Time-Efficient DDoS Identification Framework for Real-Time Enterprise Networks

SAD-F: Spark Based Anomaly Detection Framework

AWAIS AHMED, SUFIAN HAMEED, MUHAMMAD RAFI, and QUBLAI ALI MIRZA

ABSTRACT

Enterprise networks face a large number of threats that are managed and mitigated with a combination of proprietary and third-party security tools and services. However, the techniques and principles employed by the said tools, processes, and services are quite conventional. They lack the rapid evolution, as required to protect against modern, state-of-the-art threats faced, specifically, against distributed denial of service (DDoS) attacks. The lack of efficiency of a network is directly proportional to the number of applications and services it hosts, mainly to protect against external and internal threats. Moreover, the effectiveness of such security mechanisms relies on their independent and proactive approach, which is useful for known malware and their attack vectors, but become obsolete when there is a new malware or zero-day vulnerability is exploits.

This paper presents an intelligent, highly responsive, and scalable security framework for enterprise networks. The proposed framework incorporates Apache Spark Framework for security analytics. It accurately identifies anomalies related to DDoS attacks from real-time network traffic by using customized machine learning algorithms, meticulously trained against selected feature-set. Encouraging results are obtained when tested against different scenarios and bench-marked with the results achieved by related studies in similar scenarios.

INDEX TERMS Anomaly Detection, Apache Spark, Big Data Analytics, DDoS, Machine Learning, Malware, Network Security, Security Analytics

INTRODUCTION

THE heterogeneity in modern enterprise networks has enabled organizations to not only expand their business operations, but it has also provided an opportunity for businesses to efficiently use their skilled resources to optimize the effectiveness of every operation. The inclusion of a diverse set of devices and tools in regular enterprise networks has significantly increased the efficacy of the majority of related operations. This level of heterogeneity has also diminished physical and geographical boundaries generally faced by enterprises in the past [1].

The benefits of loosely coupled infrastructure to modern enterprises outweighed the regular network policies and compliance, which were followed by such enterprises to enhance performance and security [2]. However, securing such a

loosely coupled enterprise network is significantly complicated and requires a combination of tools and techniques that can detect and prevent proactively with minimum to no human dependency.

In the current security ecosystem, if an organization is infected, it takes approximately six months to identify that infection [3], and that too, if there is a predefined malicious behavior identified by intrusion detection systems (IDS). Generally, the technique of exploiting enterprise networks without triggering security alerts with malicious activities is used by attackers with an intent of distributed denial of service (DDoS) attacks [4].

To timely detect DDoS or similar attacks, enterprises heavily rely on tools based on security analytics techniques, which means identification based on; signatures, heuristics, and behaviors of known malware, along with their attack vectors [5]. This approach is quite effective and can protect against

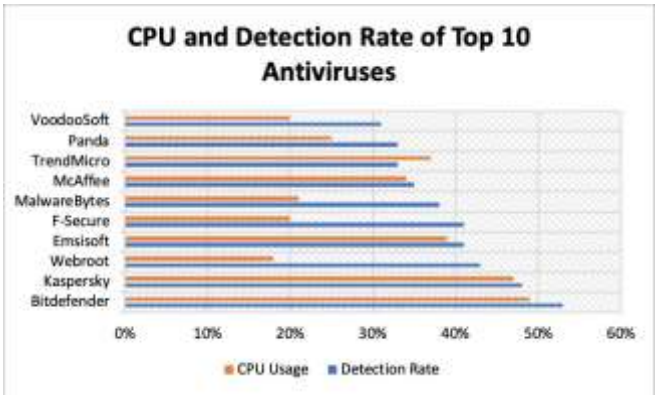

FIGURE 1. Antivirus Evaluation w.r.t. CPU Usage and Detection Rate [7]

numerous attacks, as the majority of malware released every day are variants of previously known malware [6]. Therefore, using signature, heuristics, and brief behavioral feature-set is quite effective. However, if the attack is originated by a new/unknown malware, exploiting a zero-day vulnerability in a network, can bypass the said security measures without getting detected.

The effectiveness of Intrusion detection systems, antiviruses, and other similar security tools, against unknown malware, has been practically evaluated several times, and the results are quite different from what is claimed by vendors of said tools. The lack of threat and anomaly detection is not the only issue. The network and computing resources consumed by the network security tools, along with the level of privilege they require, give them the highest priority infrastructure. According to a recent evaluation, enterprise antiviruses consume 45% of CPU resources while scanning for threats, and they can detect less than 60% of threats [7]. These statistics quite accurately depict the performance of security tools, along with the implications they can have on host networks.

The issues related to network security tools are not recent, but it is an on-going discussion in the security community. There are several solutions proposed by different researchers that quite accurately detect previously unknown malware using novel techniques—additionally, the implementation of machine learning (ML) algorithms, artificial neural networks. To accurately

detect malicious activity is quite successfully proposed and evaluated by several studies. In some selected studies, the level of accuracy for malware identification is 97% - 98% [8], which is significantly higher than the conventional security tools.

On the other hand, similar machine learning techniques used by other selected studies, as discussed in upcoming sections, to detect malicious activities in network logs of DDoS and other similar attacks accurately. The said approaches train ML algorithms against heuristics to accurately identify anomalies in network logs. The results of these techniques also surpass the identification statistics associated with conventional security tools.

The techniques, as mentioned earlier, proposed and self-evaluated by many studies depict promising results while accurately detecting malware in a single machine or anomalies in network logs, associated with DDoS. However, there are two critical aspects of enterprise network security; live anomaly detection and energy efficiency these were commonly overlooked.

A. MOTIVATION

There exists a significant gap in security mechanisms that are currently used and even proposed to fill the existing gaps in network security.

- It is required to have a solution that independently and accurately identifies threats and anomalies in live network traffic with a minimum amount of time.
- To propose and thoroughly evaluate a solution that incorporates the implementation of SPARK with selected machine learning algorithms to provide effective and energy efficient anomaly detection
- To prefer SPARK over HADOOP to differentiate the batch and stream processing, benefit from streaming by deploying the proposed framework in near real-time.

B. CONTRIBUTION

This paper initially presents a comparative analysis for streaming NetFlow traffic, including KDD, and UNSW-NB15, and later it is used for training supervised machine learning models for early anomaly detection. Moreover, UNSWNB-15 NetFlow dataset is then incorporate with the Hadoop Framework: Spark, the results of this integration are compared with the existing Hadoop Framework: MapReduce. This comparative analysis presents a significant improvement in real-time data analysis w.r.t. Time and accuracy, by accurately detecting anomalies in a very short interval.

The framework, SAD-F: Spark-based Automated Anomaly Detection Framework, proposed in this paper, is specifically designed to befit real-time network security scenarios, namely; anomaly detection in live enterprise networks within a very short interval and anomaly identification through analysis of stored indicators of compromise (IoC). These two primary features of the proposed framework are the key elements that address the gap mentioned above in the research in the area of network security. Moreover, to ensure that the proposed framework meets the performance parameters initially set, a thorough evaluation is performed. Additionally, the results of the evaluation used to compare with the results of existing solutions and benchmarking.

SAD-F is more appealing and close to real-time network security requirements, as compared to similar approaches for the following reasons:

- Although the frameworks, solutions, tools, and techniques currently used in conventional enterprise network environments, and proposed to fill the gaps in the existing security solutions, do address the issues. However, the current solutions lack the incorporation
  of entropy associated with the real-time network traffic, and the approach incorporating real-time entropy to achieve a high level of accuracy has a lasting and scalable impact.
- Enterprise networks have a requirement of immediate identification of anomalies that can lead to malicious entities in the network, specifically, the ones leading to DDoS. The incorporation of security analytics, supported by machine learning models, trained against customized feature-set, enable SAD-F to identify anomalies in a live network traffic stream in a responsive manner. The following are the machine learning algorithms that support the proposed framework.
  - K-Nearest Neighbour (KNN)
  - Naïve Bayes (NB)
  - Random Forest (RF)
  - Decision Tree (DT)
  - Support Vector Machine (SVM)
- The response time offered by SAD-F is benchmarked against existing solutions, which means that the effectiveness of the proposed framework is to check against accuracy and timeliness. These are the two parameters that complement each other while identifying the effectiveness of such a framework.
- The proposed framework is implemented using an opensource highly responsive analytics engine designed for large-scale data processing, known as Spark. Therefore, the results presented, discussed, and benchmarked later in this paper are not limited to the experimental dataset, it is also highly scalable to accommodate any form of packet stream in a real-time heterogeneous network.

The rest of the paper is structured as follows; Section II presents a discussion on the literature work. Section III presents an in-depth discussion of the proposed framework and how different aspects of the framework implemented. Section IV presents a discussion on the datasets used in the experiments and benchmarking. Section IV presents a step-by-step walk-through of testbed design. Section VI is comprises of experiments performed, analysis of results, along with the critical evaluation of the proposed framework and benchmarking based on related research results. Finally, section VII presents the conclusion of the study.

RELATED WORK

tion [12].

In this study [13] Romain et al. proposed a new framework; Hashdoop (extension to Hadoop MapReduce). Hashdoop suggests a solution over a drawback of Hadoop (Spatial and Temporal structure) when it comes to the point of splits of

network traffic over different nodes for distributed processing. Hashdoop splits traffic with a hash function to preserve traffic structure which leads to outstanding performance over the detection of network anomalies. Hashdoop evaluates with two anomaly detectors and fifteen traces of Internet backbone traffic captured between 2001 and 2013. Hashdoop with 6node cluster increased the throughput and enabled real-time detection of large analyzed traces. Hashdoop improves the overall accuracy of the anomalies detection process.

Hashdoop, experiments were conducted on MAWI archive dataset and more precisely, traffic captured between U.S and Japan on Sample point B and Sample point F. Authors noted performance trade-off that dividing traffic into many small splits can cause for adverse results because each split may contain insufficient traffic for the statistical analysis, they left this gap for future work or considered to be an open challenge.

Considering the above-stated gap as our research study, we are expecting a step ahead solution; instead of using the MapReduce technique we will utilize distributed computing stream processing for analysis of the same data and bears better results from an existing study.

In this paper [14] Juliette et al. stated in their study, the problem of unsupervised network anomaly has studied since the last decade. Many studies were proposed from time to time. Furthermore, they commented mostly unsupervised network anomaly detectors depended on clustering techniques. Clustering algorithms group similar flows on to the same cluster and the study presented a novel technique, unsupervised Network Anomaly Detectors Analysis (UNADA). UNADA took advantage of the distributed computing system to speed up their process.

Dividing the features space into subspaces which allows UNADA to run in multiple parallel DBSCANs and EA algorithms. Lastly, the authors claimed that their proposed technique UNADA's deployment in real-time stream tool 'Spark' can improve execution time by a factor of 13. Experiments conducted on grid' 5000 testbeds.

Extensive work has proposed in the field of SDN for DDoS mitigation. In paper [15] authors proposed a collabo-

In the literature, significant research efforts have been conducted over the years in an attempt to improve the quality/accuracy of network anomaly detection techniques.

There are some solutions published on network anomaly detection techniques [9], including studies incorporating machine learning in their solutions. There are also such studies that focus on anomaly detection in big-data related domains [10], [11]. In recent studies, many different techniques studied to solve the traffic classification problem. The majority of current classification approaches still rely on packet header based anomaly detection and protocol-based anomaly detecrative distributed denial of service (DDoS) attack mitigation scheme using a software-defined network. They design a secure controller to controller (C-to-C) protocol that allows software-defined network controllers exists in different autonomous systems (AS) to communicate and transfer attack information with each other securely. Further, as in extension, they proposed another scientific work [16]. They introduced three different deployment approaches, i.e., linear, central, and mesh, which enables efficient notification along the path of an ongoing attack and active filtering of traffic near the source of the attack, thus saving valuable time and network resources. In experiments, they showed SDN based collaborative scheme is capable of efficiently mitigating DDoS.

