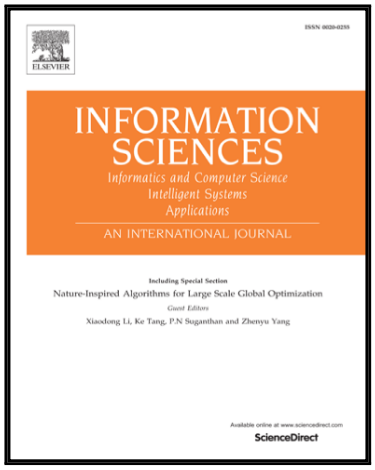
Journal Pre-proof



A Fully Scalable Big Data Framework for Botnet Detection Based on Network Traffic Analysis

S.H. Mousavi, M. Khansari, R. Rahmani

PII: S0020-0255(19)30970-3
DOI: https://doi.org/10.1016/j.ins.2019.10.018
Reference: INS 14934

To appear in: *Information Sciences*

Received date: 27 May 2018
Revised date: 3 October 2019
Accepted date: 12 October 2019

Please cite this article as: S.H. Mousavi, M. Khansari, R. Rahmani, A Fully Scalable Big Data Framework for Botnet Detection Based on Network Traffic Analysis, *Information Sciences* (2019), doi: https://doi.org/10.1016/j.ins.2019.10.018

A Fully Scalable Big Data Framework for Botnet Detection Based on Network Traffic Analysis

S.H. Mousavi[a], M. Khansari[a,*], R. Rahmani[b]

[a] *Faculty of New Sciences and Technologies, University of Tehran, North Kargar Street, Tehran, Iran*
[b] *DigikalaNEXT Research, Tehran, Iran*

Abstract

Many traditional Botnet detection methods have trouble scaling up to meet the needs of multi-Gbps networks. This scalability challenge is not just limited to bottlenecks in the detection process, but across all individual components of the Botnet detection system including data gathering, storage, feature extraction, and analysis. In this paper, we propose a fully scalable big data framework that enables scaling for each individual component of Botnet detection. Our framework can be used with any Botnet detection method - including statistical methods, machine learning methods, and graph-based methods. Our experimental results show that the proposed framework successfully scales in live tests on a real network with 5Gbps of traffic throughput and 50 millions IP addresses visits. In addition, our run time scales logarithmically with respect to the volume of the input for example, when the scale of the input data multiplies by 4x, the total run time increases by only 31 percent. This is significant improvement compared to schemes such as Botcluster in which run time increases by 86 percent under similar scale condition.

Keywords: Botnet Detection, Big Data, Hadoop, Spark, Machine Learning, Scalability

1. Introduction

The rapid growth and scale of current internet applications and the rise in network security threats have rendered many traditional network security methods ineffective[23]. Today, Botnets are among the most serious network security threats[27]. Each Botnet consists of a network of infected hosts called Bots that are usually distributed throughout the internet. Each Bot places a software agent in its infected host that is controlled by a unique person or business called the Botmaster[20]. The communication between Bots and Botmaster is known as Command & Control channels (C&C channels). Hosts are commonly infected without any awareness of their owners and so Botmasters use them to perform malicious activities throughout the Botnet network. These malicious activities may include Distributed Denial of Service (DDoS) attacks, spam emails, malicious adware, identity theft, sensitive information collection, malware distribution, and cyber terrorism[21, 15]. The rapid growth and scale of current internet applications and the rise in network security threats have rendered many traditional network security methods ineffective[23]. Today, Botnets are among the most serious network security threats[27]. Each Botnet consists of a network of infected hosts called Bots that are usually distributed throughout the internet. Each Bot places a software agent in its infected host that is controlled by a unique person or business called the Botmaster[20]. The communication between Bots and Botmaster are known as Command & Control channels (C&C channels). Hosts are commonly infected without any awareness of their owners and so Botmasters use them to perform malicious activities throughout the Botnet network. These malicious activities may include Distributed Denial of Service (DDoS) attacks, spam emails, malicious adware, identity theft, sensitive information collection, malware distribution, and cyber terrorism[21, 15].

*Corresponding author
Email addresses: s.hadi.mousavi@ut.ac.ir (S.H. Mousavi), m.khansari@ut.ac.ir (M. Khansari), r.rahmani@digikala.com (R. Rahmani)

1.1. Botnet Detection Method Categorization

There are multiple categorizations and taxonomies for Botnet detection methods[10, 1, 33, 25, 2]. Amini et al. presents a comprehensive taxonomy for Botnet detection methods[2]. Botnet detection methods can be categorized based on four aspects: the data source used in the detection, the type of data used, the classification method, and the type of detector interaction with the Botnet. Figure1 shows Botnet detection categorization and those methods which are focused in this paper.

