Scalable distributed traffic monitoring for enterprise networks with Spark Streaming

Andrés F. Ocampo, Tim Wauters, Bruno Volckaert, Filip De Turck
Ghent University - imec - IDLab
andres.ocampo@ugent.be
tim.wauters@ugent.be
bruno.volckaert@ugent.be
filip.deturck@ugent.be

Abstract: In this paper we present an architecture for traffic monitoring in enterprise networks based on big data technology allowing both streaming and batch processing. Through our streaming processing approach, we can monitor user network activity by analysing parameters like active sessions, used ports, and bandwidth usage per session. Both streaming and batch analysis will form the basis of mechanisms for the detection and identification of anomalous or malignant behaviour as well as mis-configured services. For validation purposes, our prototype has been deployed on the iLab.t Virtual Wall facility. Evaluations are conducted to analyse user behaviour under different traffic types and workloads. Furthermore, we evaluate the scalability of this architecture by analysing the performance when incrementing the number of users in the network, as well as the impact of distributing the big data processing nodes on the responsiveness and performance of the system. Our numerical results show that the proposed architecture scales linearly with respect to the number of users.

Keywords: Network Traffic Monitoring, Network Security, Big Data, Data Streaming Processing, Spark Streaming.

1. Introduction

Network traffic monitoring and analysis represents a key component for network security as it allows the development of detection and identification mechanisms for networks threats and anomalies. Traffic monitoring systems can be classified as active and passive monitoring. Active approaches inject traffic into a network to collect measurements and perform analysis. On the other hand, passive approaches collect traffic observed by a measurement point for being analysed (Hofstede, 2014). Packet capture and flow export are common implementations of passive monitoring. In the former case, complete packets are captured providing deep insight into the network traffic unlike flow export in which packets are aggregated into flows and exported for storage and analysis. The system presented in this paper is based on passive monitoring approach based on packet capture.

Nowadays, modern distributed big data stream processing systems have evolved drastically, in that they have become a feasible technology choice for real time network traffic processing, and then for being considered as tools for passive network monitoring (Jirsik, 2017). Moreover, most of those frameworks implement data science libraries like machine learning that may be integrated and used for network security. A suitable system deployment must be able to ingest and process data in real time, requires storage capabilities for the huge amount of data generated by network monitors, must be able to query and analyse historical data in order to detect anomalies based on captured traffic that are not based on fingerprints of known anomalies, and must have availability of traffic data analysis algorithms (Casas, 2017).

We are focused on the implementation of a traffic monitoring system based on big data technology for enterprise networks. The system presented in this paper first captures user traffic on the network switches using the pcap library (libpcap) (Jacobson, 2016) and then publishes each packet into a data pipeline based on Apache Kafka (Kafka, 2017) responsible for streaming captured network data. Those packets are then fed into a distributed stream processing engine based on the Apache Spark Streaming library for data processing (SparkStreaming, 2017). A second Kafka consumer is used for storing captured network data into a datastore, prototyped using MongoDB (MongoDB, 2018) and integrated with Apache Spark for batch processing. Through our streaming processing approach, we can monitor user network activity by analysing parameters like active sessions, used ports, and bandwidth usage per session. This analysis is performed in a configurable time window allowing to keep track of user behaviour over various time spans. Along with the streaming analysis, batch processing of the stored user traffic aims to compare recent user activity on a meta data level versus historical data. Both streaming and batch analysis will form the basis for anomaly detection mechanisms, capable of identifying anomalous or malignant behaviour as well as mis-configured services.

The reminder of the paper is organized as follows. Section 2 describes an overview on the related work about network monitoring based on big data systems. In Section 3 describes the main characteristics of our prototype. Section 4 presents the experimental results of the study. Finally, section 5 concludes the paper.

