Distributed Network Traffic Feature Extraction for a Real-time IDS

Ahmad M Karimi[1], Quamar Niyaz[1], Weiqing Sun[2], Ahmad Y Javaid[1], and Vijay K Devabhaktuni[1]

Dept. of EECS[1]/ET[2], College Of Engineering
The University of Toledo, Toledo, OH-43606, USA
{ahmadmaroof.karimi, quamar.niyaz, weiqing.sun, ahmad.javaid, vijay.devabhaktuni}@utoledo.edu

***Abstract*—Internet traffic as well as network attacks have been growing rapidly that necessitates efficient network traffic monitoring. Many efforts have been put to address this issue; however, rapid monitoring applications are needed. We propose a distributed architecture based intrusion detection system (IDS) that is capable of detecting the anomalies in the network in real-time. To achieve this, we exploit the Apache Spark framework and Netmap– a line-rate packet capturing tool. In this work, we implement one of the challenging modules of an IDS, i.e., feature extraction, and present the computational results of the same for TCP-based traffic. Related results are presented along with the insight gained for future work.**

***Keywords*—Apache Spark, Netmap, HDFS, Feature Extraction, Intrusion Detection System.**

I. Introduction

Recent years observed an enormous growth in the Internet traffic. According to the Cisco report, the global Internet traffic would reach zettabytes (10^{21}) in the year 2016 and twice of that by 2019 [1]. As rapidly the network traffic increases, so does the growth in attacks. For example, the proportion of attacks targeting TCP applications increased by 18%, and 75% of 50-100 Gbps attacks occurred in US & Canada of which 99.2% were TCP SYN attacks in Q2 2015 [2]. Therefore, it is important for organizations to get equipped with proper attack detection and mitigation tools. Organizations use either statistical or machine learning based approaches for implementing IDSs to detect attacks on their networks. All of these methods heavily rely on features or statistics, computed either for network packets or flows.

Increased traffic volume makes feature extraction task computationally intensive and imposes several challenges to implement a real-time IDS. In one of the conventional methods, a monitoring system uses sampling based technique to capture traffic using NetFlow [3]. Then, a high-end stand-alone system receives the captured traffic for feature extraction. It takes a lot of time in computation to extract features even in offline analysis. With a wide acceptance of Hadoop [4] in big data analysis, few researchers used it as an alternative approach to make the feature extraction faster. In this method, data files (binary/text) for network packets/flows are broken into different blocks and distributed over the cluster for processing.

The alternative approach significantly improved the performance compared to the stand-alone system. However, it still suffers from a few limitations. First, Hadoop performs too many disk I/O operations on transient data that puts a constraint on its processing speed [5]. Second, the IDSs implemented using features extracted from the sampled flows produce too many false-positives. Feature extraction from sampled flows is reasonable for core or backbone networks as capturing network packets/flows at gigabits/s rate is highly resource consuming. However, at the access or campus networks, sampling can be avoided for better results. Few researchers used `dumpcap` or `tshark` [6] to capture all the network flows/packets at campus networks to implement IDSs that could produce more accurate results. Although these tools work fine for regular traffic, they drop packets when voluminous traffic or DDoS attacks occur. These issues altogether put a hindrance on the implementation of a real-time IDS.

We propose system architecture of an IDS that could run in near real-time to address the issues mentioned earlier. Importance of feature extraction in development of a real-time IDS serves as the primary motivation of this work. To develop a system for efficient feature extraction, we use Netmap [7] and Apache Spark [8]. Both tools are open-source. Netmap is capable of capturing packets at line-rate in contrast to the previously mentioned tools, hence, will help in reducing the false-positives. Apache Spark is a distributed computing framework that processes data more efficiently compared to Hadoop and will make our system faster for feature extraction. We use our implemented system for feature extraction of TCP-based network traffic and evaluate the results for attacks modeled on CAIDA attack dataset [9].

Rest of the paper is organized as follows. In Section II, we discuss previous works related to distributed network monitoring. Section III gives a brief overview of various tools that we use in our system. System architecture and design are discussed in Section IV. Section V presents experimental setup and results. Finally, we discuss future work that we plan to incorporate in our system for its enhancement in Section VI.