In this paper, [9] Ahmad et al. .presented in their work, one of the challenging modules of the intrusion detection system known as feature extraction, and presented comparative analysis/results for TCP based traffic. They implemented their system by using Apache Spark and Netmap. Authors claimed that their proposed system works well for a small organization. This system has designed on the top of the CAIDA NetFlow attack dataset. Limitations of their work overlooked by deploying the same system for large scale NetFlow traffic datasets, specifically for large organizations.

In this study [17] Milan et al. proposed a new performance benchmark based on standard security analysis algorithms for NetFlow data to show the suitability of the distributed stream processing system. Hadoop based system works only for batch processing and does not offer stream processing (real-time) analysis.

In order to overcome the limitations of Hadoop, three of the most used distributed systems taken for stream processing experiments such as Spark, Samza, and Storm and experiments conducted on the top of the CAIDA NetFlow dataset. Lastly, this study presents a performance benchmark chart to justify the outcome.

In paper [10] and [18] Pedro Crass et al. introduced a new framework for network analytic named BIG-DAMA, a Big Data Analytic framework (BDAF) for network traffic monitoring and analysis applications (NTMA). BIG-DAMA is a new and flexible framework, which is capable of analyzing and storing extensive data, including stream and batch. This study also implements multiple data analytic techniques for network security and anomaly detection. Different machine learning used. Authors claimed that they applied their technique with different types of attacks and benchmark outcomes. Experiments conducted on the top of the MAWI dataset. Further, the authors claimed that BIG-DAMA could speed up computation by a factor of 10 to existing solutions. Proposed system; BIG-DAMA can easily deployable in a cloud environment using virtualization techniques.

In study [19], the authors state that machine learning is a useful proactive/reactive analysis tool to detect anomalous activities if observed. They are considering machine learning as a suitable framework and Apache Spark data processing framework that can quickly perform processing tasks on substantial data sets. In results, the authors presented experimental results on the UNSW-NB15 dataset and evaluated in terms of detection accuracy, building time, and prediction time. This paper uses mainly four classification algorithms, namely: SVM, Naïve Bayes, Decision Tree, and Random Forest. This paper aims to reduce training and detection time. This paper also aims to develop a detection algorithm to classify the incoming network traffic into one of the nine attack categories.

In this paper [20] authors stated that with machine learning techniques for network traffic anomalies can be detected with reasonable accuracy. They continued that most of the previous work has focused on detecting anomalies using traditional machine learning frameworks. However, this paper used an Apache Spark framework to cope with the high volume. This paper investigated the feasibility of Apache Spark and to classify different attacks. In this paper, they applied traditional machine learning algorithms namely: Multinomial Logistic Regression, Decision Tree, Random Forest, Multilayer perceptron, and Naïve Bayes using a dataset presented by MAWILab and classify 15 different attack types. Further, this paper investigates the efficiency of Apache Spark in terms of accuracy and speed with different settings of spark. They investigated the performance of the detection system by varying executor cores and memory size. In the results of the future work, authors would like to extend their work using spark streaming and aim for real-time detection of network traffic anomalies.

In this study [21], authors proposed an intelligent framework based on the RF- Random Forest Classifier for quick and precise anomaly-based intrusion detection system with the help of high volume data processing framework, i.e. Apache Spark. This paper used the NSL-KDD train and test dataset from the University of New Brunswick, respectively for training and testing the model. In results, this paper presented high accuracy, low false alarm, low false negative, and high precision which is promising for applying in anomaly detection systems.

A recent study [22] proposed an efficient spark-based anomaly detection by working on UNSW-NB15 dataset. Authors stated that proposed approach gives better accuracy even using a small subset of features of the UNSW-NB15.

In this survey paper [23] authors stated nowadays anomaly detection in high-dimensional data is becoming a fundamental research problem and to addressed this mentioned gap and to understand the core problem it is necessary to identify the unique challenges in perspective anomaly detection with both high dimensionality and big data problems. Author's contributions of this survey are such as: 1.Reviewed theoretical background of existing work 2.Identified unique challenges. 3. Reviewed techniques or algorithms of anomaly detection which addressed the problems of high dimensionality and big data. 4. Reviewed the big data tools.

A recent study [24] proposed a novel approach for real time DDoS detection. Authors stated that traditional frameworks are not capable of processing live traffic in the big data environment. To cater the stated gap a streaming-based distributed and real-time DDoS detection system called S-DDoS were proposed. S-DDoS system uses K-Means clustering algorithm to recognize the DDoS attack traffic in real-time. The model were designed on the Apache Hadoop framework. The results show that the proposed S-DDoS detection system efficiently detects the DDoS attack from network traffic flows with higher detection accuracy 98%.

Readers are encouraged to refer upcoming both tables 1 and II for a comparative overview of related work with our proposed work. Table II is an excellent start to find research gaps in existing as it lists five technical questions about the related work: only a few and most relevant papers listed in the literature table II.

After going through from related work, we came up with a system and expects efficiency in terms of the time of computation and accurate enough from the existing system. To achieve the objective, we deployed two of the streaming dataset including KDD99 and UNSW-NB15, and predicting accurate results for anomalies such as DoS, DDoS, IP Scanning, Port Scanning, etc. This paper investigates existing and proposed work and benchmark the difference. Lastly, this paper also lists literature work on the Netflow data preprocessing techniques.

SPARK BASED ANOMALY DETECTION

FRAMEWORK - (SAD-F)

The purpose of this study is to contribute and to fill the research gap as highlighted from [10], [25], [26] discussed above in section Problem Statement. The follow-up sections describe the subsections, including the proposed framework, dataset selection, data preprocessing techniques, model selection, results, conclusion, and future work.

A. PROPOSED FRAMEWORK

In this study, we propose a successor of our previously purposed framework in base study [25], [26], which comprise of five major phases implemented as separate components including:

1) Netflow traffic capturing server and or NetFlow traffic selector
2) Preprocessing of Netflow data
3) Spark cluster simulation phase
4) Anomaly detector
5) Result in the form of graph or notification to the concerned body

.PCAP, or CSV) this file will pass over the "Preprocessing and Detection stage", which further discussed in the upcoming subsection.

Live traffic may contain some of the predefined filters to capture required network traffic. For example, parameters could be packet size, no of frames and desired protocol filter. As admin finalized the parameter-tuning/ filter-setting capturing tool begins to capture as depicted in figure-3. Along with the traffic capturing process that file will be logged and will be transferred further for the preprocessing and detection process, which is further discussed below in subsection. Example filters as shown in figure-4 as reference these filters may help future researchers to begin with live capturing.

C. DATA PREPROCESSING TECHNIQUES

The hard part of the anomaly detection process is the preprocessing of network traces and the data preprocessing techniques is an essential step in anomaly detection, and review shows that preprocessing needs domain experts for better features selection which leads to improvements in results. The review also finds that many studies limit their preprocessing steps and do not discuss in details in their work [10]. As network traffic files will arrive at this stage, the preprocessing and anomaly detection process will begin. Data preprocessing is a must step in all knowledge discovery tasks from which preprocessing of network-based intrusion detection took 50 percent [12] efforts of the overall process which produced a better classification of network traffic as normal or anomalous. Removing strong co-relate, irrelevant, and redundant features also improve the detection rate for learning-based algorithms. This study focuses on the Netflow preprocessing stage, and for that, we designed a novel approach for preprocessing. We studied various formal process models that proposed for knowledge discovery and data

As per limitations discussed, we upgraded framework 2 adding stream processing by spark and removing counterbased algorithm, which was supposed to find listed anomalies. However, this proposed SAD-Framework mainly focuses on finding a nomalies f rom k nown a s w ell a s unknown data with preferred low-level hardware, also known as commodity hardware. One can see the difference between proposed framework components and previous work 2 and 3.

B. NETFLOW TRAFFIC CAPTURING SERVER AND OR NETFLOW TRAFFIC SELECTOR

In this phase, one may suppose to pass the NetFlow dataset directly or captures live traffic with the help of network capturing tool Wireshark with predefined filters to filter network traffic.

This framework is flexible and offers an admin perspective mining (KDDM) as discussed in [27]. After thoroughly reviewing studies [12], [27], we found standard preprocessing steps include dataset creation, data cleaning, integration, feature

construction to derive new higher-level features, feature selection to choose the optimal subset of relevant features, reduction. The most important steps for network intrusion deduction systems briefly discussed below:

- Dataset creation: Involves identifying representative network traffic for training and testing. These datasets should be labelled, indicating whether the connection is normal or anomalous. Labelling network traffic can be a very time consuming and difficult task.
- Feature construction: It aims to create needed additional features with better discriminating ability than the initial feature set. This step can bring significant improvement to machine learning algorithms. Features

web interface through which admin can tune the capturing process by parameter tuning for desired live traffic. However, at this stage, we are not designing the admin portal; instead, we will use passive NetFlow data to the file transfer stage. After loading the capture file, file format could be (.CAP, can also created manually, or by using data mining methods such as sequence analysis, association mining, and frequent episode mining

S.No	Paper Title	Publication Year
1	Hadoop-based live DDoS detection framework	2018
2	Efficacy of live ddos detection with hadoop	2016
3	Network security and anomaly detection with Big-DAMA, a big data analytics framework	2017
4	Performance evaluation of intrusion detection based on machine learning using Apache Spark	2018
5	Network Traffic Anomaly Detection based on Apache Spark	2019
6	Network Anomaly Detection by Means of Machine Learning: Random Forest Approach with Apache Spark	2018
7	Network intrusion detection in big dataset using Spark	2018
8	A comprehensive survey of anomaly detection techniques for high dimensional big data	2020
9	S-DDoS: Apache spark based real-time DDoS detection system	2020
10	An efficient spark-based network anomaly detection	2020

TABLE 1. Listing of Referenced Papers

S.No	Does paper propose any framework to present their work?	Repeatedly used Dataset(s): 1)UNW-NB15 2)NSL-KDD 3)MAWILAB 4)Others(please specify)	Machine Learning Algorithms: 1)RF 2)DT 3)SVM 4)NB 5)LR 6)Others(please specify)	Mostly evaluation techniques used: 1)Model accuracy 2)Building time 3)Prediction time 4)Tweaking of executor cores and memory size 5)Low false alarm 6)High precision	Mostly highlighted future work: 1)Less training/detection time 2)Live traffic 3)Spark streaming	Gaps filled by SADF or comparison with SAD-F
1	HADEC	N/A	Counter Based Algorithm	1	1 and 2	1 and 2
2	N/A	N/A	HADEC Reducer Job and Counter Based Algorithm	1	1 and 2	1 and 2
3	Big-DAMA	3	1,3,4 CART and MLP	1 and 3	1 and 2	1 and 2
4	Apache Spark	3	1-2,4-5 and MLP	1 and 4	1 and 3	1 and 2
5	Apache Spark	2	1	5 and 6	N/A	1
6	Apache Spark	1	1	1 and 3	1	1
7	Apache Spark	1	1-4	1-3	1 and 2	1 and 3

TABLE 2. Literature Work

- Reduction: It is a commonly used technique to decrease the dimensionality of the dataset by discarding any redundant or irrelevant features. This process of feature optimization is called feature selection, and is commonly used to alleviate “the curse of dimensionality”. Data reduction can also be achieved with feature extraction, which transforms the original feature set into a reduced number of new features. Principal component analysis (PCA) is a conventional linear method used for data reduction.