2. Related work

The integration of big data system for network traffic monitoring is gaining even more attention in academia and industry. One of the first approaches on network traffic monitoring using Big Data was proposed by Lee et al. (Lee, 2011), through a traffic monitoring system that performs NetFlow (Cisco proprietary protocol for flow aggregation (Cisco, 2018)) analysis employing a MapReduce algorithm capable of manipulating libpcap files. A study for benchmarking stream processing frameworks such as Spark, Samza, and Storm, for real time traffic monitoring based on IP flows is presented by Cermak et.al (Cermak, 2016). They demonstrated that those systems can process at least 500,000 flows/s using 16 or 32 processor cores, which is enough for analyse networks in real time.

Several open source tools have been launched and the development continues. For example, Apache Metron is designed to ingests security telemetry data at high speed and then pushes it to computation and analytics providing a centralized tool for security monitoring and analysis. It provides a set of Storm topologies (Storm, 2017) for streaming, enriching, and storing telemetry in Hadoop. ENTRADA is a Hadoop-based network traffic analysis mainly focused on DNS application for network security, which use Impala query engine and Parquet file format (Wullink, 2016) and is focused. BigDama (Casas, 2017) is big data analytic framework for network traffic monitoring based on Apache Spark with both stream and batch processing and Cassandra as the datastore technology. This framework implements several algorithms for anomaly detection and network security using supervised and unsupervised machine learning models. This system is focused on internet traffic analysis unlike the system presented here which is focused on enterprise networks.

Karimi et al. (Karimi, 2016) proposed a distributed network traffic feature extraction method with Spark for a real time intrusion detection system. They capture packets from the switch and extract the required information from packet headers, periodically according to a time window. Gupta et al. (Gupta, 2016) used Spark Streaming for network monitoring using programmable switches to capture flow aggregation traffic and perform analysis focused on port scan detection. The framework presented in this paper is focused on non-programmable switches for enterprise networks.

3. Network traffic monitoring system: architecture

The system presented in this paper is designed to monitor and analyse user activity on enterprise networks, which are composed of private Local Area Networks (LAN), a demilitarized zone (DMZ) with publicly accessible servers such as DNS, FTP, Web, and e-mail, and an output segment to the internet. A network user is defined as an element belonging to a LAN, and therefore has been granted an IP address into the addressing scheme. It may represent a host of an employee or a service running on top of a server, for example (Stallings, 2015). Figure 1 illustrates a typical enterprise network scenario including the architecture for traffic monitoring and analysis proposed in this paper, which is composed basically of three stages as described below: packet capture, streaming processing, and data storage and batch processing.

3.1 Traffic capturing

Packet capture takes place on network switches following the port mirroring approach, a packet capturing technique used on a network switch that is based on sending a copy of packets seen on the desired ports or even an entire sub network (depending on what was selected for analysis) to another port for data collection and analysis. Moreover, in our prototype we use pycapa (OpenSOC, 2015), an open source tool under Apache license which allows to capture packets from switches interfaces following the Libpcap library and, operating as a Kafka producer, publishes captured data into a Kafa Topic. In effect, Apache Kafka is the technology used for streaming the captured network data for being processed, as depicted in figure 1.

3.3 Stream processing

We use Apache Spark Streaming as the tool for real time processing of network traffic. Packets stored on Kafka topics are ingested into Spark Streaming using the Kafka consumer API for Spark (Spark-Kafka), generating input discretized streams (DStreams) composed of raw Libpcap data. Spark Streaming provides its own high-level