II. Related Work

Various applications using distributed computing have been developed for fast packet/flow processing, inbound or outbound to networks, to detect different attacks. Hadoop's *map-reduce* received wide attention among them. In [10], Lee et al. proposed a traffic flow processing system using map-reduce. The authors exported the network flows to the distributed system and gathered traffic statistics. They mentioned that the system can be deployed for online traffic flow processing; however, the emphasis was

978-1-4673-9985-2/16/$31.00 ©2016 IEEE

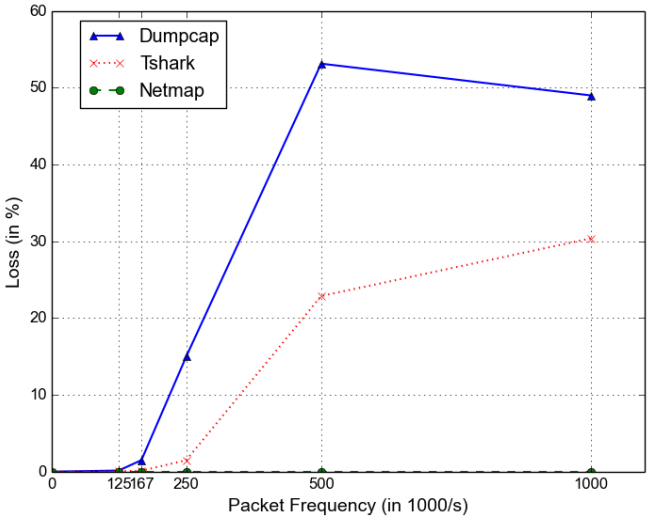

Fig. 1. An evaluation of `netmap-libpcap`, `tshark`, and `dumpcap` under TCP flooding using `hping3` tool

given to offline analysis. They worked on samples collected using NetFlow instead of analyzing every flow. In [11], Singh et al. proposed peer-to-peer botnet detection system in quasi-real-time using Hadoop. They captured network packets using `dumpcap` and extracted required fields using `tshark`. Map-reduce, then, processed these fields for feature extraction. In [12], Luo et al. proposed traffic analysis with deep packet inspection using Hadoop and integrated it with a web crawler. In this work, they focused on analysis of e-commerce websites and investigated popularity of different products and brands. In [13], Fontugne et al. proposed map-reduce based anomaly detection framework called Hashdoop. To maintain a spatial and temporal structure of the stored captured packets, they have used hash functions to split data. In [14], Tazaki et al. proposed cyber threat detection modules and implemented Matatabi framework using Hadoop. The framework performs batch-processing in every 24 hours on logged data and packet traces collected during the day. In [15], Bumgardner et al. developed a Hadoop-based scalable system to monitor traffic information. They have proposed a method of batch processing and stream processing with lower speed. They have deployed their system at university network. To overcome the slow read/write in Hadoop Distributed File System (HDFS), they used HBase module.

Although Hadoop performs well in scaling-out the storage space, it is not fast enough to process the stored packets/flows due to the frequent occurrences of disk I/O in it. Few researchers used different alternatives of Hadoop. In [16], Bar et al. proposed a solution for network traffic monitoring with DBStream. They focused on rolling data in their work. In [17], Simoncelli et al. proposed a framework for fast data analysis on distributed hosts using BlockMon, a network monitoring platform. They have compared their result with Apache Storm and S4 with performance improvement of 2.5 and 23 times, respectively.

After going through all these related works, we wanted to come up with a system which is efficient, fast and implemented using latest tools as well as predominant framework. To accomplish this objective, we focused on real-time processing of network packets to collect traffic features in our work.

III. Tools Overview

In this section, we describe the tools that we use for traffic feature extraction to develop an IDS. They are as follows:

A. Netmap

Capturing packets at line-rate is one of the most important requirements for feature extraction to develop a real-time IDS. We use Netmap to address the issue of fast packet capturing [7]. Its implementation is hardware-independent and has been easily integrated with Linux or Unix based systems with minimal modifications. Network applications find huge performance improvement when implemented over Netmap since it utilizes low-level packet I/O system calls. Netmap facilitates `libpcap` to run on top of it without any modifications. We use `netmap-libpcap` in our work.