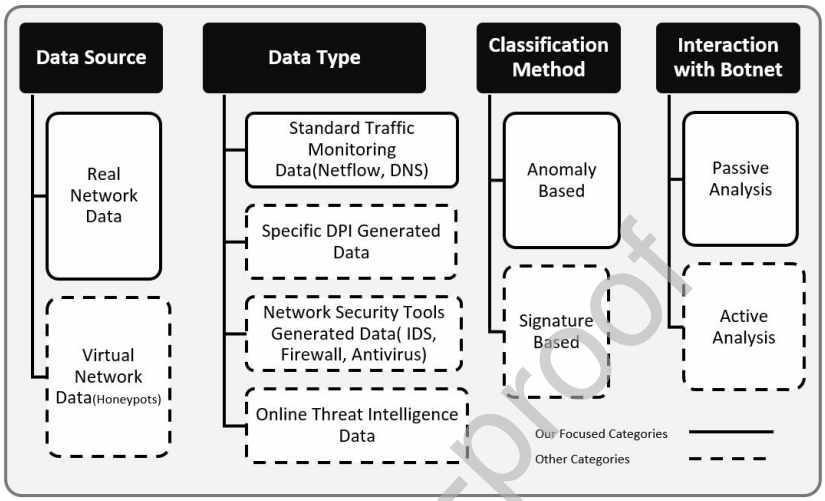


Figure 1: Botnet detection categorization based on four aspects

1. *Botnet Detection Method Categorization based on Data Source:* Based on the data source, Botnet detection methods can be categorized into two sub-categories: virtual network and real network data. Some methods use real network data while others use data which has been gathered through deploying a honeypot in the network. When real network data is used as a data source, the scale of network will vary from a local area network to a wide area network with multi Gbps traffic and millions of distinct hosts. As the scale of network becomes larger, the complexity and diversity of detection methods will get harder to tackle. Therefore, one of the most difficult problem to solve in Botnet detection methods is the scalability in real large networks. Hence in this paper we propose a framework for processing real large scale network data to deploy Botnet detection methods.
2. *Botnet Detection Method Categorization based on Data Type:* Based on the data type, Botnet detection methods can be categorized into four sub-categories: standard traffic monitoring data, Deep Packet Inspection (DPI) extracted data, network security tools generated data, and online threat intelligence. The first category includes a scheme that utilizes standard traffic monitoring tools such as Netflow and DNS data. The second category includes specific features which are extracted from network traffic directly by deploying a DPI method, while the third includes a scheme that utilizes logs generated by network security tools such as IDS, Anti-viruses, and Firewalls and so on. The last class includes a scheme that utilizes online threat intelligence data. Threat intelligence data is usually used as an auxiliary data. Employment of DPI methods for Botnet detection suffers two main drawbacks; the first is that processing user packets and payloads are against the privacy of users, and the second is that these schemes would not be applicable when network traffic is encrypted. Moreover, the requirement for processing of all packets in a network with high bandwidth is another challenge for these data types. Fortunately, applications of NetFlow and DNS data have become widespread recently as these are accessible in any protocol, which the Botnet uses, and are not affected by encryption. Hence applications of these data would be the best choice in a large scale network scenario, and the focus of this paper.

3. *Botnet Detection Method Categorization based on Classification Method:* Based on the classification method, Botnet detection methods can be categorized into two sub-categories: signature-based methods and anomaly-based method. In signature-based methods which are also known as misuse detection methods, detection is made based on known Botnet signatures. This means that the network traffic or extracted features that match with known Botnet signatures are classified as Botnet traffic. Signature based methods are not usually robust enough against minor changes in attack behavior. In contrast, the anomaly based methods are more resilient and try to learn malicious and benign traffic patterns. Botnet detection methods that are based on machine learning, statistical analysis and graph theory fall into this category. Hence, considering the advantages of these latter methods they enjoy more popularity in latest research studies[13]. Actually, there have been many activities in this field that leverage supervised and unsupervised[30, 31] as well as hybrid[16] and complex machine learning methods to detect Botnets[17, 14].
4. *Botnet Detection Method Categorization based on Interaction Type:* Based on the interaction type, Botnet detection methods can be categorized into two sub-categories: Passive Analysis and Active Analysis. The detector interaction with Botnet C&C or other bodies in the target network helps to reveal some good information in many cases. Hence active methods will try to collect this kind of information to improve Botnet detection performance. In passive methods detector tries to detect malicious activity without interacting with the network and does all things offline. In passive methods, detection is usually made by processing huge volume of network extracted data. Since in these methods detection is made without any interaction with Botnet bodies many evasion techniques[26] for attackers will not be effective. Given the advantages of passive detection methods, we will focus on these methods for the remainder of the paper.

1.2. Scalability in the Botnet Detection Methods

The staggering growth in the Internet usage and bandwidth on the one hand, and the complexity of the Botnet attacks on the other, has led to the realization that most published detection solutions are not effective on the high rate networks with millions of users and diverse applications. Given this situation two main functionalities are required to detect Botnets efficiently. It is important to note that using just one technique will not be sufficient to detect all types of Botnets and that several types of methods must be implemented at the same time. Moreover, the scalability for deployment in a real high rate network traffic poses a major challenge and this becomes more obvious in passive methods. To overcome these challenges, we propose a fully scalable big data framework which is highly effective in dealing with passive Botnet detection methods and can provide a solution to the scalability problem across all of the Botnet detection phases - including Data Gathering, Storage, Feature Extraction, and Analysis. The proposed framework can be applied concurrently to multiple detection methods which use any kind of network extracted data (especially DNS and NetFlow data) - including methods such as signature based detection, anomaly based detection, graph based detection, and machine learning based detection. The detection methods can be used alone or simultaneously in parallel.

1.3. Botnet Detection and Big Data

Big data attributes are commonly divided into the 4 Vs (i.e Volume, Velocity, Verity and Veracity)[32]. Botnet detection in large scale networks needs to process a huge volume of various data types that are created at almost high rate and speed. Therefore, in this paper, we will look at Botnet detection as a big data problem and use a big data approach to solve it. To Detect new complex Botnets, especially Advanced Persistent Threats (APT), we usually need to process historical network logs as these threats usually take a long period of time - in some cases up to two years, to infect hosts[6]. Since new and complex Botnets use some techniques to be more resilient[19] and usually hide their activity over a long period of time, our data storage and processing solution must take hiding duration into consideration. Additionally, storage of historical network traffic logs, at least NetFlow and DNS data, is a legal requirement for forensic purposes. Given these requirements, storing and processing these huge volume of data will require big data infrastructure. In summary, the main contribution of this work is to unveil a comprehensive framework for

Botnet detection that is fully scalable in all of its modules. The proposed framework that is based on big data approach, meets all the requirements of any Botnet detection method. This is advantage that makes the new framework suitable in large scale ISPs to deploy any kind of Botnet detection methods, concurrently.

1.4. Organization of this Paper

The rest of the paper is organized as follows: Section 2 explores related works on the scalability challenges in Botnet detection. Section 3 introduces our proposed scalable big data framework for Botnet detection. Section 4 presents our evaluation methodology and experimental setup using real world NetFlow data. Section 5 describes the results of our experiments. Section 6 presents our conclusions and future work.