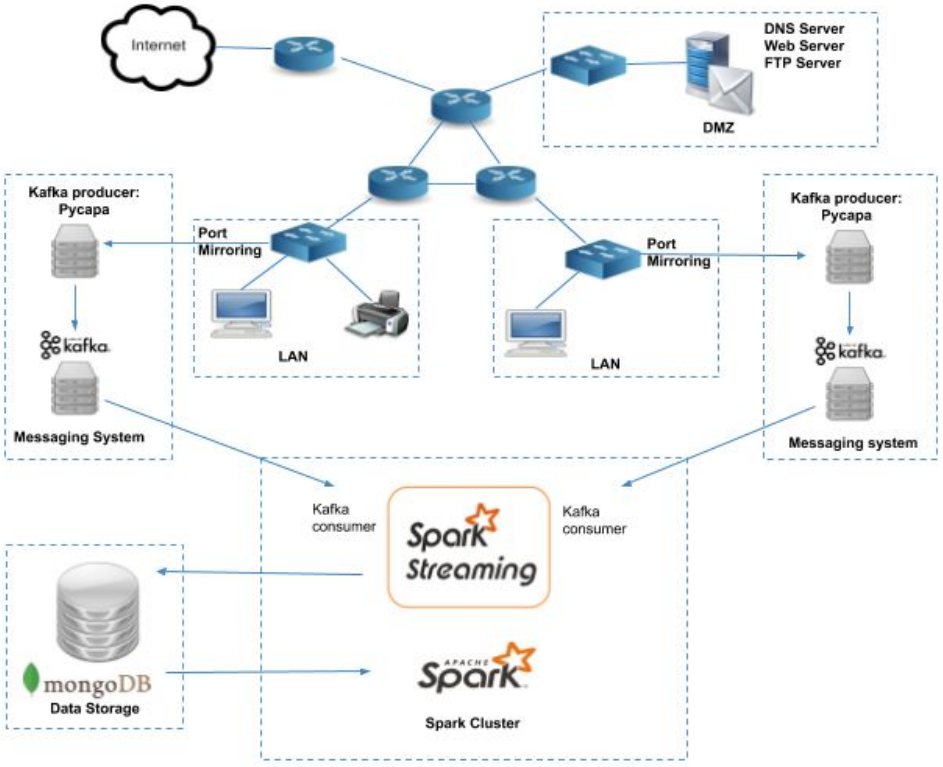

Figure 1. Network scenario and traffic monitoring architecture

abstraction for continuous data stream called DStream, which processes data in micro batches depending on a configured streaming interval (Spark-Streaming, 2017). Figure 2 depicts the data illustrates the pipeline followed by this framework for processing the captured traffic.

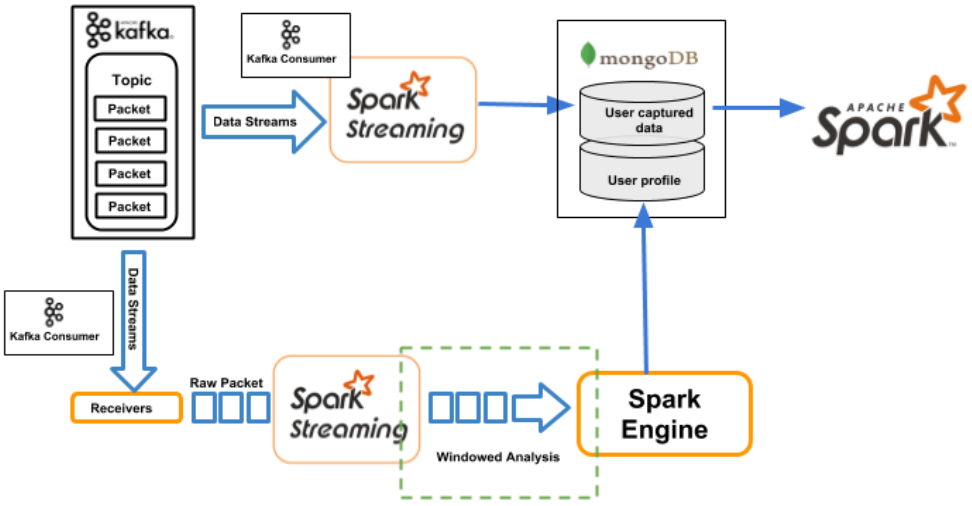

Figure 2. Streaming processing architecture

A set of MapReduce operations are applied to the input Dstream in order to establish a traffic monitoring mechanism, as depicted in figure 3. A first processing stage involves parsing the input raw data which is subsequently divided into incoming and outgoing Dstreams based on whether captured packets have users from the monitored LAN as origin or destination. In other words, traffic arriving or leaving the LAN are analysed separately. Thereafter, DStreams per user are obtained from such traffic streams. It is important to