Preprocessing converts network traffic into a series of observations, where each observation represents a feature vector. Observations optionally labeled with its classes, such as “normal” or “anomalous.” These feature vectors are then suitable as input to data mining or machine learning algorithms. Machine learning is the use of algorithms that evolve according to the labeled data instances (observations) provided to it. The algorithms can generalize from these observations, hence allowing future observations to be automatically classified. Machine learning is widely used in anomaly-based NIDs with examples including [28] and the Principal Component Classifier by [29]

After a successful process, the preprocessed file will transfer further to spark cluster manager for further classification process, and the file will test against the pre-trained model.

Results logged for the further notification process, which further discussed in detail in the upcoming section.

D. ANOMALY DETECTION TECHNIQUES: NETWORK INTRUSION DETECTION SYSTEMS

To the best of the literature review, we are concluding that intrusion prevention techniques are imperfect, so monitoring for security compromises is required, and it is an essential role of network intrusion detection systems (IDSs). Our proposed NIDs aims to detect malicious activity in near *realtime and raise an alert. Anomaly detection in intrusion based detection systems provides a way to detect several attacks, which results in network failures in real-time or be the basis for future mishaps. Curing anomaly detection at early stages is preferable due to many factors including cost, the response time of system and customer satisfaction.

In this paper, we are utilizing the following listed supervised techniques for anomaly detection.

- k-nearest neighbors algorithm
- Naive bayes
- Random forest
- Decision Tree
- Support Vector Machine

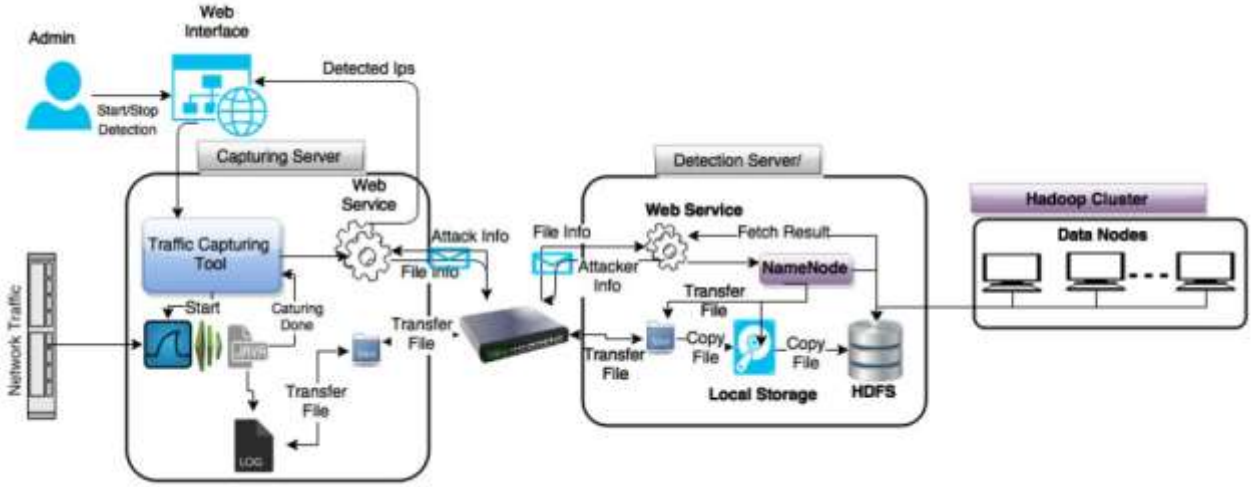

FIGURE 2. HADEC: Hadoop based Live DDoS Detection framework;figure reused from scientific paper [25], [26]

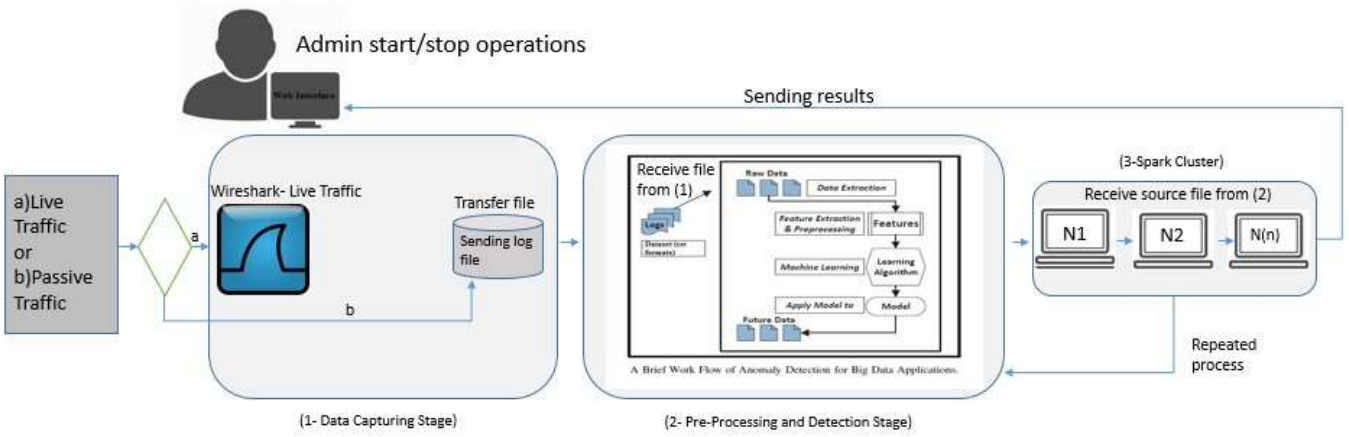

FIGURE 3. SAD-F: Spark Based Anomaly Detection Framework

In this section, we are not supposed to discuss models details from the perspective of machine learning, although interested readers referred to these [30], [31]

E. SPARK CLUSTER SIMULATION PHASE

According to our architecture, after preprocessing of the dataset, Spark comes in action to deliver the requested work. In this paper, we are considering that the standalone spark cluster consists of five worker nodes, including one of them working as a master node. Apache Spark is a cluster computing platform as stated in [19], [32] designed to be as fast as near real-time response rate, Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Its speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory. Still, the system is also more efficient than MapReduce for complex applications running on disk. Apache Spark also designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads

in the same engine, Spark makes it easy and cost-effective to combine different processing types, which is often necessary for production data analysis pipelines. Besides, Apache spark reduces the management burden of maintaining separate tools. Spark also supports the collection of different programming languages, including R, Python, Scala, etc. Refer to the following subsection for configuration guidelines for getting started with Spark.

```
TCP (SYN)
17956  45.406170  10.12.32.1 -> 10.12.32.101 TCP 119 [TCP
Retransmission] 0 > 480 [SYN] Seq=0 Win=10000 Len=43 MSS=
1452 SACK_PERM=1 TSval=422940867 TSecr=0 WS=32

HTTP
46737 2641.808087 10.12.32.1 -> 10.12.32.101 HTTP 653 GET
/posts/17076163/ivc/dddc?_=1432840178190 HTTP/1.1

UDP
139875 138.04015 10.12.32.1 -> 10.12.32.101 UDP 50
Src port: 55348  Dst port: http

ICMP
229883 2658.8827  10.12.32.1 ->  10.12.32.1O1 ICMP 42
Echo (ping) request  id=0x0001,seq=11157/38187, ttl=63
reply in 229884)
```

FIGURE 4. Wireshark (T-Shark)- Example filters for Live Capturing;"figure reused from scientific paper" [25], [26]

1) Spark Cluster Configuration

Spark cluster is a network of computers. A cluster consists of a master and n* slaves. (N is limited)

Master: Master is the leader of all servers that monitors how slaves are working. It divides the task and takes care of rest.

Slaves: These are the computers that receive a job from the master node and perform a job. They are responsible for processing chunks of your massive datasets following the MapReduce paradigm. A computer can be a master and slave at the same time.

Spark offers/supports three types of clusters, including standalone, Yarn, and Mesos. Standalone: Spark is responsible for managing its cluster.

Yarn: Spark uses Hadoop's Yarn resource manager. Mesos: Spark Apache dedicated resource manager.

F. RESULT AND NOTIFICATION

In this stage, our proposed framework suggests generating a notification to avoid future vulnerabilities and to mitigate from current attacks. On successful completion of learning tasks, the final result file transferred, and the admin (server) gets notified for anomalies (for this, we are using a commandline based interface to track all interaction). Figure-3 presents a complete illustration of our proposed framework.

IV. SAD-F: SCIENTIFIC DATASETS

Network anomaly detection is a challenging task due to the dynamic nature of NetFlow traffic. The paper focuses on the general analytics/techniques to detect anomalies in the network. The study exploring the following datasets

- KDD Cup 99
- UNSW-NB15

All of the mentioned datasets are in .pcap format provided by sources [33], [34]. The following further subsection describes the detailed discussion of the listed dataset individually.

A. DATASETS FOR SYSTEM EVALUATION

This study does not require any specific traffic data because, in this paper, we propose a general framework. But we recommend these Netflow datasets KDD-CUP 99 and UNSWNB15. Even though KDD-CUP 99 is not entirely usable anymore because it lacks most of the new types of attacks, we took this dataset for testing and proof of concept of our deployed framework.

- KDD Cup 99: Since 1999, KDD99 noticed to be the widely used dataset for evaluation of anomaly detection methods [35]–[37]. This dataset is prepared by [35] and is built based on the data captured in DARPA'98 IDS evaluation program [35]. "This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model that distinguished between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment." The dataset's attacks fall into four categories:
  - Denial of Service Attack (DoS): DoS is one the attack type in which the intruder makes some computing or memory resource too busy to handle legitimate requests, or denies legitimate users access to a machine.

  - User to Root Attack (U2R): It is a class of exploit in which the attacker starts with access to a standard user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and can exploit some vulnerability to gain root access to the system.
  - Remote to Local Attack (R2L): It occurs when an attacker who can send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine.
  - Probing Attack: Probing is an attempt to gather information about a network of computers for the apparent purpose of circumventing its security controls.
- UNSW-NB15: For the evaluation of the performance and effectiveness of the network intrusion detection system, we require a comprehensive dataset that contains both normal and abnormal behaviors. Much research work has done using older benchmark data sets like KDDCUP 99 and NSLKDD, but these data sets do not offer realistic output performance. The reason is that KDD CUP 99 has lots of redundant and missing records in the training set. So these

datasets are not a comprehensive representation of the modern low footprint attack environment, which concludes that we need a dataset that fulfills the requirements.

The UNSW-NB 15 data set created by utilizing an "IXIA PerfectStorm tool" to extract a hybrid of modern and contemporary attack activities of network traffic. A tcpdump tool used to capture "100 GB" of raw network traffic (pcap files). Each pcap file contains "1000 MB" to make the analysis of packets easier.

UNSW-NB15 dataset is available in comma-separated values(CSV) file format.This data set contains 2,540,044 records, which stored in four CSV files. Moreover, apart from this data set was divided into a training set and a testing set. The training set involved 175,341 records, while the testing set contained 82,332 records with all different nine types of attack and normal records. There are 45-49 attributes or features with 10 class values in this dataset. All records divided into two major categories of the records - normal and attack. Furthermore, the attack categories subdivided into nine categories of attack types. [38].