2. Related Work

Scalability is one of the most important challenges that Botnet detection methods are faced with and is thus the main focus of many of the latest studies in the big data sphere[24, 12, 22]. Soltanaghaei et.al, for example, used the Bloom filter technique to reduce volume of data that had to be processed[24]. It must be pointed out that, reducing data volume results in scalability in the processing stage alone and does not deal with the scalability problem that also exists in the data gathering and storage phases. Note also that this particular technique could only be applied to their own method and, therefore, is unsuitable as a base technique for all detection methods, because historical data is required for some detection methods. The Bloom filters other limitation is that the real data does not get stored which, therefore, makes this solution unsuitable for forensic purposes. Another scalable Botnet detection method studied by Kwon et al. was PsyBoG[12]. In PsyBog, authors used a signal processing technique called PSD to detect periodic DNS request patterns that usually occurs in fast flux network and Botnet communications. Detecting hosts that have periodic DNS requests is the first step in detecting all infected hosts that attend in specific Botnet. Scalability in PsyBog was achieved through low overhead computation using only three features, namely; request time, IP of host sending request and the domain requested. As this method uses few features to derive scalability in the processing phase compared to other DNS based Botnet detection schemes, it could still be applied to large scale networks. It is important to note that the PsyBog solution does not provide scalability in data gathering and storage phases for forensic purposes and that it only considers one detection method.

It should be borne in mind that the huge volume of network extracted data that must be processed and the high velocity of data in large networks with multi Gbps bandwidth, are just two common features with any big data landscape. Therefore, implementation of the big data approach in Botnet detection could lead to an effective solution in such scenarios which is what Singh et al. adopted for peer to peer Botnet detection[22]. They revealed a Hadoop based framework that had three modules. The first was Traffic Sniffer which was responsible for preprocessing packets. The second was for feature extraction, and the third was for machine learning and detecting malicious traffic. The traffic sniffer module captured network packet and saved raw data into HDFS. The feature extraction module used Hive query engine for its task and the third module did the last step with Mahout Library. Based on the results in[22], Maximum traffic rate that the sniffing module can support is less than 200 MBps. So this module too is not suitable for real multi Gbps traffic. Since in their proposed framework there is no queuing system the framework could not support burst traffics and the sniffing module is the main bottleneck of the framework. Also, since in the second module, they used Hive and Group by queries in HQL, performance is heavily dependent on the storage format and compression methods. Our experiment shows that using columnar storage formats, such as Parquet, results in 20x better response time in such queries, as illustrated in Section3.6 The last module in[22] uses Apache Mahout[3] library. Although the Mahout library is a scalable Machine Learning library and is based on MapReduce platform, many researches have shown that performance of Spark based Library, i.e. SparkML, is superior to that of Mahout[18]. Better performance in Spark based Library is obtained as a result of proper job scheduling in Yarn compared with high latencies in the MapReduce model.

Another recent research that made use of MapReduce processing platform to handle big data challenges in Botnet detection devoted itself to detecting P2P Botnet[30]]. Authors in[30] applied their method to a large

Table 1: Recent related works on scalable Botnet detection

Method	Type of Scalability	Limitation
BotCluster: A Session-based P2P Botnet Clustering System on NetFlow[30],2018	storage and processing	• Process NetFlow data in Offline mode and did not consider online data processing • Does not make use of all big data potential capabilities to provide best scalability and speedup • Just affords to run single Botnet detection method
PsyBoG: A Scalable Botnet Detection Method for Large-scale DNS Traffic[12],2016	processing	• Lack of scalability in data gathering and storage phases • Does not store real DNS requests and just stores timing features of each DNS request. So, that is not applicable for forensic purpose • Just affords to run single Botnet detection method, so, it is not comprehensive and is not applicable for other detection methods
Detection of fast-flux Botnets through DNS traffic analysis[24] ,2015	processing	• Lack of scalability in data gathering and storage phases • Using Bloom Filter led to this method does not store all data and it is not applicable for complex and long living Botnets, especially those in APTs. • Just affords to run single Botnet detection method, is not comprehensive and is not applicable for other detection methods
Big data analytics framework for peer-to-peer Botnet detection using random forests[22],2014	storage and processing	• Limitation in sniffing module only up to 200Mbps network rate can be held in this module • Does not make use of all big data potential capabilities to provide best scalable framework • Just affords to run single Botnet detection method
DISCLOSURE: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow Analysis[4],2012	without scalability	• Requires huge computation and processing for large scale data sets specially in the feature extraction phase

Hadoop cluster with 256 processing node and 4096 cpu cores and more than 16TB memory. This approach was based on NetFlow data and clustering algorithm for P2P Botnet detection. They imported NetFlow raw data into HDFS and implemented feature extraction, Wight-list and blacklist filtering and clustering algorithm using MapReduce on the Hadoop cluster. Although they made use of a scalable platform for storage and processing, the proposal lacked scalability in high Data velocity scenarios and in processing real time network traffic. It will be shown in the evaluation section that our proposed framework outperforms this method.

As mentioned above, Passive Botnet detection methods offer some benefits over active methods. Among passive Botnet detection methods, Bilge et al. introduced a C&C server detection method through large scale NetFlow analysis, called Disclosure[4]. Although Disclosure uses good features selected from NetFlow data that enables it to detect C&C servers with acceptable precision if implemented in large scale network, it could not be deployed in large scale ISPs. This is because it fails to tackle scalability, especially in processing NetFlow data, to extract selected features for the random forest machine learning algorithm that it uses. Since the main idea of the Disclosure method was novel, it has become a research reference in the literature which has been cited 150 times in other researches till now and it is also the choice in our proposed framework for evaluation.

In many previous works, there has been not much focus on scalability in high rate network bandwidth, and most studies on Botnet detection and scalability overlook this key challenge. Scalability could be classified into two main classes; processing scalability and storage scalability. Those methods that provide scalability in large scale deployment by reducing processing overhead or by using parallel processing techniques offer processing scalability. In many Botnet detection methods the goal is detection itself and storing network extracted data for forensic or other long term analysis is not considered, despite the fact possibility that long term storage might be needed to detect some kinds of Botnet attacks. Those methods that do take long term storage and analysis into account are classified as storage scalable detection methods. It should be noted that processing scalability does not result in storage scalability and vice versa. Table1 summarizes recent works on scalable Botnet detection.