Before selecting `netmap-libpcap` for fast packet capturing tool, we compared it with `dumpcap` and `tshark`. These tools are quite popular for packet sniffing, and they have been used in previous works. We ran experiments for packet capturing using all these tools. We used `hping3` [18] tool to generate TCP SYN floods from one host and captured traffic on another host using them. We varied packet frequencies from 0.125 million packets per second (Mps) to 1 Mps. Fig. 1 shows that loss observed is negligible for `netmap-libpcap`, whereas the other two incur significant losses when we increase packet frequency. One important point to be mentioned here is that we captured packets as well as extracted header fields with `netmap-libpcap`. In contrast, we captured packets and dumped raw headers for the others, even then we found reduced performance compared to `netmap-libpcap`. This motivated us to use `netmap-libcap` for packet capturing.

B. Apache Spark

Applications involving high-speed and voluminous data need fast processing requirements and huge storage space. Systems like Hadoop are good in providing distributed storage space, however, lack in efficient data processing. Apache Spark was introduced to address this issue [5]. It is an open-source distributed framework for fast data processing that facilitates in-memory storage to hold transient data that is produced due to actions taken over various transformed data. Spark performs 100x and 10x faster computation than Hadoop for in-memory (cached) and disk data, respectively [8]. It provides an abstraction of the read-only collection of objects called Resilient Distributed Dataset (**RDD**). It stores data in the form of immutable RDDs. These RDDs are fault-tolerant, and Spark can cache them in memory for further computations. Spark provides various APIs to *transform* the RDDs from one form to another and perform *actions* on them including map, flatmap, filter, reduce, collect, and count [8].

Spark SQL was developed to build a system that supports relational operations on datasets similar to traditional databases. It runs on top of Spark framework and convert an RDD to dataframe. The dataframe lets the users interact with it using Spark SQL. A dataframe is equivalent to a table in the relational