- Fuzzers: It is an attack in which the attacker attempts to discover security loopholes in an application, operating system, or a network by feeding it with the massive inputting of random data to make it crash.
- Worms: It is an attack in which the attacker replicates itself to spread on other computers. It often utilizes a computer network to spread itself, depending on the security failures of the target computer used to access it.
- Reconnaissance: This attack category is also known as a probe, and it is an attack that gathers information about a computer network to evade its security controls.
- Analysis: It is a type of variety intrusions that penetrate the web applications via ports (e.g., port scans), emails (e.g., spam), and web scripts (e.g., HTML files).
- Backdoors: It is a technique of bypassing a stealthy normal authentication, securing unauthorized remote access to a device, locating the entrance to plain text, as it struggles to continue unobserved.
- DoS: It is an intrusion which disrupts the computer resources via memory to cause excessive business, to prevent authorized requests from accessing a device.
- Exploits: It is a sequence of instructions that takes advantage of a glitch, bug, or vulnerability, causing an unintentional or unsuspected behavior on a host or a network.
- Generic: It is a technique that establishes against every block-cipher using a hash function to cause a collision without respect to the configuration of the block cipher.
- Shellcode: It is a malware in which the attacker penetrates a slight piece of code starting from a shell to control the compromised machine.

Furthermore, research studies [6], [38] divided this dataset in to six different categories as follows:

- Flow features: This group includes the identifier attributes between hosts, such as client-to-serve or server-to-client.
- Basic features: this category involves the attributes that represent protocols connections.
- Content features: this group encapsulates the attributes of TCP/IP and contains some attributes of HTTP services.
- Time features: this category contains the attributes of time, for example, arrival time between packets, start/end packet time, and round trip time of TCP protocol.
- Additional generated features: it includes generalpurpose features and connection features.
- Labelled Features: this group represents the label of each record.

From the selected datasets, we focus on a specific group of attacks, as discussed above in subsection IV. Considered algorithms trained to detect each of these attack types independently and in parallel, in the same fashion as in [10], [39]. As a result, each detection approach can detect the occurrence of an attack and also classify its nature.

SNo	Feature Name	Feature Description
1	srcip	Source IP address.
2	sport	Source port number.
3	dstip	Destinations IP address.
4	dsport	Destination port number.
5	proto	Protocol type, such as TCP, UDP.

TABLE 3. Flow Features; Table reproduced from scientific papers [34] and [6], [38]

SNo	Feature Name	Feature Description
6	state	The states and its dependent protocol e.g., CON.
7	dur	total duration.
8	sbytes	Source to destination bytes.
9	dbytes	Destination to source bytes.
10	sttl	Source to destination time to live.
11	dttl	Destination to source time to live.
12	sloss	Source packets retransmitted or dropped.
13	dloss	Destination packets retransmitted or dropped.
14	service	Such as http, ftp, smtp, ssh, dns and ftpdata.
15	sload	Source bits per second.
16	dload	Destination bits per second.
17	spkts	Source to destination packet count.
18	dpkts	Destination to source packet count.

TABLE 4. Basic Features; Table reproduced from scientific papers [34] and [6], [38]

SNo	Feature Name	Feature Description
19	swin	Source TCP window advertisement value.
20	dwin	Destination TCP window advertisement value.
21	Stcpb	Source TCP base sequence number.
22	dtcpb	Destination TCP base sequence number.
23	smeansz	Mean of the packet size transmitted by the srcip.
24	dmeansz	Mean of the packet size transmitted by the dstip.
25	transdepth	The connection of http request or response transaction.
26	resbdylen	The content size of the data transferred from http

TABLE 5. Content Features;Table reproduced from scientific papers [34] and [6], [38]

SNo	Feature Name	Feature Description
27	sjit	Source jitter.
28	djit	Destination jitter.
29	stime	start time.
30	ltime	last time.
31	sintpkt	Source inter-packet arrival time.
32	dintpkt	Destination inter-packet arrival time.
33	tcprtt	Setup round-trip time, the sum of 'synack' and 'ackdat'.
34	synack	The time between the SYN and the SYN-ACK packets.
35	ackdat	The time between the SYN-ACK and the ACK packets.
36	ismipports	If srcip (1) = dstip (3) and sport (2) = dsport (4), assign 1 else 0.

TABLE 6. Time Features; Table reproduced from scientific papers [34] and[6], [38]

SNo	Feature Name	Feature Description
37	ctstatettl	No. of each state (6) according to values of sttl (10) and dttl (11).
38	ctflwhttpmthd	No. of methods such as Get and Post in http service.
39	isftplogin	If the ftp session is accessed by user and password then 1 else 0.
40	ctftpcmd	No of flows that has a command in ftp session.
41	ctsrvsrc	No. of rows of the same service (14) and srcip (1) in 100 rows.
42	ctsrvdst	No. of rows of the same service (14) and dstip (3) in 100 rows.
43	ctdstltm	No. of rows of the same dstip (3) in 100 rows.
44	ctsrcltm	No. of rows of the srcip (1) in 100 rows.
45	ctsrcdportltm	No of rows of the same srcip (1) and the dsport (4) in 100 rows.
46	ctdstsportltm	No of rows of the same dstip (3) and the sport (2) in 100 rows.
47	ctdstsrcltm	No of rows of the same srcip (1) and the dstip (3) in 100 records.

TABLE 7. Additional generated Features;Table reproduced from scientific papers [34] and [6], [38]

SNo	Feature Name	Feature Description
48	Attackcat	The name of each attack category.
49	Label	0 for normal and 1 for attack records.

TABLE 8. Labelled Features;Table reproduced from scientific papers [34] and [6], [38]

V. TESTBED DESIGN STEPS

This section highlights and discusses the implementation design steps of the proposed framework. The following subsections discuss overall components that need to configure before the execution of the system.

A. TESTBED SETUP

We design and test our framework on two different testbeds we named one as "low end" testbed and "high end" testbed. Both have different specifications in terms of hardware.

For low-end testbed, we configured three worker nodes and one master node. Each node is physically independent of each other. Each system contains 8GB of memory and 500 GB of Hard Disk and 2.7 GHz of processor. This cluster setup collectively consists of 10*16 GB of memory, 10*500 GB of space, and 3*3.5 GHz of the processor as per the previous study's configuration [10].

For high-end testbed, we configured two worker nodes and one master node. Each node is physically independent of each other. Each system contains 16GB of memory and 1TB of Hard Disk and 3.5 GHz of processor. This cluster setup collectively consists of 2*16 GB of memory, 2*1 TB of space, and 3*3.5 GHz of processor.

B. NETFLOW TRAFFIC SELECTOR

As shown in figure-3, our proposed framework is flexible with either live traffic or passive traffic selector. At the current stage, we only used a passive traffic selector. At this step, the system requires the required traffic file after that system will transfer that file to the log file for further processing. The following two options are available as traffic selectors.

- Live network traffic capturing server
- Passive traffic

C. LOG SERVER

After receiving the file at this stage log server will keep that file for future need; if there is any discrepancy in results, admin/network administrator can do manual work to take respective actions.

D. PREPROCESSING FILE

We designed a preprocessing function that is efficient, and it is designed for the specific NetFlow dataset. However, it is quite easy to change its parameters to fit with a new NetFlow dataset; the following are the few steps we considered while designing the preprocessing module.

- Data file splitting by timestamp
- Calculating hash of features at run-time
- Converting features into OneHotEncoding at run-time

Before beginning for further network traffic classification, we split targeted data by time-slot to reduce the size of the file. For example, a one hour network trace file split into multiple 10-Seconds files. The general command to split network trace file based on time-slot. "editcap -i file-splittime filename.pcap file-split-time.pcap"

For preprocessing, we designed our own methodology which aims to read the data file and load it into data-frame. Then data-frame were read individually by features set. By Deep feature inspection we noticed data type of each feature after that we used appropriate conversion method for each feature refer to tables [3, 4, 5, 6, 7 and 8] for detailed dataset feature set.

E. FILE TRANSFER TO SPARK CLUSTER

After taking all the necessary steps, the preprocessed file arrives at the spark phase on which the system distributes the file to the worker node, which is one of the spark's core job.

F. MODEL SELECTION

We implemented five supervised learning algorithms KNN, RF, DT, NB and SVM to train testbeds as referenced in [10].

G. MODEL TESTING

We used the training model as our base model to classify whether the incoming network traffic to the server or main node is normal or an attack.

H. ADMIN PANEL

Whoever will deploy our framework to improve their network operation in terms of DDoS detection; they can design their admin panel as per requirements. The framework is capable of further integration. At this stage, we used Ubuntu command prompt system to tackle admin related tasks like start/stop the operations.

VI. EXPERIMENTS AND RESULT ANALYSIS

The framework is independent of Netflow dataset selection. However, we explored and practically implemented the UNSW-NB-15 dataset is publicly available at [34] since 2015. UNSW-NB-15 is the most suitable and welldescribed dataset available for researchers to work to the best of our knowledge. Apache Spark Framework: Apache Spark-based anomaly detection framework and MLib libraries used. Five well-known machine learning algorithms used, namely [KNearestNeighbour, NaïveBayes, DecisionTree, SuportVectorMachine, and RandomForest], used for performance evaluation. Recently, similar work has been proposed by [19]. One may compare the significant result difference between our proposed work with this work.

A. PERFORMANCE EVALUATION:

The overall performance of SAD-F depends upon the time taken to capture and transfer the log file to the testbed in real-time mode. The system depends upon the time taken to read a chunk of traffic volume and time taken to classify each chunk and moves forward for the next chunk of traffic as in upcoming tables [1-3], we presented two variations of chunk size i-e 300 and 1000. It also depends upon the time taken by a specific classifier to detect chunk and generate false alarm + negligible overhead of the system.

For performance evaluations, we benchmark our results, which focuses on the selection of classifier, file size, the total volume of traffic, false alarm rate, chunk size of traffic volume, and detection time.

Figure-5 depicts the total time taken to capture and transfer real traffic (using Wireshark T-Shark library) on our deployed primary node. Traffic capturing time is almost linear to the file size; as the file size is increasing, it is also increasing. It took almost 20 seconds to capture a file size of 10MB while it took 420 seconds to capture 1 GB file. On the other hand, it is clear that as the file size is increasing, the volume of traffic is also increasing, so both relations are linear co-related. It shows a definite improvement in throughput with the increase in file size and volume of traffic.

At this stage, these calculations are proof of concept while we did not continue with real-time strategy. However, we are reading a chunk of data and simulating like a real-time (near real-time) these measurements will be entirely useful for the future researcher.

Figure-6 list dataset volume w.rt. to file size and packets count.

1) Comparison of preprocessing, training and testing cost on local machine vs. SAD-Framework

Scientific reading presented in this Table 9 taken from local machine (8GB of RAM and 2.50 GHz of processor) while reading presented in Table 11 and 13 are taken from SADFramework. Measurements of KNN is not presented in above tables 11 and 13 because kNN is not supported by SPARK distributed architecture [40]. By analysing readings from above presented tables 9, 11 and 13 which clearly shows

TABLE 9.

Model	Train/Test file size and Traffic volume	Overall Time	Accuracy	SE_T	SE_A
DT	150 MB 40 MB (999999, 300000)	90	94.40	±1.074	±0.758
RF	150 MB 40 MB (999999, 300000)	360	92.98	±1.747	±0.760
NB	150 MB 40 MB (999999, 300000)	204	72.6	±1.945	±0.447
SVM	150 MB 40 MB (500000, 200000)	14400	87.62	±5.145	±0.756

Relationship of train/test time (in seconds) and Accuracy of model in percentage on a local machine

TABLE 10.