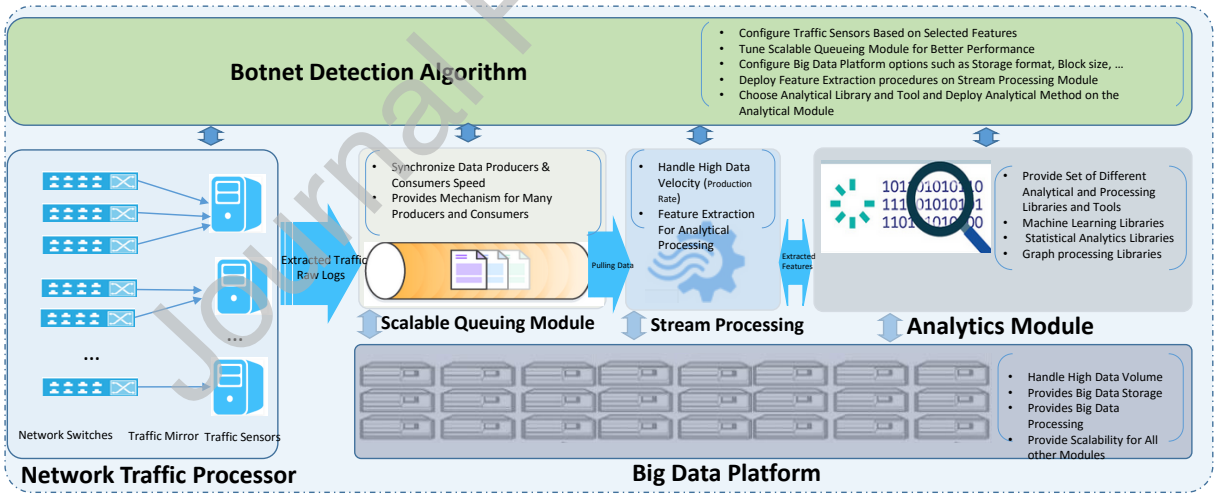


Figure 2: Architecture of the proposed Botnet detection framework

3. Proposed Scalable Big Data Framework for Botnet Detection

Scalability must be considered in all phases of detection from network sniffing and data collection to feature extraction and deployment of any kind of analysis such as machine learning, graph based analysis and statistical analysis. Figure2 displays the general architecture of the proposed framework which can

collect all needed data from monitored network gateway, is designed for passive analysis, and is also suitable for deployment in large ISPs or any enterprise scale organization that has many Internet users

3.1. Network Traffic Processor Module

As the name suggests, this module processes network traffic, and based on data requirement in the detection method, raw network data is extracted from the received traffic. Some common data items that usually are used in Botnet detection methods are placed here which include DNS traffic extractor, Http traffic extractor and NetFlow traffic extractor. The selected tools must do traffic processing at line rate and must also process input traffic in parallel. In this module only raw data is extracted without any overhead and selected tools could use some common parallel network traffic processing tools such as PFring[8]. In addition to the software based parallel network processing tools, application of a high performance hardware such as Napatech[19] which is developed specifically for high rate network traffic processing in the NIC processors is quite effective.

3.2. Scalable Queuing Module

The second modules task is to synchronize data producers and consumers. Since there must be many data producers and consumers to process raw data in parallel, this module provides a distributed queuing system to solve the problem. With this module the availability of framework will not be affected even if the subsequent modules is not ready or is congested. There could be many data producer sensors distributed across the network where each processes just a portion of the whole network and sends its raw data to the single queuing system. This module will aggregate all raw data in one queue and provides an infrastructure for parallel processing in the next module. In this module distributed queuing systems such as Apache Kafka[9] and Redis[5] could be used.

3.3. Stream Processing Module

The queuing module acts like a kind of intermediate module which receives and holds raw data without any processing. All Received data must be processed and stored in the scalable data store. These tasks are carried out by the third module. More specifically, this module has two main duties to discharge. First, it must process the received data, then clean and parse them, and finally store them in a scalable big data storage (such as HDFS). Second, it must extract features which is performed by the detection method. Extracted features are then used in the analytics module for any detection method.

3.4. Big Data Platform

As mentioned in the Introduction Section1.3, one requirement for detecting APT and new complex Botnets is to store and process network activities in big time window. Storing network activities data is also needed for forensic purposes. With this in mind, the Hadoop platform is selected for data storage and processing. It should be noted that Hadoop is scalable and now also a de facto platform for big data with many analysis tools based on it that could be used in other modules in the proposed framework.

3.5. Analytics Module

After selecting the best storage format and compression method based on input data, preprocessed and formatted data are stored on the scalable big data platform. The next step in scalable Botnet detection framework is to provide various tools and facilities for different Botnet detection methods. As mentioned in the Introduction section1.1, Botnet detection methods fall into three categories based on analytic scheme they use which are Statistical methods, machine learning methods and graph based methods. In the proposed framework all tools must be available for deployment depending on the selected analytic scheme which is required in Botnet detection method.

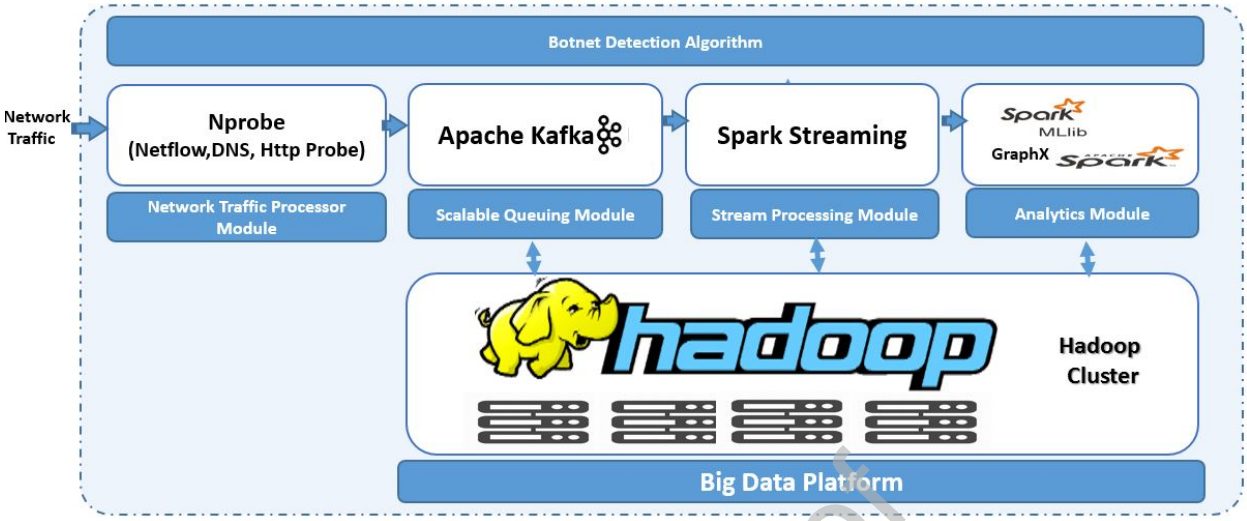


Figure 3: Tools and technologies deployed in each module in the proposed Botnet detection framework.