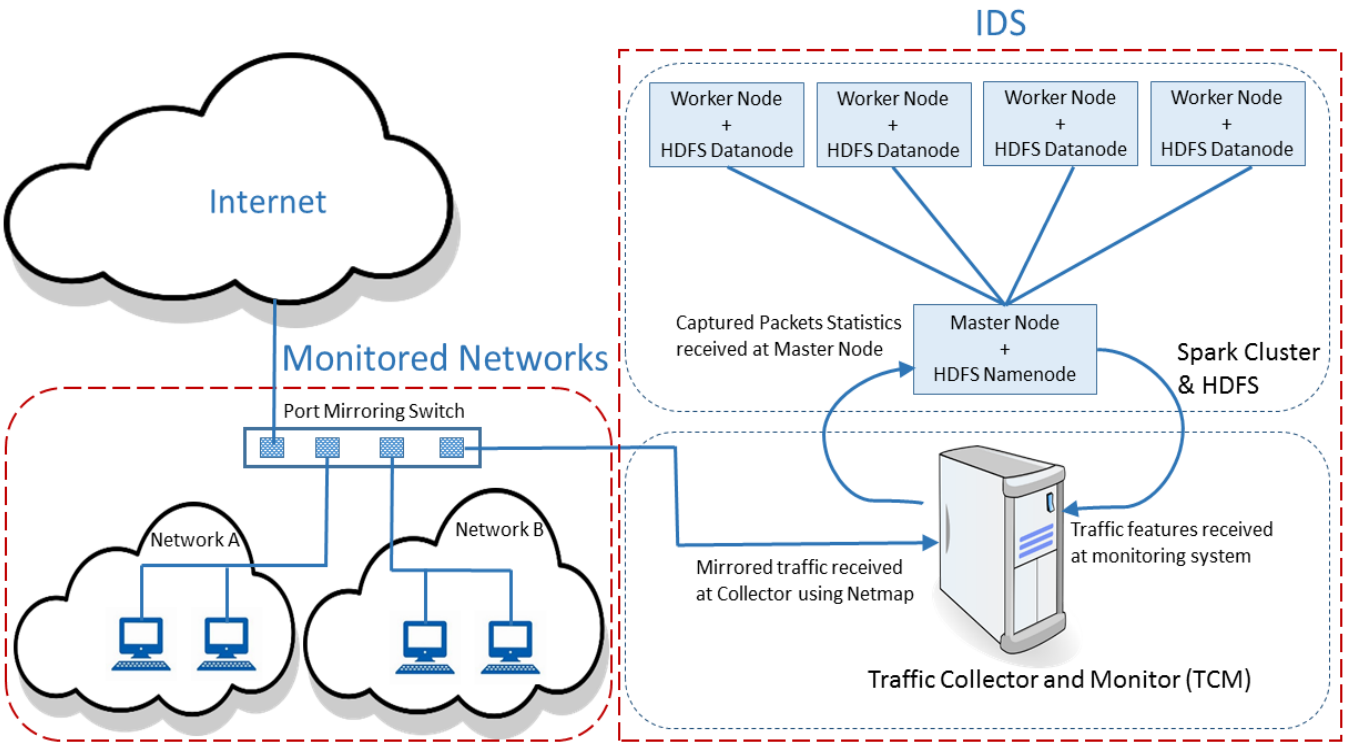

Fig. 2. Architectural representation of the proposed IDS. The IDS consists of two components: i) Traffic Collector and Monitoring (TCM) and ii) Spark Cluster

database once gets instantiated. It can be registered as a temporary table as well. It allows the users to execute SQL commands on it in both cases. Our work heavily relies on the Spark SQL. Spark also provides a fast machine learning library, *MLib*, that leverages with high-quality learning algorithms.

As a last note, an important feature of Spark is its ability to work alongside other distributed systems like Hadoop and share the data stored on Hadoop cluster, i.e., HDFS.

C. Hadoop Distributed File System

Hadoop Distributed File System (HDFS) is a framework for distributed data storage and processing. The main objective of HDFS is to provide reliable disk space for huge volume of data storage. HDFS achieves high reliability on account of data replication. The number of times data has to be replicated is configured during the initial set-up of Hadoop. Although it is built to run on commodity hardware, HDFS provides high throughput to the application [19]. We use HDFS for data storage in our work. Spark applications can connect to Hadoop either through stand-alone mode or Hadoop's own resource cluster manager. Job scheduling is done by Spark itself in the former mode, and by the Hadoop's resource manager in the latter.

IV. System Architecture

We propose an IDS to detect different kinds of DDoS attacks in real-time. Although we describe here the architecture of our IDS, this work addresses efficient feature extraction- a major challenge in enabling an IDS to work in real-time. We modularize the IDS into two main components that perform five tasks. Fig. 2 shows the architectural diagram of our IDS. One of the components is Spark cluster used for extracting traffic features and the other is network Traffic Collector and Monitor (TCM). Hadoop is installed in the Spark cluster so that HDFS can provide distributed data storage facility. Following are the five tasks that our IDS performs:

1) Live capturing of packets from monitored networks and extract required headers from them
2) Distribute the extracted headers on the data storage cluster
3) Extract traffic features from the distributed packet headers data
4) Analyze the traffic features for anomaly detection
5) Train and update the IDS algorithm for anomaly detection using *MLib*

The IDS performs all these tasks periodically, except the last one, within a small-time window of a few minutes to make itself nearly real-time. It performs the last task after certain gaps, for example, once in a month, after its proper training for anomaly detection.

The *collector* component of TCM receives packets from a port mirroring switch installed on the gateway of monitored networks of the organization and Internet. The switch copies all the inbound and outbound packets of the monitored networks and sends them to the *collector* through the mirrored port. The *collector* application runs over `netmap-libpcap`, captures the mirrored traffic, and extracts required packet headers. It saves these extracted headers to CSV files. For each time window, the *collector* creates a separate CSV file and labels it with a