Model	Accuracy or Detection Rate	Precision	False Positive Rate	True Positive Rate	True Negative Rate	False Negative Rate	F_{score}
kNN	0.9206	0.9471	0.0528	0.9654	0.5258	0.0345	0.9562
DT	0.9407	0.9697	0.0302	0.9674	0.4937	0.0325	0.9685
RF	0.9298	0.9474	0.0525	0.9790	0.2493	0.0209	0.9629
NB	0.7269	0.8698	0.1301	0.7903	0.4103	0.2096	0.8282
SVM	0.82621	0.8648	0.1351	0.9069	0.5956	0.0930	0.8854

Measurements from Confusion Matrix

Model	Train/Test file size and Traffic volume	Overall Time	Accuracy	SE_T	SE_A
kNN	150 MB 40 MB (999999, 300000)	12605	92	±2.265	±1.235
DT	150 MB 40 MB (999999, 300000)	6565	94.40	±3.185	±1.050
RF	150 MB 40 MB (999999, 300000)	6549	92.98	±3.417	±1.050
NB	150 MB 40 MB (999999, 300000)	6536	72.6	±1.586	±0.448
SVM	150 MB 40 MB (500000, 200000)	71317	87.62	±1.732	±0.756

TABLE 11. Relationship of train/test time (in seconds) and Accuracy of model in percentage on a low end testbed

Model	Accuracy or Detection Rate	Precision	False Positive Rate	True Positive Rate	True Negative Rate	False Negative Rate	F_{score}
DT	0.9407	0.9697	0.0302	0.9674	0.4937	0.0325	0.9685
RF	0.9298	0.9474	0.0525	0.9790	0.2493	0.0209	0.9629
NB	0.7269	0.8698	0.1301	0.7903	0.4103	0.2096	0.8282
SVM	0.82621	0.8648	0.1351	0.9069	0.5956	0.0930	0.8854

TABLE 12. Measurements from Confusion Matrix

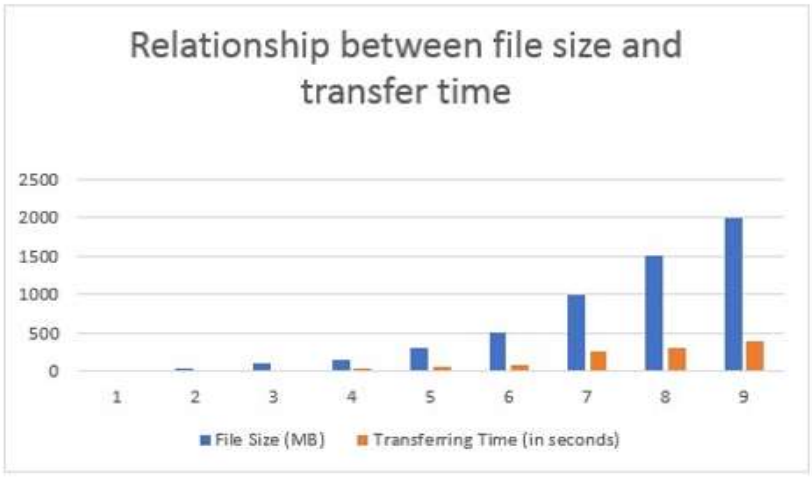

FIGURE 5. Capture and transfer time of a log file

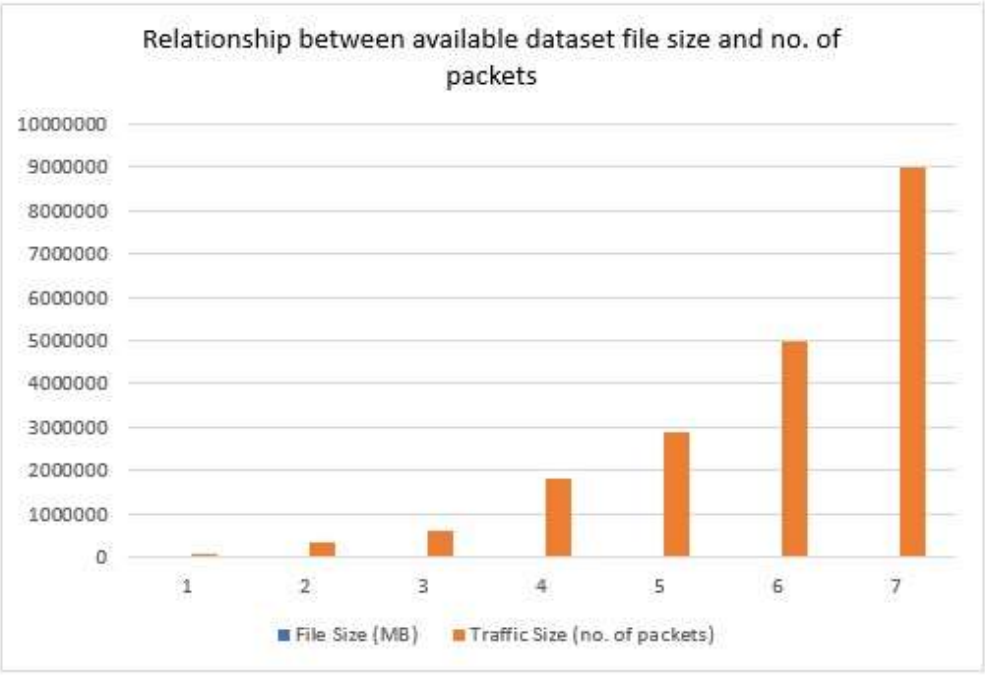

FIGURE 6. Relationship of Dataset Size and Traffic Volume

the improvement. Here we conclude our proof of work that our deployed frameworks. The next section focuses on actual error calculations are as per standard calculations [?]. scientific reading taken from SAD-F on both testbeds individually by simulating near real-time traffic. Calculated standard

Model	Train/Test file size and Traffic volume	Overall Time	Accuracy	SE_T	SE_A
DT	150 MB 40 MB (999999, 300000)	32	92.23	±0.981	±0.564
RF	150 MB 40 MB (999999, 300000)	200	91.68	±1.236	±0.525
NB	150 MB 40 MB (999999, 300000)	104	78.54	±1.826	±0.671
SVM	150 MB 40 MB (500000, 200000)	2890	86.52	±3.235	±0.721

TABLE 13. Relationship of train/test time (in seconds) and Accuracy of model in percentage on a high-end testbed

Model	Accuracy or Detection Rate	Precision	False Positive Rate	True Positive Rate	True Negative Rate	False Negative Rate	F*score*
DT	0.9223	0.9645	0.0354	0.9508	0.5823	0.0491	0.9576
RF	0.9168	0.9465	0.0534	0.9645	0.3274	0.0354	0.9554
NB	0.7854	0.8791	0.1208	0.8607	0.4103	0.1392	0.8698
SVM	0.8652	0.8716	0.1283	0.9594	0.5956	0.0405	0.9134

TABLE 14. Measurements from Confusion Matrix

B. NEAR REAL-TIME TRAFFIC CLASSIFICATION BY SAD-F:

Our system can simulate traffic in the form of single or many rows (a.k.a instances and a chunk of traffic volume. After that, our program will pass this traffic to classifier/function to classify the traffic behavior and generate a false alarm if the system detected as any malware available in the selected chunk of data and repeats the process to select another chunk of volume till the end of the file.

NetFlow Traffic can be real-time traffic refer to figure-5. However, it needs an extra effort in terms of time for preprocessing to include all the required features set for proper working and to increase accuracy and time taken by logging and transferring such traffic volume to the testbed/deployed framework.

We evaluated the performance of our framework based on different sizes of the log file, as in figure-6. For testbed evaluation, we individually noted spark file loading and distributing time, file preprocessing time, detection time of the algorithm, and total time is taken by spark cluster to complete the experiment. For scientific reasons, we calculated an average of three measurements for individual experiments. The following paragraphs show low-end testbed and highend testbed results, respectively.

C. LOW END TESTBED RESULT CHARTS

Packet Count	DT	RF	NB	SVM
100K	1.2008	0.9705	0.7536	153.2593
450K	1.5635	1.5776	5.9583	543.8486
1000K	3.3746	2.8163	21.0618	946.4972
1800K	179.453	66.4375	38.9206	1176.6666
2500K	428.3554	376.1895	442.9408	2008.8368
3400K	643.8411	613.1385	850.6137	2900.5085

TABLE 15. Detection time taken by different classifier w.r.t Packet Size

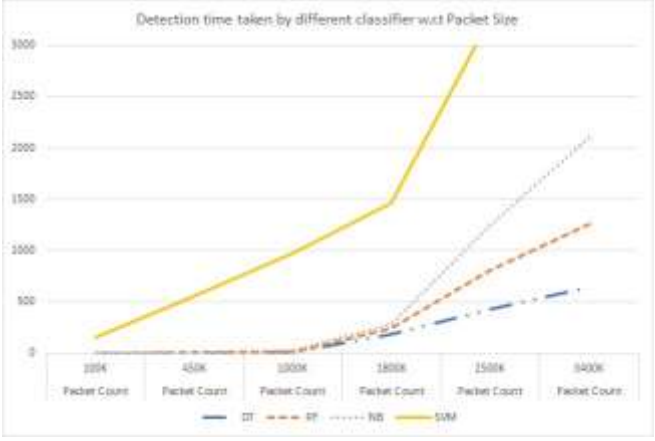

FIGURE 7. Detection time taken by different classifier w.r.t Packet Size

Table 15 shows the average detection time taken by different detection models, namely [DT: Decision Tree, RF: Random Forest, NB: Naive Bayes and SVM: Support Vector Machine]. We noticed that NB performed better than all three remaining models and took 0.6 and 2.5 seconds for 100K and 450K packets count, respectively, and RF took 4.46 seconds for 1000K packets count. We also noticed that RF performed well w.r.t all three models and took a maximum of 335 seconds for 3400K packets count while SVM is taking maximum time w.r.t all three models. SVM took almost 90 seconds for a 10MB log file, which consist of 100K packets count, and it took approx 860 seconds for 500 MB file, which reflects 3400K packet count. We also presented these results in a pictorial form, which is present in figure-7.

Figure figure-7 shows complete picture of detection time taken by different detection models. On horizontal axis we placed total packet counts I.e [100K, 450K, 1000K, 1800K, 2500K and 3400K]. On vertical axis we placed time in seconds. This figure-7 is exact representation of the above 15.

Table 16 shows the average file loading and distribution time taken by spark cluster w.r.t to file size in megabytes. However, reading shows that the least time taken by spark 0.204 seconds for 10MB file while the maximum time taken by spark for the file size of 500MB is 98 seconds, is almost 1.5 minutes. We also presented the above tubular form results in the pictorial form below refer to the figure figure-8. It shows a complete picture of file time taken by different models and log file size. On the horizontal axis, we placed the log file size in megabytes, i.e., [10M, 50MB, 100M, 400MB, and 500MB]. On the vertical axis, we placed time in seconds, i.e., varies from 1 second to 120 seconds. Table 17 and figure-9 respectively show the average of three scientific readings of time taken by each model to the total packets count. We noticed that preprocessing time increases as the file size or packets count increases in every case.