note that this framework assumes that users from the monitored LAN are known. Then, a windowed analysis is performed over the user Dstreams; based on a time interval with a configurable length we compute metrics such as bandwidth usage of active sessions, bandwidth usage of used ports, most visited destination, most used port, allowing to keep track of user behaviour over the time spans. A second and longer time window is used for cumulative computation of such metrics, which are used as user profiles and updated inside a collection of a database prototyped in MongoDB, as shown in figure 2. Those profiles will form the basis for anomaly detection mechanism.

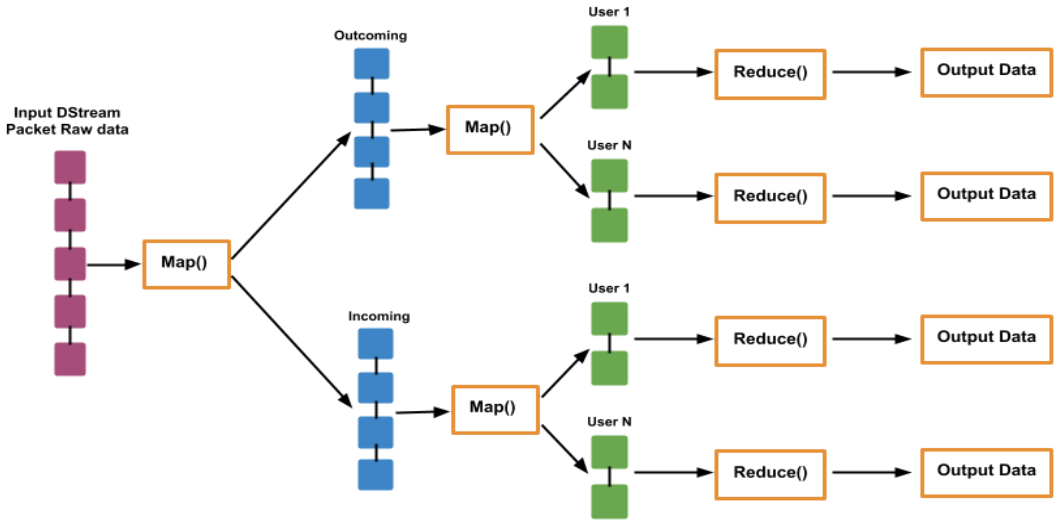

Figure 3. MapReduce programming model

3.4 Data Storage and Batch processing

A second Kafka consumer implemented in Spark Streaming is used for storing captured network data into a datastore prototyped in MongoDB as well, as shown in figure 2. In effect, the input Dstream (as stated in previous subsection follows a Libpcap format) is parsed in order to get the user involved, which is used to write the stream data into a collection document of the form *<user, packets>* of the database designated to store users with all its captured packets. Furthermore, this database is integrated with Apache Spark for batch processing; thus, one can query a specific user and perform processing over its historical traffic data. Along with the streaming analysis, batch processing of the stored user traffic aims to compare recent user activity on a meta data level versus historical data.

4. Evaluation

The outlined architecture was prototyped and deployed on the iLab.t Virtual Wall experimental testbeds (iLabt, 2018). For validation purposes, we set up a network scenario composed of a single LAN where users generate traffic when accessing different services such as FTP, Web, Email, and DNS. We have devised a method to emulate user traffic activity based on a Poisson process. Thus, once a user has finished a transmission, a new service is launched after a waiting or interarrival time which is distributed as an exponential random variable with mean of two minutes for this experiment. In other words, users wait a random time before starting a new packet transmission established for one of the aforementioned services chosen randomly according to a uniform distribution. Packets from the user side are build using Scapy, a Python-based interactive packet manipulation library (Scapy 2016). It is important to note that this arrival process determines the beginning of a service, whereas the traffic dynamics depends entirely on the nature of service being accessed.