3.6. Technology Mapping for each of the 5 Modules in the Proposed Framework

The proper choice of tools and technology can significantly affect the overall performance and flexibility of the proposed framework. In addition, all tools must provide adequate scalability so as to avoid bottlenecks in the total framework. Figure3 shows the selected tools in each module.

As mentioned above, the first module is responsible for gathering data from the network traffic, so both software and hardware probes could be used to gather NetFlow data. *Nprobe*[7] is a good software based probe that can generate NetFlow records of up to 100Gbps network throughput by means of *Pfring* library. This tool can also extract DNS and Http metadata that might be needed for other Botnet detection methods.

The second module provides a queue to synchronize producer and consumers. Since Apache Kafka is a scalable and flexible message passing tool, it was selected for this purpose. The nprobe tool also has a *Kafka* export plugin that could export flow records into Kafka cluster in text format.

The third module must consume the produced data at high speed and store them in HDFS storage. Stream processing tools are responsible for processing stream of incoming data at high speed and in a scalable manner. Tools such as Apache Storm and Spark Streaming could be used in this module. To provide consistency in the overall solution the Spark streaming library is selected for this module which is responsible for parsing and storing extracted data in the storage platform.

The forth module in our framework is the Big Data platform that we choose Hadoop platform. There are some key considerations that affect performance of the overall framework. Note that Data storage format and compression affect performance of all other modules considerably, as was confirmed by our initial result evaluation. Usually the bottleneck in big data processing is the I/O and the speed in transferring data from hard disk to main memory. Performance improvement obtained from compression is the result of less data volume swapped between main memory and disks. In addition to compression, proper choice of data storage format based on the processing requirements as well as the data structure do improve performance in big data platform. As the NetFlow and DNS are structured data and usually all processing methods undergo many aggregations and statistical operations, the best choice is the columnar storage format. Apache Parquet is an efficient, structured, columnar storage, compressed, binary file format which supports several compression codecs[29]. With this architecture there could be many consumer processes where each of them processes a portion of input records and stores the preprocessed and parsed data into HDFS parquet files.

Finally, for the fifth module which is the Analytical module, depending on the other modules and the Botnet detection method, a portfolio of tools could be used in the analytics phase. For example, for machine learning algorithms application of SparkML is recommended. It is possible to use apache Spark and R on

Table 2: Machines specifications in the deployed framework

Machine Role	Number of Machines	Number of CPU Core/Machine	Main Memory/Machine
Hadoop Master Services	2	2*14(*2 HT*)	128GB
Haoop Slave Services	12	2*10(*2 HT)	128GB
Apache Kafka	1	2*14(*2 HT)	128GB
Nprobe Sensor	1(VM)	8(virtual core)	20GB

* Hyper Threading

Hadoop for implementing any kind of statistical processing. And finally, apache Giraph or GraphX Spark library is an efficient choice for graph based methods.

4. Experimental Setup and Results - Using Real World NetFlow Big Data

To evaluate the proposed framework we employed the *Disclosure* [4] model which is put forward by Bilge et.al and is in the forefront of research in this field. The main idea in this model is to detect C&C servers based on NetFlow features and machine learning algorithm. Let us see how this is made to work in practice: It begins by detecting server IPs based on the Network behavior. Next, desirable features are selected and then extracted from NetFlow data. At this point, a training data set is created based on Threat Intelligence of known C&C and Botnet IPs along with the known white-list IP. Finally C&C servers detection model is created by using Random Forest algorithm. Since machine learning based Botnet detection algorithms are the most effective methods and the *Disclosure* is one of the key topic of research in this area we chose to implement this method to evaluate our solution.

4.1. Deployment Environment

The whole framework was implemented over an Internet Service Provider environment in Tehran and all network data were gathered from probes that took mirror of traffic after the ISP Gateway. In addition, a Hadoop cluster with 15 servers was deployed together with Ambari and HDP-2.6.2 distribution of Hortonworks. Table2 illustrates resources in each server as well as the role of each server in the framework.

The network traffic throughput in the test environment was on average 5 Gbps, and also there was on average 46 millions distinct destinations IP visits per day by clients in the network. Figure4 shows several statistics on NetFlow records and unique IP count of our landscape over 10 days between November 20, 2017 and November 30, 2017. This figure shows the scale of data that had be processed in each day, and Figure 5 shows data volume over each day during the evaluation period.

4.2. Gather and Import NetFlow Data

The first step in implementing the Disclosure method over the proposed framework was to gather NetFlow data and import raw data into the Hadoop distributed file system. One Nprobe sensor was employed in the network gateway which collected NetFlow data and exported flow records to the Kafka in text format. Another module that was responsible for parsing and storing extracted data into HDFS parquet file was developed based on Spark streaming library. With this architecture there could be many consumer processes where each consumer processed a portion of input records and stored the preprocessed and parsed data into HDFS parquet files. Application of Spark streaming provided scalability in the parsing and loading phase of data processing. Thus there was not any bottleneck here in this phase and full scalability was attained.