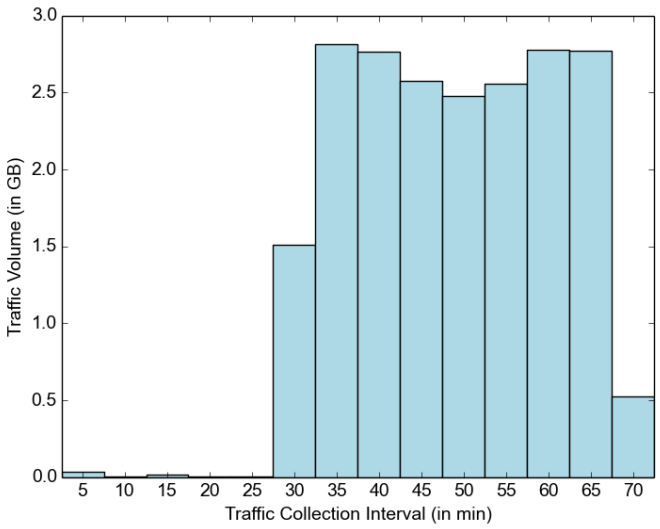

Fig. 3. An hour traffic distribution during a DDoS attack analyzed from CAIDA dataset [9]

timestamp to distinguish it from other CSV files. As soon as the CSV file is created for a particular window, the *collector* transfers it to the HDFS on the cluster for storage and processing. In the processing stage, Spark converts the CSV file into dataframe and registers it as a temporary table. Following that, it queries the structured data using Spark SQL to compute and extract traffic features from it. Once traffic features are extracted, the master node of Spark cluster sends them to the *monitor* component of the TCM for anomaly detection. The anomaly detection algorithm of the *monitor* is trained and updated in the Spark cluster using its *MLib* library.

V. Experimental Set-up and Results

In this section, we discuss our experimental setup and related results. In order to have a good estimate of various DDoS attacks inside an organization, we analyzed the CAIDA's attack dataset [9]. The dataset contains the trace files of a DDoS attack which happened approximately for 40 minutes. Traces were divided into pcap files for every five minutes with a total size of 21.1 GB. Fig. 3 shows the traffic volume for each five minutes. We find that maximum traffic volume was ≈2.8 GB, observed in the interval of 30-35 minutes. We evaluate our system performance with varying traffic volume from 1 GB to 3 GB generated in 5 minutes using `hping3` to model this dataset.

Although we have used TCP-based network traffic in our experiments, the system can be easily extended to all types of network traffic with few modifications in it. TCP provides reliable communication to the applications and ensures that data is delivered to the end hosts with its entirety. It performs three-way handshake for connection set-up and sends an acknowledgment for each received message at the end hosts. Adversaries exploit these features of TCP and make it vulnerable for TCP SYN and ACK based DDoS attacks. The SYN attack overwhelms the listening server with half-open connection requests. The server becomes incapable and unresponsive for processing further incoming requests. In TCP ACK attack, the victim's system gets flooded with ACK messages from a server in response to the

TABLE I. Traffic features extracted for TCP DDoS attack detection

Features	Description
IFF	Number of incoming flows
OF	Number of outgoing flows
FSF	Fraction of symmetric flows
FAF	Fraction of asymmetric flows
BPIF	Bytes per incoming flows
BPOF	Bytes per outgoing flows
PPIF	Number of packets per incoming flows
PPOF	Number of packets per outgoing flows
DSP	Number of distinct source port for incoming flows
DDP	Number of distinct destination port for incoming flows
FHDP	Fraction of destination port less than 1024
FLDP	Fraction of source port greater than 1024
FIPUSHS	Fraction if PUSH flag set for incoming flows
FOPUSHS	Fraction if PUSH flag set for outgoing flows
FISYNS	Fraction if SYN flag set for incoming flows
FOSYNS	Fraction if SYN flag set for outgoing flows
FIACKS	Fraction if ACK flag set for incoming flows
FOACKS	Fraction if ACK flag set for outgoing flows
FIFINS	Fraction if FIN flag set for incoming flows
FOFINS	Fraction if FIN flag set for outgoing flows
FIURGS	Fraction if URG flag set for incoming flows
FOURGS	Fraction if URG flag set for outgoing flows
FIRST	Fraction if RST flag set for incoming flows
FORST	Fraction if RST flag set for outgoing flows