We defined a stream processing cluster consisting of one Kafka node with one topic and four partitions (as many as the available cores on the node), three Spark nodes operating in standalone mode and one MongoDB node. All those nodes are deployed using 4 cores, 16GB memory, 100GB drive virtual machines, running on top of 2x Quad core Intel E5520 (2.2GHz) CPU shared machines into the Virtual Wall. The following software and distributed stream processing systems were installed on virtual nodes:

- Ubuntu 16.04 x64
- Python 2.7
- Apache Kafka 0.8.2.1
- Apache Spark 2.2.0
- MongoDB 3.6.2

4.1 Traffic monitoring and user profiling

The stream processing system receives network traffic data in time slots of one second length. Thereafter, processing the stream data is performed based on MapReduce operations as illustrated in figure 3, in order to compute user network activity over time windows of one minute. This in turn enable the online monitoring of parameters like active sessions, used ports, and bandwidth usage per session. Figure 2 shows a 10 minutes sample observation for a certain emulated user, as it exhibits a Web browsing operation during that period, it depicts the measured protocol bandwidth usage.

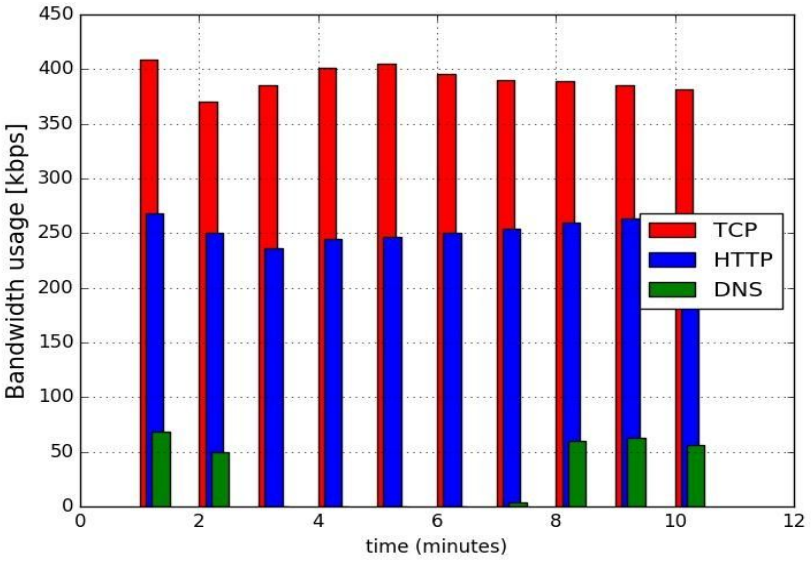

Figure 4. Measured protocol bandwidth usage for one emulated user

Under the assumption that users are habitual when using network services, we can compute and update their behaviour into the user profiles collection in MongoDB. The idea behind user profiling is to provide the basis for further mechanism that can compute outliers on user network usage that allows to detect anomalies such as potentially malignant behaviour from a potentially infected machine or malign user. As an example, consider that at certain point abnormal behaviour might happen with a usual Web browser user when many new connections are being initiated to numerous destinations over the same port number. These are the kind of issues we aim to catch and tackle into the scope of this research, and without the use of known exploit malign behaviour fingerprints, to detect unknown threats propagating through the enterprise network.

We performed cumulative computation over one-hour time window of the following metrics: protocol bandwidth usage, protocol count, port bandwidth usage, port count, destination count (based on IP address). This computation is written into the collection for user profile in MongoDB as an update operation. Table 1 relates output operations performed over certain user profile after being query and processed from Spark.