As mentioned in Section3.6, proper selection of appropriate storage format and compression improves performance. Our investigation verified that in the same Hadoop cluster, using *Parquet* storage format with compression had improved performance significantly. During another experiment, We extracted NetFlow and DNS data from a network with just 300 Mbps traffic over one week and then loaded it into a Hive based table and ran two query on them. First query was a simple *count(*)* on each table and the second one was an aggregate count based on source IP. Table3 depicts the scale of such data set. We conducted experiments under two scenarios; one without compression and using text storage format and the other with snappy compression and using parquet storage format. Table 4 shows the compression and storage format effects on response time for each of these two kink of queries.

Figure 4: NetFlow data scale over 10-day period - experiment.

Table 3: NetFlow and DNS data volume in a 300 Mbps throupout network over one week

Data Type	Number of Records	Raw Data Volume	Compressed Data Volume
NetFlow	1,036,504,923	114GB	28.5GB
DNS	775,942,634	98GB	29.8GB

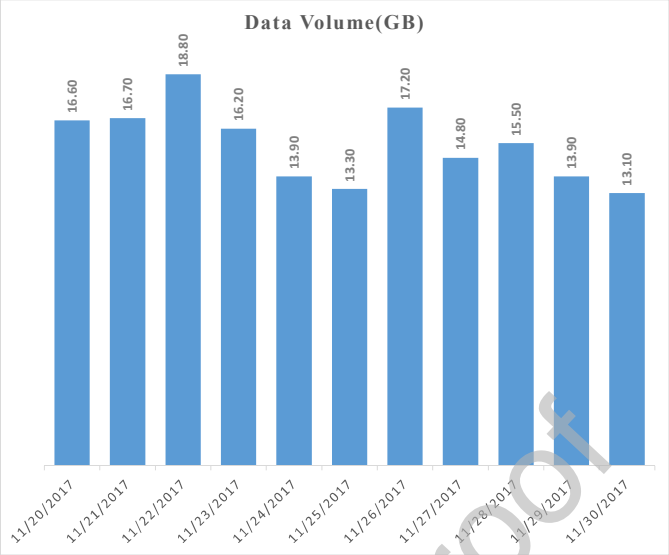


Figure 5: NetFlow data volume over 10-day period - experiment.

Table 4: Compression and storage format effects on response time for different queries on one week network big data

Data Type	Response Time Without Compression with Text Format		Response Time Using Compression Parquet Format	
	simple query	complex query	simple query	complex query
NetFlow	576.77s	18.14s	612.34s	87.86s
DNS	494.53s	5.95s	471s	59s

4.3. Feature Extraction Based on Detection Algorithm

The next step in deploying Botnet detection algorithm dealt with feature extraction which was the most resource consuming and complicated phase. In this phase all extracted data had to be processed for a given time window (usually one day) to extract some features. Usually extracting each feature has its own complexity. To detect C&C servers, first, Server IPs must be extracted. IPs that had at least 90 percent of their received flows destined their two top ports were selected as Servers IPs. Next, for each server IP selected features had to be extracted from the stored NetFlow records.

4.4. NetFlow Based Selected Features

Based on selected C&C server detection method some features had to be extracted for each Server IP. Table5 tabulates selected features and summarizes process of feature extraction.

4.5. Deploying Feature Extraction Using Hive and User Defined Functions

Processing huge number of data records to extract and compute selected features is quite cumbersome in any machine learning algorithm especially where big data is involved. To overcome this challenge in the proposed framework, Hive queries were used. Since raw NetFlow data were stored in parquet files, Query engines such as Apache Hive on Tez, had to be used without any overhead which avoided developing a MapReduce program. Given that, extraction of complicated features such as client access pattern is beyond basic Hive queries we introduced a new solution. This solution allowed us to develop some user defined

Table 5: Extracted features for each Server IP

Feature Category	Extraction Process	Extracted Features	Number of Features
Statistical flow size features	Aggregate flows for each server IP and compute statistical feature for each server IP	Mean and std.[a] of flow size for received and sent flows for each Server IP	4 feature
Unique Flow Size	Count Unique size of each flows for each Server IP	Median and Mode of flow size for received and sent flows for each Server IP	4 feature
Client Access Pattern	Create sets of flows for each s_i and c_j(from s_i to c_j) compute enter arrival time between each tow sequence flows for each pair si,c_j and compute min, max, mean and std. for each pair and for each s_i compute statistical features from each of the above.	Min, Max, Mean and std. for each pair and for each s_i compute statistical features from each of the min, max, mean , std. values	16 feature
Unmatched Flow Density	Computing unidirectional flow count for each server IP	Mean and std. of number of unidirectional flow for each Server IP	2 feature
Timing Features	Partition each day into 4 time periods. Compute number of flow for each server IP over each time period then compute mean and std. for each Server IP over each time period	Mean and std. of number of flows for each server IP in each of 4 time periods(0 to 6, 6 to 12, 12 to 18 and 18 to 24)	8 feature

[a] Standard Deviation

aggregate functions based on Hive Java API and led not only to improved feature extraction performance but reduced computation complexity. Another technique that also improves performance utilizes some intermediate Hive tables.

The flexibility of basic Hive functions together with the benefit of application of User Defined Functions make the process of any feature extractions from any traditional Botnet detection methods possible and results in desirable scalability.

4.6. Preparing Data Set for Training and Testing

Data set creation consists of two main parts. The first part prepares a list of known C&C IP addresses and the second part prepares a list of benign IP addresses. For the first part some of the free IP blacklists is used and for the white-list IP list we obtain popular IP based on the top 500 Alexa [11] list and also select the top 10000 Cisco Umbrella domains [28]. To resolve IP list from Domains, a batch script was created and executed through one machine in the same network, then the white-list IP is extracted. Training data set was created for one day (on 20th Nov. 2017) from the NetFlow traffic mirror in our experiment environment. There were flow records from 472 unique C&C IPs and 2111 unique benign IPs in the server IP list obtained on that day. We extract features and build our train and test data sets from these server IP addresses for that day. The Random Forest algorithm was deployed using SparkML library and the prepared data sets.

4.7. Deploying Machine Learning Algorithm with SparkML

To provide scalability in all modules in the proposed framework, the latest Hadoop based machine library was used. Using SparkML and Java API the machine learning phase of the C&C Server detection was deployed. SparkML outperforms other libraries such as Apache Mahout because it employs Yarn instead of MapReduce which reduces the task scheduling overhead in the basic MapReduce platform [18].