File Size	DT	RF	NB	SVM
10MB	0.3833	0.204	0.4069	1.0088
50MB	4.1349	0.5576	1.7388	1.2349
100MB	5.4611	0.5618	2.6002	3.2608
200MB	8.8026	0.5812	4.5489	9.1338
400MB	53.7333	2.9055	32.6329	15.5418
500MB	98.2872	60.3406	121.6788	26.4873

TABLE 16. File loading and distributing time taken by spark w.r.t file size

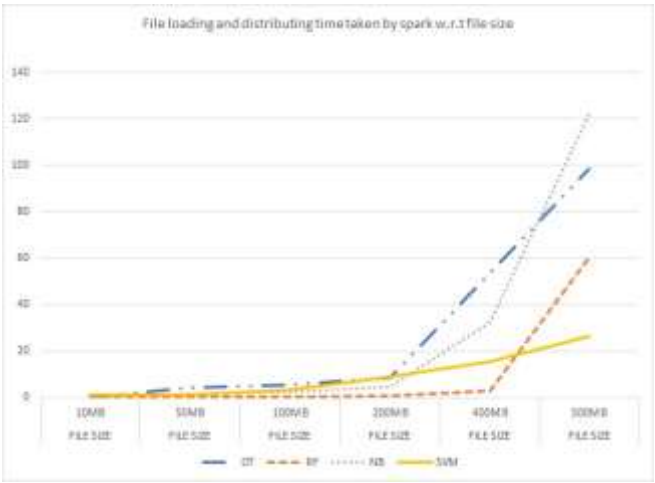

FIGURE 8. File loading and distributing time taken by spark w.r.t file size

Packet Count	DT	RF	NB	SVM
100K	1.5676	1.2247	0.5849	89.9029
450K	3.4877	2.5367	2.5141	137.0856
1000K	7.7287	4.4672	4.4725	281.1723
1800K	25.3764	9.1906	8.7517	642.9673
2500K	180.3885	216.0378	94.5724	724.8311
3400K	295.8082	335.2831	338.404	861.5172

TABLE 17. Preprocessing time taken by model w.r.t file size

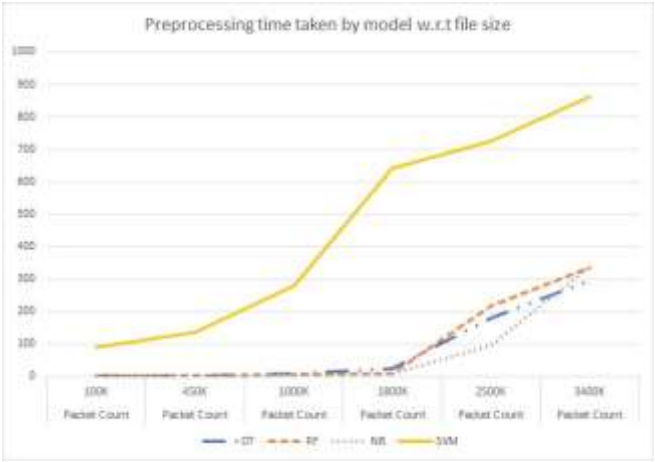

FIGURE 9. Preprocessing time taken by model w.r.t file size

File Size	DT	RF	NB	SVM
10MB	6.3333	5.0001	4.1666	352.6666
50MB	11.0002	7.3333	11.3333	706.6666
100MB	19.0001	11.3333	36.0001	1270.3333
200MB	252.6666	105.6666	165.6666	2086.6666
400MB	707.3333	644.3333	614.3333	3466.6666
500MB	1079.3333	1075.6666	1550.6666	4460.3333

TABLE 18. Time taken by spark w.r.t file size a.k.a SPARK UI TIME

However, at the same time, we noticed that it is not directly proportional to packets count. However, it is only dependent upon the machine learning model how fast it preprocess the small chunk of packet count and how much time it will take for a large chunk of data. Naive Bayes remains the best preprocessing model for 100K packets, and it took only 0.6 seconds to preprocess while at the same time it took approx 338 seconds for 3400K packets count. On average, RF remained the best model among all; it took only [1.23 seconds and 335 seconds] for 100K and 3400K packets counts, respectively. SVM remains the worst preprocessor as well as worst classifier refer to table 15 and figure-7.

File Size	DT	RF	NB	SVM
10MB	6.3333	5.0001	4.1666	352.6666
50MB	11.0002	7.3333	11.3333	706.6666
100MB	19.0001	11.3333	36.0001	1270.3333
200MB	252.6666	105.6666	165.6666	2086.6666
400MB	707.3333	644.3333	614.3333	3466.6666
500MB	1079.3333	1075.6666	1550.6666	4460.3333

TABLE 18. Time taken by spark w.r.t file size a.k.a SPARK UI TIME

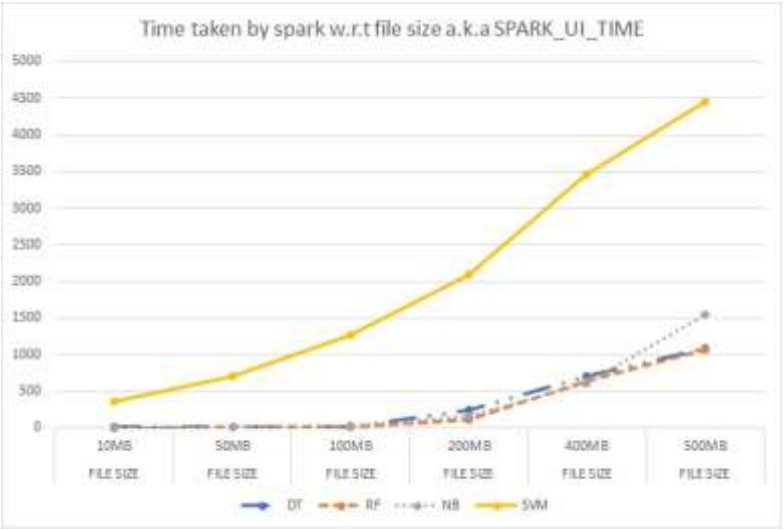

FIGURE 10. Time taken by spark w.r.t file size a.k.a SPARK UI TIME

Table 18 lists the total time taken by the spark user interface, which is the actual sum of all individual times, including (file time, preprocessing time, and detection/classification time). Figure-10 is the pictorial presentation of the above-mentioned table. Refer to Figure [7, 8, 9, 10, 11 and 12] for low testbed results conclusion. figure-7 and Table 15 shows detection time taken by a specific model/classifier to classify packets with respect to the packet count. We noticed that for both [Decision Tree and Random Forest] models took less 5 seconds to classify 100K packets while [Random Forest] model took approx 21 seconds and SVM took maximum time than all three. We also noticed that overall [Random Forest] perform better than all other classifier in terms of detection packets counts per second.

ODDNESS needs to go here

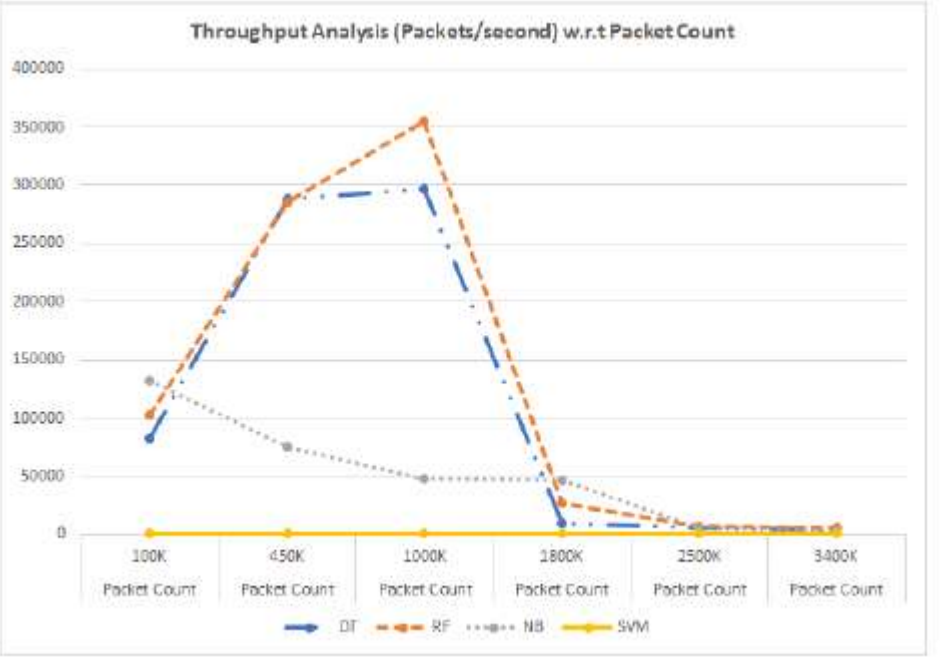

Figure 11. Throughput Analysis (Packets/second) w.r.l Packet Count

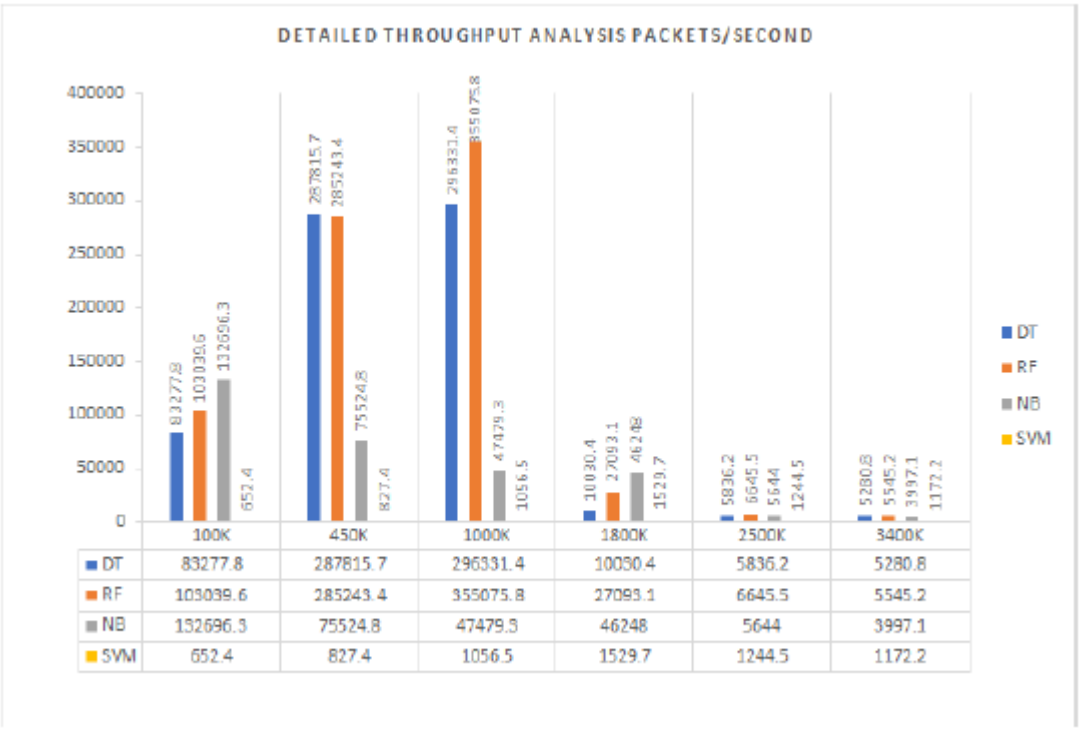

Figure 12. Detailed throughput analysis packets/second

Packet Count	DT	RF	NB	SVM
100K	0.1707	0.4046	2.518	305.1077
450K	0.5454	1.2576	7.1007	454.2456
1000K	1.8796	3.0908	11.2807	916.556
1800K	5.0688	5.5479	27.5153	1540.5229
2500K	6.6027	93.9212	180.1262	1718.1377
3400K	79.8791	156.749	478.5542	1844.2807