Table 1: user profile query and processing

	Top protocol bandwidth usage	Top used port	Top port bandwidth usage	Top visited IP
user	70% (http)	40780	63%	192.168.x.x

4.2 Performance evaluation

We evaluate the scalability of this architecture by analysing the mean execution time of the MapReduce processing operations taken over the streaming data. This analysis was made by incrementing the number of users by twenty up to a hundred, all of them accessing to network services as stated above for a period of five hours. We observed the mean time execution when processing is performed by one, two, and three Spark nodes into a cluster operating in standalone mode. Our numerical results show that the proposed architecture scales linearly with respect to the number of users, as shown in figure 5. The highest execution time is reached when processing is performed by one node into the cluster, just above 4 ms delay in performing MapReduce operations. Then, with a batch interval of 1 s such delay is acceptable for processing the amount of user traffic inside this kind of networks.

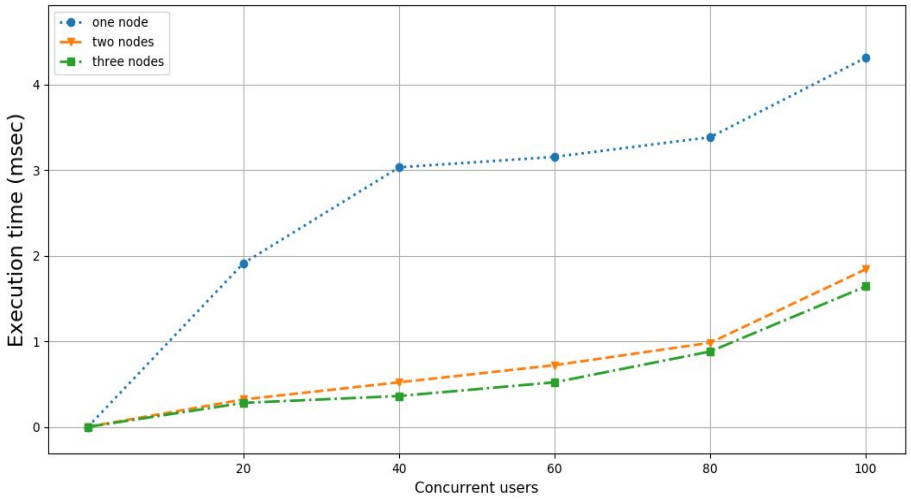

Figure 5. Distributed streaming processing execution time

5. Conclusions and future work

We have presented a framework for traffic monitoring in enterprise networks with Apache Spark as the processing data streaming engine. The conducted experiments have shown that our system scales well under different number of users and workloads. Through the streaming processing approach presented in this paper, it is possible to monitor user network activity over various time spans, which allows to compare current user behaviour with historical data, as well as to allow users to change behaviour without being flagged as malign. Along with the streaming analysis, our system allows batch processing of the stored user traffic aiming to further compare current user activity versus historical data. Both streaming and batch analysis will form the basis for anomaly detection mechanisms, capable of identifying anomalous or malignant behaviour as well as mis-configured services.

Further research is expected to be conducted over this architecture. In first place, it is necessary to design algorithms to update user profiles that captures current and historical data, instead of fixed updating based on windowed computation. On the other hand, bind both streaming and batch processing engines for automate comparisons between current and historical data for those cases in which an anomaly detection alarm is triggered is mandatory. Therefore, the development of anomaly detection mechanisms able to detect potentially malignant behaviour, from a potentially infected machine or malign user is the target of this research project.

References

Cardenas, A., Manadhata, P. and Rajan, S. (2013). *Big Data Analytics for Security*. *IEEE Security & Privacy*, 11(6), pp.74-76.

Casas, P., Soro, F., Vanerio, J., Settanni, G. and D'Alconzo, A. (2017). *Network security and anomaly detection with Big-DAMA, a big data analytics framework*. 2017 IEEE 6th International Conference on Cloud Networking (CloudNet).