5. Experimental Results

The main objective is to evaluate the scalability of the proposed framework and the techniques used in C&C server detection deployment. Since scalability is provided in each module, from data gathering to storage and feature extraction and finally machine learning, we can say that the proposed framework provides full scalability. We have also shown that the most complex phase of Botnet detection is feature extraction.

In the test cluster, this phase took about 17 minutes to process the entire data gathered over one day time window. In contrast, in the Disclosure method [4], feature extraction for a data series amounting to 1.6 billion flow records and 78 million unique IP addresses which the processing phase took almost 11 hours. To compare our results with those of the Disclosure method, we generate similar data set size by three days sequence of traffic in which sum of flows is near to the Disclosure case and total unique IP is more than twice that of the Disclosure data set (179 million unique IP in three days of our data set). In this case, feature extraction took just 1299 seconds. Table 6 summarizes these results. It should be noted that the Disclosure method ran its experiments in single server and did not provide any scalability.

Table 6: Scale and feature extraction duration, three days sequence of Our ISP data set vs Disclosure[4] data set

Criterion	**Disclosure[4]**	**Our ISP data set**
Flow Record	1.6 Billion	1.3 Billion
Number of Unique IP	78 Million	179 Million
Feature Extraction Duration	11Hours	1299 Second

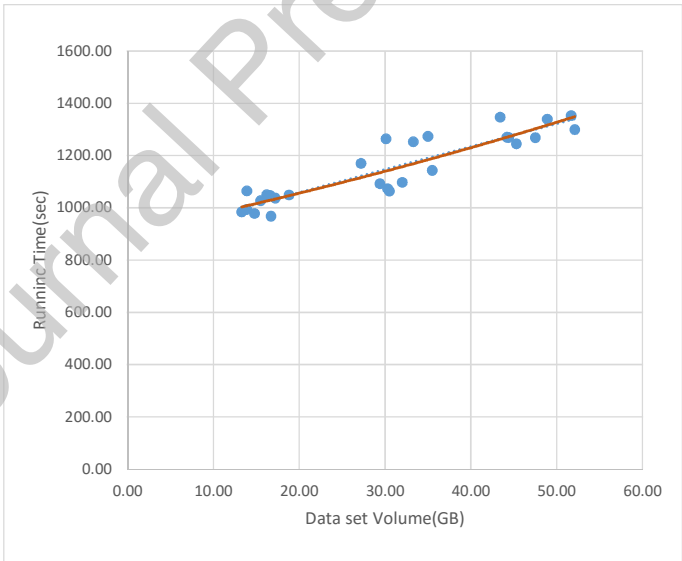


Figure 6: Scalability in feature extraction phase in Our framework- 4 times larger data input and just 31% time overhead

To assess scalability of the proposed framework we increased flow data and measured the time taken for the feature extraction phase. Figure 6 displays the result of this simulation. Note that the volume of compressed NetFlow data in the HDFS storage for just one day is 15.7GBs on average. Extracting features from flows over a number of days is the same as that for one day with corresponding bigger scale. Average processing time for different processing time windows comes in Table 7. Bearing this in mind, we scaled our input and computed the running time for feature extraction which demonstrated that when input data

Table 7: Average processing time for different processing time window in feature extraction phase in Our framework

Time window	Average Input data Size(GB)	Processing Time(s)
1 Day	**15.69**	**1020.35**
2 Day	**31.48**	**1159.14**
3 Day	**47.19**	**1299.17**

Table 8: Comparison of the scalability in our proposed framework vs. the Botcluster[30]

Criterion	**Botcluster[30]**	**Our Proposed framework**
Hdoop Cluster Resources	256 Processing node with 4096 cpu core and 16TB memory	14 Processing node with 268 cpu core and 1.75TB memory
Processing overhead for near 4x larger input data volume (percent)	86% increase in processing time	31% increase in processing time
Other conditions	increasing input data volume from 10GB to 40GB	increasing input data volume from 13.3GB to 52.1GB

increased almost fourfold (3.92), the running time just increased 31 percent on average. This proves that the proposed framework provides linear scalability.

The machine learning phase for one day just took 15.7 seconds on average which beats [30] where increases in input data volume from 10GB to 40GB causes the total processing time to go up from 942 seconds to 1749 seconds (a huge rise of 86 percent). Note that the experiment in Hadoop cluster consisted of 256 nodes with 4096 cpu cores whereas in our deployment we had only 14 similar nodes with only 560 cpu cores. Table 8 tabulates these results.

Our last evaluation focused on machine learning performance. Using Gini impurity criterion and Random Forest learning algorithm and cross validation evaluation method, we obtained 73 percent for detection rate and just 2 percent for false positive rate. The results confirmed that detection rate and false positive rate for the created data set was closely comparable to those for the Disclosure approach. This proves, therefore, that the proposed deployment was successful, as illustrated in Table 9.

Table 9: Machine learning performance in Disclosure[4] vs. Our deployment

Criterion	Disclosure [20]	our Deployment
Detection Rate	64%	72%
False Positive Rate	1%	2%

6. Conclusions

Recent growth in network scale and Botnet complexities are challenging Botnet detection and security countermeasures in scalability. Published studies to date have not come up with effective solutions on scalability and deployment bottleneck in bigger networks. Authors in this paper proposed a new framework that was based on the Hadoop platform and provided scalability with a comprehensive framework to deploy all kinds of network based Botnet detection methods especially in big networks. The proposed framework was implemented in a large ISP network with near 5Gbps traffic and experimental results confirm the scalability and usability of the proposed framework. Despite its advantages, the proposed framework has its drawback. Providing enough computational resources which is required to install the proposed framework is

not affordable for smaller enterprises. This includes total cost of ownership along with skillful human resource to deploy this framework. However, using cloud service providers to provide computational infrastructure for big data framework deployment would overcome these challenges. Furthermore, it is possible to provide security as a service from managed security service providers (MSSP) where MSSPs deploy this framework to provide Botnet detection service for their customers. Since the proposed scheme was successful in deploying Botnet detection method, for the future works we decided to deploy our new passive Botnet detection technique to detect FastFlux networks using NetFlow and DNS extracted features in the proposed framework. Moreover, this model is a general framework with general modules needed in any Botnet detection methods, Botnet detection methods which suffer from scalability could be incorporated into it for large scale networks.