Table 19. Detection time taken by different classifier w.r.l Packet Size

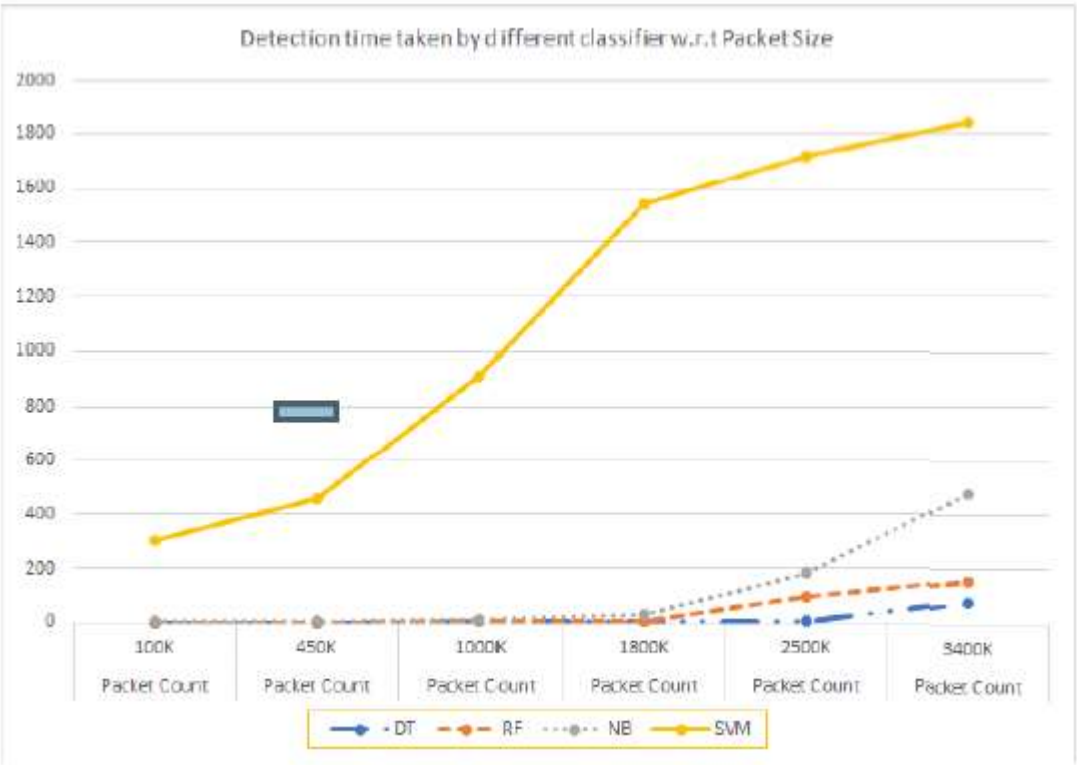

Figure 13. Detection time taken by different classifier w.r.l Packet Size

D. HIGH END TESTBED RESULT CHARTS

Table 19 shows the average detection time taken by different detection models, namely [DT: Decision Tree, RF: Random Forest, NB: Naive Bayes and SVM: Support Vector Machine]. We noticed that DT performed better than all three remaining models and took 0.17 and 0.5 seconds for 100K and 450K packets count respectively. In contrast, DT took only 80 seconds to process 3400K packets, which is far better than our previous spark cluster i.e., low-end testbed. On the other hand, RF took 0.4 seconds for 100K packets count. We also noticed that on average, RF performed well w.r.t NB and SVM while SVM is taking maximum time w.r.t all three models. SVM took almost 305 seconds for a 10MB log file, consisting of 100K packets count, and it took 1844 seconds for 500MB file, which reflects 3400K packet count. We also presented these results in a pictorial form in Figure-13.

Figure-13 shows complete picture of detection time taken total packet counts i.e. [100K, 450K, 1000K, 1800K 2500K and 3400K]. On vertical axis we placed time in seconds. This figure-13 is exact representation of the above 19.

File Size	DT	RF	NB	SVM
10MB	0.4007	0.2865	0.4239	0.5681
50MB	1.0771	0.2757	1.0425	0.7659
100MB	1.7559	0.2793	1.7353	1.7181
200MB	4.2826	0.2687	3.4016	3.1394
400MB	5.305	0.272	7.3116	9.2446
500MB	11.4324	0.2751	9.8913	11.9497

TABLE 20. File loading and distributing time taken by spark w.r.t file size

Table 20 presents the average file loading and distribution time taken by spark cluster w.r.t to file size in megabytes, reading shows that least time taken by spark 0.2865 seconds for 10MB file while the maximum time taken by spark for the file size of 500MB is approx 12 seconds which is also better than our low-end cluster.

We also presented the above tubular form results in the pictorial form below refer to the figure-14. It shows a complete picture of file time taken by different models and log file size. On the horizontal axis, we placed a log file size in megabytes, i.e., [10M, 50MB, 100M, 400MB, and 500MB]. On the vertical axis, we placed time in seconds, i.e., varies from 1 second to 120 seconds. Table 21 and figure-15 respectively show the average of three scientific reading of time taken by each model to the total packets count. We noticed that preprocessing time increases as the file size or packets count increases in every case. However, at the same time, we noticed that it is not directly proportional to packets count.

Packet Count	DT	RF	NB	SVM
100K	0.5234	0.5074	1.1769	2.6776
450K	1.5384	1.433	3.1147	4.5669
1000K	3.4672	3.2363	5.259	6.524
1800K	8.9085	6.0414	11.2844	82.0879
2500K	18.3818	11.5615	94.2082	359.9464
3400K	20.7432	19.8323	122.1618	564.5891

TABLE 21. Preprocessing time taken by model w.r.t file size

However, it is only dependent upon the machine learning model how fast it preprocess the small chunk of packet count and how much time it will take for a large chunk of data.

Random Forest remains the overall best preprocessing model for 100K till 3400K packets, and it took only 0.5 seconds to preprocess 100K while at the same time it took approx 19.8 seconds for 3400K packets count. While DT, NB performed well w.r.t low-end testbed and SVM remains the worst preprocessor as well as worst classifier refer to table 19 and figure-13.

File Size	DT	RF	NB	SVM
10MB	4.3333	5.3333	6.0001	311.6666
50MB	6.0001	7.0001	12.0001	522.3333
100MB	8.0001	9.3333	19.0001	965.6666
200MB	19.3333	16.6666	44.6666	1648
400MB	32.3333	126.3333	289.3333	2118.3333
500MB	191.0001	201.3333	647.3333	2493

TABLE 22. Time taken by spark w.r.t file size a.k.a SPARK UI TIME

Table 22 lists the complete time taken by spark user interface which is actual sum of all individual times including (file time, preprocessing time and detection/classification time). figure-16 is the pictorial presentation of above mentioned table.

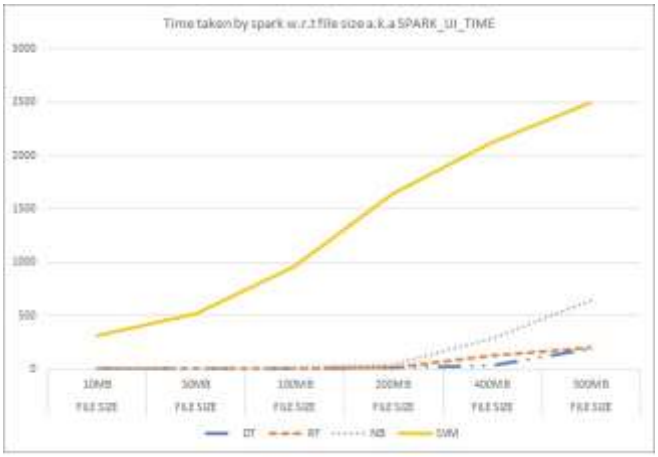

FIGURE 16. Time taken by spark w.r.t file size a.k.a SPARK UI TIME

Refer to figures[13, 14, 15, 16, 17 and 18] for low testbed results conclusion. figure-13 and Table 19 shows detection time taken by specific model/classifier to classify packets with respect to the packet count. We noticed that for both [Decision Tree and Random Forest] models took less than five seconds to classify 1800K packets while [Random Forest] model took approx 27 seconds, and SVM took maximum time than all three. The reader can notice the actual difference in time to detect several packets with low-end and high-end testbeds. We also noticed that overall, [Random Forest] performs better than all other classifiers in terms of detection packets counts per second. It took only 156 seconds to detect 3400K packets, which is 400 percent better than the low-end.

FIGURE 17. Throughput Analysis (Packets/second) w.r.t Packet Count

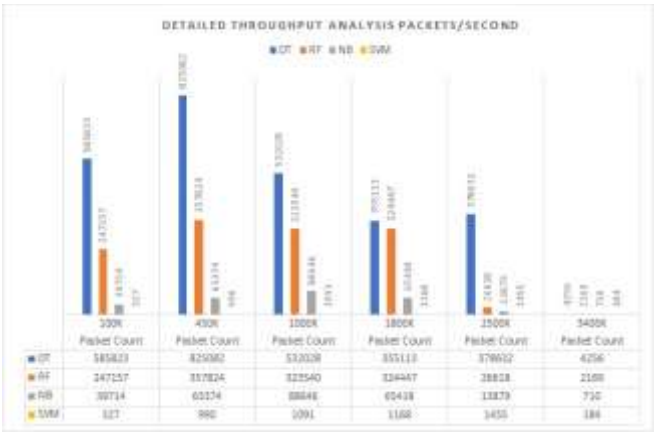

FIGURE 18. Detailed throughput analysis packets/second

E. OVERALL DISCUSSION

We computed and presented actual results in terms of detection time and remaining referenced figures[14, 15 and 16], and tables[20, 21 and 22] are for reference to the reader for self understanding that how much time (in seconds) our framework took to load file, to preprocess and last but not the least how much actual time taken by spark testbed on each run.

Figures[11 and 17] are representing throughput of both testbeds respectively. While figures[12 and 18] are showing additional information of throughput and referenced both charts are additional efforts to understand results in better manner.

In addition to the system execution, figure 19, figure 20, figure 21 and figure 22 present a step by step result of the testbed, which includes "view of Spark executor after submitting job to testbed", "preprocessing of data","snapshot of spark user interface after submission of job to testbed" and "view of simulated traffic".

After submission of spark job, one can see its status, including worker node information, spark cluster specification, running jobs, and completed the job by visiting spark webuser interface using REST URL or SPARK URL.

Spark executors are proof that the job is being executed in the right manner, as reflected in figure 19. Figure-20 reflects the snapshot of preprocessing work, which is impressive for detailed working information of the preprocessor module to the model/classifier.

In Figure-22 vulnerable traffic is highlighted with red color to distinguish from normal traffic. The proposed framework is not generating any automated notifications for manual action, but this can be extend and incorporated in the framework for further actions.

VII. CONCLUSION

In this paper, we present SAD-F, a scalable Spark-based live DDoS detection framework capable of analyzing potential DDoS attacks with no time delays, as the performance of the framework tested against live and passive traffic. SAD-F captures live network traffic, preprocesses it to extract relevant information in brief form, and uses ML-Spark algorithms to run detection algorithms for DDoS flooding attacks. SAD-F solves the scalability, memory inefficiency, and the process complexity issues of the conventional solution by utilizing parallel data processing with low latency and high efficiency.