Cermak, M., Tovarnak, D., Lastovicka, M. and Celeda, P. (2016). A performance benchmark for NetFlow data analysis on distributed stream processing systems. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*.

Cisco (2018). *Cisco IOS NetFlow*. [online] Cisco. Available at: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html [Accessed 14 Feb. 2018].

Gupta, A., Birkner, R., Canini, M., Feamster, N., Mac-Stoker, C. and Willinger, W. (2016). *Network Monitoring as a Streaming Analytics Problem*. Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16.

ilabt (2018). *Virtual Wall — iLab-t testbeds 1.0.0 documentation*. [online] Available at: http://doc.ilabt.iminds.be/ilabt-documentation/virtualwallfacility.html [Accessed 7 Feb. 2018].

Hofstede, R., Celeda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. and Pras, A. (2014). *Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX*. *IEEE Communications Surveys & Tutorials*, 16(4), pp.2037-2064.

Jacobson, V., Leres, C., McCanne, S. (2016). *The libpcap packet capture library*. Lawrence Berkeley Laboratory, Berkeley, CA, [online] Available at: http://www.tcpdump.org/ [Accessed 12 Feb. 2018].

Jirsik, T., Cermak, M., Tovarnak, D., and Celeda, P. (2017). *Toward Stream-Based IP Flow Analysis*. *IEEE Communications Magazine*, 55(7), pp.70-76.

Kafka (2017). *Apache Kafka*. [online] Available at: https://kafka.apache.org/intro [Accessed 12 Feb. 2018].

Karimi, A., Niyaz, Q., Weiqing Sun, Javaid, A. and Devabhaktuni, V. (2016). Distributed network traffic feature extraction for a real-time IDS. *2016 IEEE International Conference on Electro Information Technology (EIT)*.

Lee, Y. and Lee, Y. (2012). *Toward scalable internet traffic measurement and analysis with Hadoop*. *ACM SIGCOMM Computer Communication Review*, 43(1), p.5.

Lee, Y., Kang, W. and Lee, Y. (2011). A Hadoop-Based Packet Trace Processing Tool. *Traffic Monitoring and Analysis*, pp.51-63.

Platfora. *Big Data Discovery | Big Data Analytics | Platfora*. [online] Available at: https://www.platfora.com/ [Accessed 9 Feb. 2018].

Metron. *Apache Metron Documentation*. [online] Available at: http://metron.apache.org/documentation/ [Accessed 9 Feb. 2018].

MongoDB. *MongoDB for GIANT Ideas*. [online] Available at: https://www.mongodb.com/ [Accessed 12 Feb. 2018].

OpenSOC (2016). *Pycapa*. [online] Available at: https://github.com/OpenSOC/pycapa [Accessed 13 Feb. 2018].

Scapy (2016). *Scapy*. [online] Available at: http://www.secdev.org/projects/scapy/ [Accessed 14 Feb. 2018].

Spark-Streaming. *Spark Streaming - Spark 2.2.1 Documentation*. [online] Available at: https://spark.apache.org/docs/latest/streaming-programming-guide.html [Accessed 13 Feb. 2018].

Spark-Kafka. *Spark Streaming + Kafka Integration Guide - Spark 2.2.0 Documentation*. [online] Available at: https://spark.apache.org/docs/2.2.0/streaming-kafka-integration.html [Accessed 13 Feb. 2018].

Stallings, W. and Brown, L. (2015). *Computer security*. Boston: Pearson Prentice Hill.

Storm (2017). *Apache Storm*. [online] Available at: http://storm.apache.org/ [Accessed 14 Feb. 2018].

Terzi D. S., Terzi R. and Sagiroglu S. (2017) "*Big data analytics for network anomaly detection from netflow data"* 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, 2017, pp. 592-597.

Wullink M., Moura G., Muller M. and Hesselman C. (2016). ENTRADA: A high-performance network traffic data streaming warehouse. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*.