SYN messages sent to the server with spoofed IP of the victim in a botnet environment. Also, few adversarial hosts send a large chunk of datagram segments to a TCP server to launch DDoS flooding attack [20]. We extract various features from TCP traffic flows. These features were taken into consideration after reviewing previous works for TCP-based DDoS attack detection. Table I shows the extracted features for flows. Spark cluster extracts these features from CSV files that consist of header fields and sent to it by the *collector* in each time window. The cluster extracts these features for each destination host residing in the organization. We store following TCP/IP header fields of networks packets into the CSV files to compute the features:

- Source IP
- Destination IP
- Source Port
- Destination Port
- IP Protocol
- IP Payload
- TCP FIN Flag
- TCP SYN Flag
- TCP RST Flag
- TCP ACK Flag
- TCP URG Flag
- TCP PUSH Flag

For our experimental set-up, we used Supermicro SYS-6028R-WTRT server shipped with Intel (R) Xeon (R) @ 2.30 GHz, 96 GB RAM, and 20 CPU core × 2.99 GHz. The server runs VMware ESXi host. We created virtual machines for Spark cluster & HDFS, TCM, and monitored networks in the ESXi host. The Spark cluster consists of 7 nodes, one master node and six worker nodes. We vary the number of worker nodes from 1 to 6 for performance evaluation. Each node in the Spark cluster and TCM are assigned with 4 vCPUs, 8 GB RAM, and 60 GB disk storage. For the monitored networks, we created 10 virtual machines with a configuration of 1 vCPU, 2 GB RAM, and 15 GB disk. We run `hping3` in few of them to launch

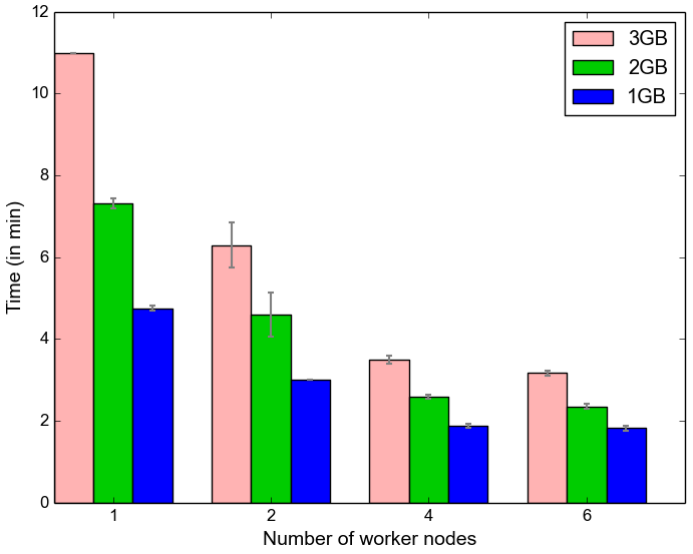

Fig. 4. Time taken in extracting features from different size of CSV files by varying the number of worker nodes in Spark cluster

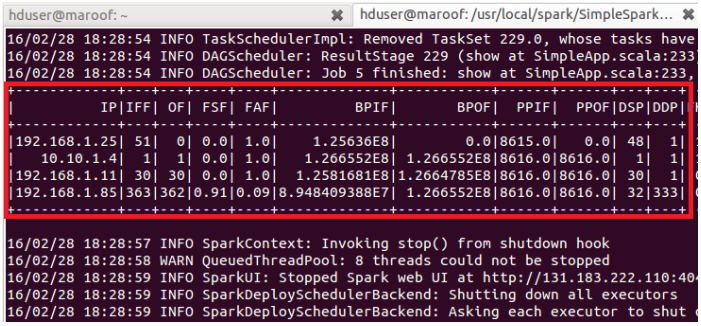

Fig. 5. Extracted traffic features statistics gathered at the master node of the Spark cluster

attacks and made its victims to few of them. Any reduction in performance can be attributed to the fact that the server also hosts several other virtual machines for other research work in our group. A dedicated server was not used to demonstrate the use of commodity hardware in an academic research infrastructure.