Before deployment of the framework, we test the dataset on a local machine having a specification of (8GB of RAM and 2.50 GHz of a processor). The test results show that SAD-F would take a maximum time of 12605 sec, i.e., 210 minutes to train and test the overall file for the worst case with 92% accuracy with the KNN model. It took 6536 seconds, i.e., 108 minutes for the best case with 72% accuracy, with an NB classifier for the maximum file size of 150MB, which comprised an estimated 1300K packets count. However, our suggested SAD-F framework with [low-end] testbed configuration took [5.0001 seconds and 1075.6666 seconds] to process (file loading, preprocessing to classification) for 100K and 3400K packets count respectively with RF classifier as the best case. While the proposed framework showed best results, according to proof of study, SAD-F framework with [high-end] testbed configuration took [4.3333 seconds and 191.0001 seconds] to process (file loading, preprocessing to detecting) for 100K and 3400K packets count respectively, with DT classifier as the best case. Based on the framework benchmarks, we conclude that the active mode of our proposed framework is significantly efficient in terms of preprocessing and DDoS detection than the passive mode, the traffic selector method.

Further, we noticed that the data capturing phase consumes more memory than the preprocessing phase. Moreover, the appropriate increase in cluster size can increase the lack of CPU utilization, which will improve each stage of this framework. The proposed framework can be useful in parallel with conventional security mechanisms to optimize the results and detection time.

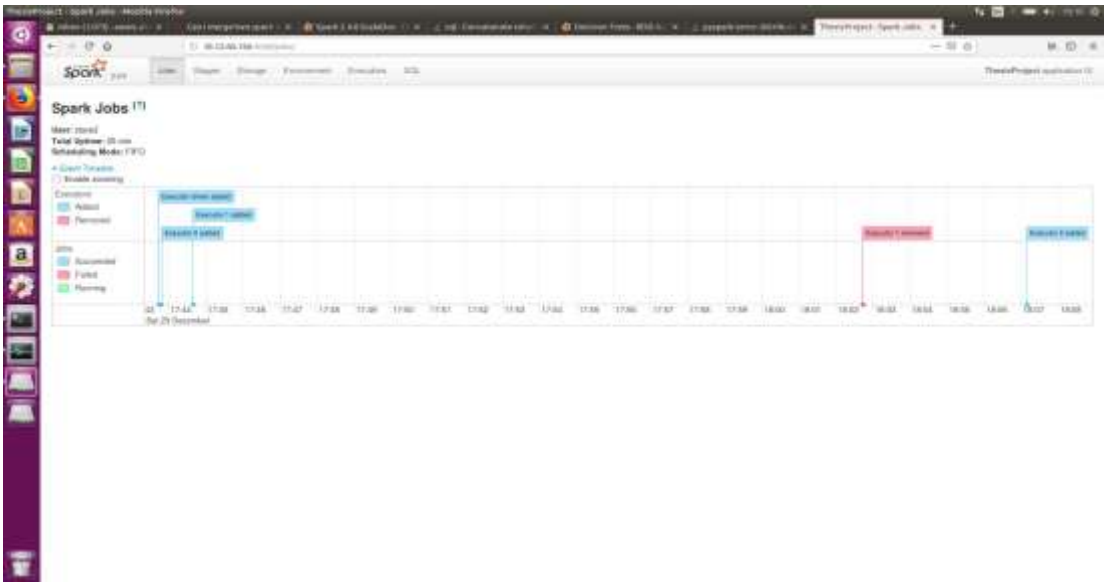

FIGURE 19. View of Spark Executor after submitting job to testbed

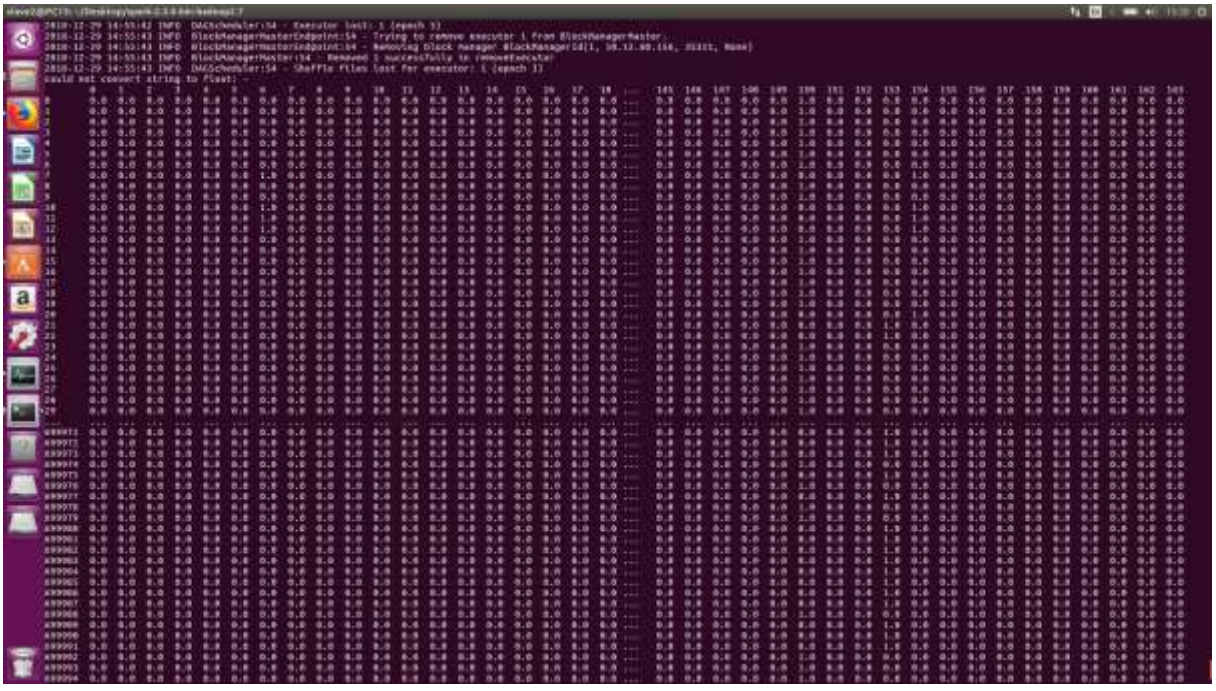

FIGURE 20. SAD-F: View of Preprocessing of Data

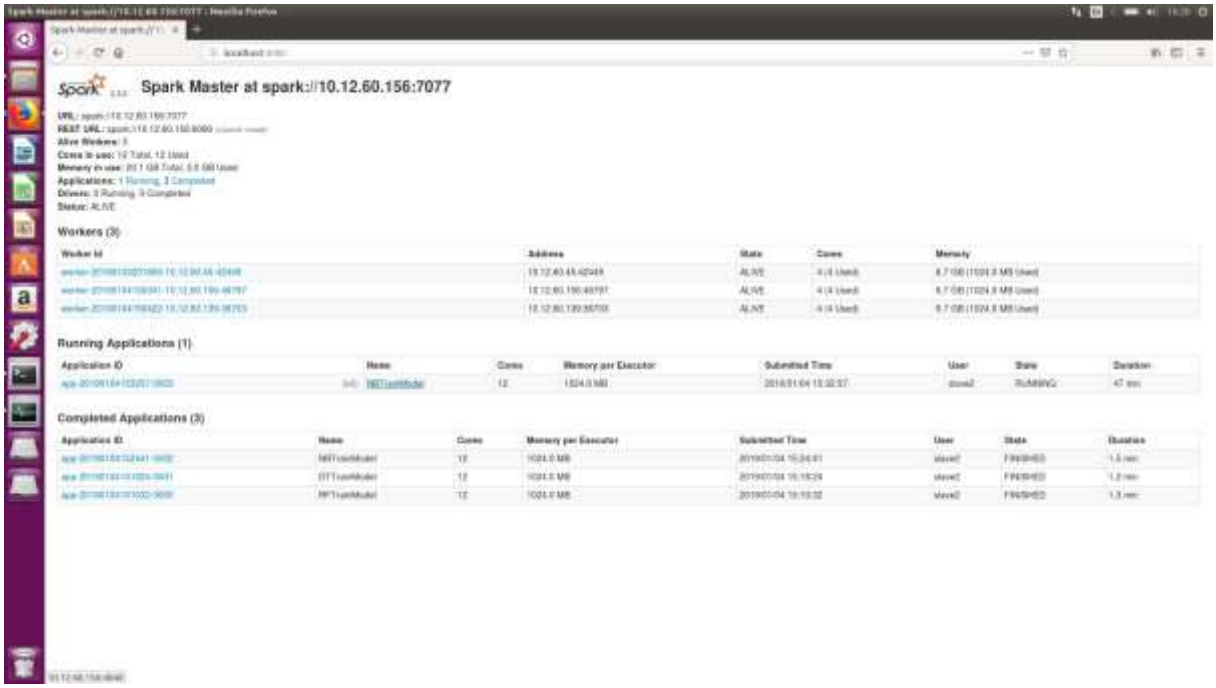

FIGURE 21. Spark-UI: After submitting a job to testbed

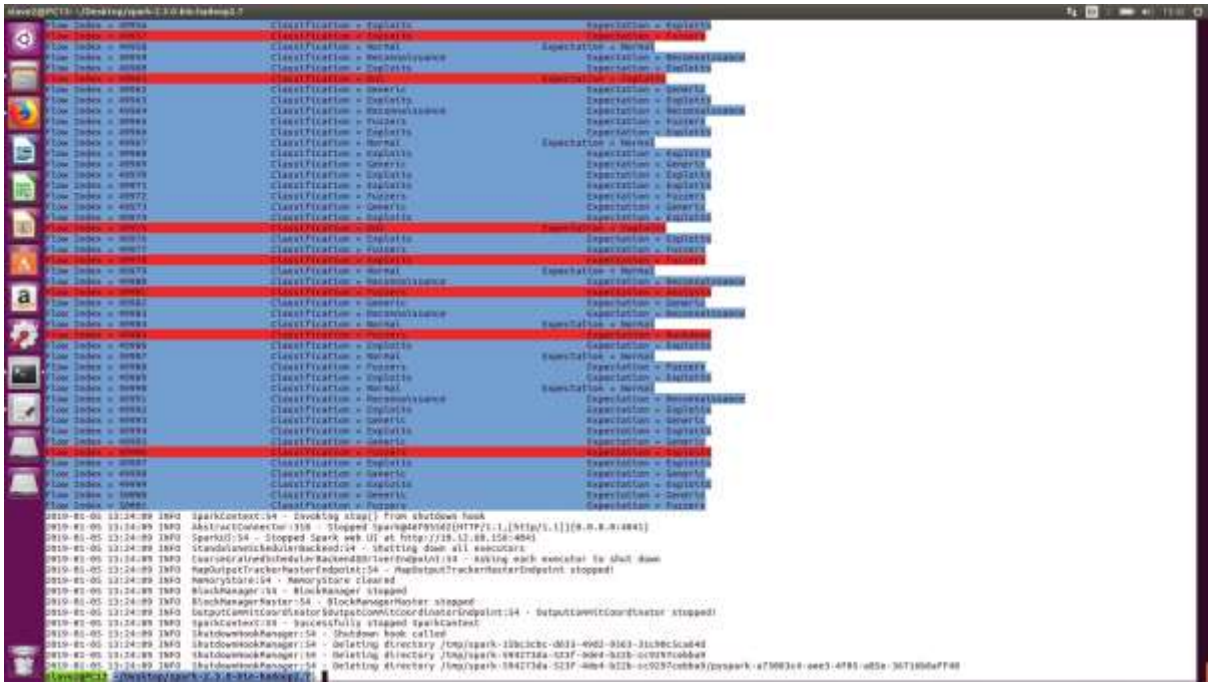

FIGURE 22. Snapshot view of simulated traffic: SAD-F-Result