Declaration Of Competing Interest

We the undersigned declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere. We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome. We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us. We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property. We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from INS@elsevier.corn.

References

References

[1] K. Alieyan, A. ALmomani, A. Manasrah, M. M. Kadhum, A survey of botnet detection based on DNS, Neural Computing and Applications 28 (7) (2017) 1541–1558.

[2] P. Amini, M. A. Araghizadeh, R. Azmi, A survey on Botnet: Classification, detection and defense, in: Electronics Symposium (IES), 2015 International, IEEE, ISBN 1-4673-9344-4, 233–238, 2015.

[3] Apache Mahout, URL `https://mahout.apache.org/`, 2018.

[4] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, C. Kruegel, Disclosure: detecting botnet command and control servers through large-scale netflow analysis, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACM, ISBN 1-4503-1312-4, 129–138, 2012.

[5] J. L. Carlson, Redis in action, Manning Publications Co., ISBN 1-61729-085-8, 2013.

[6] J. Chen, C. Su, K.-H. Yeh, M. Yung, Special Issue on Advanced Persistent Threat, 2018.

[7] L. Deri, N. SpA, nProbe: an open source netflow probe for gigabit networks, in: TERENA Networking Conference, 2003.

[8] S. Gallenmller, P. Emmerich, F. Wohlfart, D. Raumer, G. Carle, Comparison of frameworks for high-performance packet IO, in: Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on, IEEE, 29–38, 2015.

[9] N. Garg, Apache Kafka, Packt Publishing Ltd, ISBN 1-78216-794-3, 2013.

[10] T. Hyslip, J. Pittman, A Survey of Botnet Detection Techniques by Command and Control Infrastructure, Journal of Digital Forensics, Security and Law 10 (1) (2015) 7–26.

[11] Keyword Research, Competitor Analysis, & Website Ranking | Alexa, URL `http://www.alexa.com/`, 2017.

[12] J. Kwon, J. Lee, H. Lee, A. Perrig, PsyBoG: A Scalable Botnet Detection Method for Large-scale DNS Traffic, Computer Networks .

[13] A. H. Lashkari, G. D. Gil, J. E. Keenan, K. Mbah, A. A. Ghorbani, A Survey Leading to a New Evaluation Framework for Network-based Botnet Detection, in: Proceedings of the 2017 the 7th International Conference on Communication and Network Security, ACM, 59–66, 2017.

[14] S. Lysenko, K. Bobrovnikova, O. Savenko, A botnet detection approach based on the clonal selection algorithm, in: 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 424–428, 2018.

[15] M. Mahmoud, M. Nir, A. Matrawy, A Survey on Botnet Architectures, Detection and Defences., IJ Network Security 17 (3) (2015) 264–281.

[16] L. Mai, D. K. Noh, Cluster Ensemble with Link-Based Approach for Botnet Detection, Journal of Network and Systems Management 26 (3) (2018) 616–639, ISSN 1064-7570, 1573-7705, URL http://link.springer.com/10.1007/s10922-017-9436-x.

[17] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, N-BaIoTNetwork-Based Detection of IoT Botnet Attacks Using Deep Autoencoders, IEEE Pervasive Computing 17 (3) (2018) 12–22, ISSN 1536-1268, 1558-2590, URL https://ieeexplore.ieee.org/document/8490192/.

[18] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, Mllib: Machine learning in apache spark, The Journal of Machine Learning Research 17 (1) (2016) 1235–1241.

[19] Napatech | Zero Packet Loss, URL http://www.napatech.com/, 2016.

[20] C. Schiller, J. R. Binkley, Botnets: The killer web applications, Syngress, ISBN 0-08-050023-4, 2011.

[21] S. S. Silva, R. M. Silva, R. C. Pinto, R. M. Salles, Botnets: A survey, Computer Networks 57 (2) (2013) 378–403.

[22] K. Singh, S. C. Guntuku, A. Thakur, C. Hota, Big data analytics framework for peer-to-peer botnet detection using random forests, Information Sciences 278 (2014) 488–497.

[23] M. Singh, M. Singh, S. Kaur, Issues and Challenges in DNS based Botnet Detection: A Survey, Computers & Security .

[24] E. Soltanaghaei, M. Kharrazi, Detection of fast-flux botnets through DNS traffic analysis, Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical 22 (6) (2015) 2389.

[25] S. Soltani, S. A. H. Seno, M. Nezhadkamali, R. Budiarto, A survey on real world botnets and detection mechanisms, International Journal of Information and Network Security 3 (2) (2014) 116.

[26] E. Stinson, J. C. Mitchell, Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods., WOOT 8 (2008) 1–9.

[27] The Facts about Botnets | Malwarebytes Labs, URL https://blog.malwarebytes.org/cybercrime/2015/02/the-facts-about-botnets/, 2016.

[28] Umbrella Popularity List (Top Million Domains) Umbrella Investigate Rest API, URL https://investigate-api.readme.io/docs/top-million-domains, 2018.

[29] D. Vohra, Apache parquet, in: Practical Hadoop Ecosystem, Springer, 325–335, 2016.

[30] C.-Y. Wang, C.-L. Ou, Y.-E. Zhang, F.-M. Cho, P.-H. Chen, J.-B. Chang, C.-K. Shieh, BotCluster: A session-based P2P botnet clustering system on NetFlow, Computer Networks 145 (2018) 175–189, ISSN 13891286, URL https://linkinghub.elsevier.com/retrieve/pii/S1389128618308351.

[31] W. Wu, J. Alvarez, C. Liu, H.-M. Sun, Bot detection using unsupervised machine learning, Microsystem Technologies 24 (1) (2018) 209–217.

[32] C. Yang, Q. Huang, Z. Li, K. Liu, F. Hu, Big Data and cloud computing: innovation opportunities and challenges, International Journal of Digital Earth 10 (1) (2017) 13–53.

[33] S. Zhou, A Survey on Fast-flux Attacks, Information Security Journal: A Global Perspective 24 (4) (2015) 79–97.