Fig. 4 shows the results of experiments with deviations. We observed that when we used 4 and 6 worker nodes, all three different sizes of files were processed within 4 minutes. The 3 GB files, generated in 5 minutes, took 3.17±0.05 and 3.5±0.1 minutes on 6 and 4 worker nodes, respectively. This makes our system to work in real scenarios. When we used 1 or 2 worker nodes in the cluster, the 2 GB and 3 GB files took more than 5 minutes for feature extraction, however, 1 GB files were processed within 5 minutes in all cases. Fig. 5 shows a few of the extracted feature statistics gathered at the master node for each time window.

VI. Conclusion and Future Work

In this work, we implemented an efficient feature extraction system using Apache Spark and Netmap. Currently, our system can extract features from network traffic for small organizations. We verified it by modeling the CAIDA attack dataset. For future work, we aim to scale our system to enable it to process the traffic for core and backbone networks efficiently. We can achieve this by considering two important things: i) an increase in the number of worker nodes in Spark cluster, and ii) an improvement in the implementation. In the current implementation, all the Spark SQL based queries for feature extractions are written in a single Spark job. However, there are many queries that can execute in parallel. We can divide them into separate Spark jobs to make the feature extraction more faster. These two approaches will reduce the computation time significantly and make it real-time for a small time window of a few seconds instead of minutes for large data traffic. Finally, we plan to integrate our system with Spark MLib library to detect the anomaly using extracted features from the traffic.

References

[1] Cisco Visual Networking Index, “http://tinyurl.com/mev32z8.”

[2] Arbor Networks Attack Data, “http://tinyurl.com/htpbl95.”

[3] Cisco Netflow, “http://tinyurl.com/mx4g277.”

[4] Apache Hadoop, “http://tinyurl.com/5f4ojf.”

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2, USENIX Association, 2012.

[6] Wireshark, “http://tinyurl.com/b4ab9gu.”

[7] L. Rizzo, “Netmap: A Novel Framework for Fast Packet I/O,” in *21st USENIX Security Symposium (USENIX Security 12)*, pp. 101–112, 2012.

[8] Apache Spark, “http://spark.apache.org/.”

[9] The CAIDA UCSD DDoS Attack 2007 Dataset, “http://tinyurl.com/z2z7m8a.”

[10] Y. Lee and Y. Lee, “Toward Scalable Internet Traffic Measurement and Analysis with Hadoop,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 5–13, Jan. 2012.

[11] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, “Big Data Analytics Framework for Peer-to-Peer Botnet Detection using Random Forests,” *Information Sciences*, vol. 278, pp. 488 – 497, 2014.

[12] J. Luo, Y. Liang, W. Gao, and J. Yang, “Hadoop based Deep Packet Inspection System for Traffic Analysis of E-business Websites,” in *Data Science and Advanced Analytics (DSAA), 2014 International Conference on*, pp. 361–366, Oct 2014.

[13] R. Fontugne, J. Mazel, and K. Fukuda, “Hashdoop: A MapReduce Framework for Network Anomaly Detection,” in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pp. 494–499, April 2014.

[14] H. Tazaki, K. Okada, Y. Sekiya, and Y. Kadobayashi, “MATATABI: Multi-layer Threat Analysis Platform with Hadoop,”

[15] V. K. Bumgardner and V. W. Marek, “Scalable Hybrid Stream and Hadoop Network Analysis System,” in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE ’14, (New York, NY, USA), pp. 219–224, ACM, 2014.

[16] A. Bar, A. Finamore, P. Casas, L. Golab, and M. Mellia, “Large-scale Network Traffic Monitoring with DBStream, a System for Rolling Big Data Analysis,” in *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 165–170, IEEE, 2014.

[17] D. Simoncelli, M. Dusi, F. Gringoli, and S. Niccolini, “Stream-monitoring with BlockMon: Convergence of Network Measurements and Data Analytics Platforms,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 2, pp. 29–36, 2013.

[18] Hping3, “http://tinyurl.com/j4npksu.”

[19] HDFS, “http://tinyurl.com/oahxu23.”

[20] TCP DDoS Attacks, “http://tinyurl.com/jq92leq.”