Introduction

WIth the growth of the internet, related security risks has also grown. There are newer and innovative attacks beings carried out daily to breach the user data, privacy and even unauthorised access to internet connected devices. Everyday more and more devices are getting connected and with the advent of Internet of Things this rate has exploded exponentially. To address this problem Intrusion Detection systems are usually deployed in the Information Technology architecture of the organisation. Intrusion detection system can be classified in two parts namely Signature based detection and Anomaly based detection. Signature based detection methods make use of pattern matching of incoming traffic with the predefined rules and signature, If the predefined signature matches with the incoming traffic data then the system identifies it as an intrusion. Anomaly based detection works by observing the normal traffic behaviour and then it sets a baseline for determining the normal traffic. If incoming traffic deviates from the normal, it creates an alarm for probable intrusion. The term probable intrusion is used as there is a chance for false positive in anomaly based detection methods. After the alerts are generated a security personal reviews the logs and take appropriate actions. Anomaly based detection are mostly statistical models which many a times cannot capture the novel attacks. In order to tackle this problem we are proposing a new Intrusion detection system for detecting Botnets based on Machine learning and Neural Network. We applied Machine learning models to detect Botnets in a group of computers. By this method even the patterns which are unknown can be detected, as the neural network learn the underlying complex pattern of the attack and not just the superficial features. This proposed architecture could be transformed into a simple to use router that will sniff the data and send it to a server running the machine learning algorithm and send back the results obtained for further action.

Abstract

We propose various models based on machine learning and neural network for botnet detection. We passively monitor the data of any group of computer and feed the netflows into various trained models (which we will describe in this report) and get the labeled Internet protocol addresses as either malicious or normal. This system can be used in any business environment with enough number of computer. The proposed models will run on the cloud and the data can be transferred to the remote cloud server, analysed, predicted and information can be retrieved back.

2. Literature Review

Researchers have been working for Botnet detection since the discovery of automatic Spam messages. Their efforts in Botnet detection has led to a vast amount of proposals that analyse this problem from multiple perspective. The purpose of this Literature Survey is to get an overview of the studies done in this field, their datasets and results.

2.1 Previous botnets detection methods

2.1.1 Botsniffer

In BotSniffer Paper [1] they have presented three ways for botnet detection. The first two Detection methods share a common base, sniff the network packets and group together host that connects to the same remote server and port and separate these group in time windows. After this data is collected, the first approach looks for any evidence of SPAM sending or some other attack fingerprint such as executable downloading or open port scanning. If more than half of total computers shows the signs of being infected than that group is said to be a possible botnet group. Then sequential probability ratio testing algorithm is used to decide if it is a botnet or not. The second method looks for the host that answered similar IRC protocol responses using DICE distance as a similarity function.

Then the IRC responses are clustered on the basis of similarity measure. If the biggest cluster is more than half the size of the group the that group is marked as a possible botnet. After this sequential probability ratio testing is used again to decide whether the identified group is really a botnet. This is one of the most cited papers in the botnet detection field. It presents several behavioral techniques to detect botnets that

accomplish good results. However, much new data and botnets have been found since its publication. The dataset used for validation may be too scarce for generalizing the technique. Only one real IRC botnet was captured. The rest of the dataset is composed of one IRC text log and five custom-compiled LAN botnets. The dataset was verified using three methods, but the first two methods were not effective. The first verification was under the assumption of a clean normal capture because of a well-administered network.However, even well-administered networks can have infected computers. The second was the use of the same BotSniffer method over the dataset. It is commonly not considered a good practice to verify a dataset using the same proposed method. The third verification was performed using the BotHunter [2] proposal. This was a good verification. However,it was only used over the normal captures. It was also assumed that the TCP would be the primary protocol used for C&C channels. However, it could be important to add the UDP or Internet Control Message Protocol as well. The dataset used was not made public, perhaps because of privacy issues.

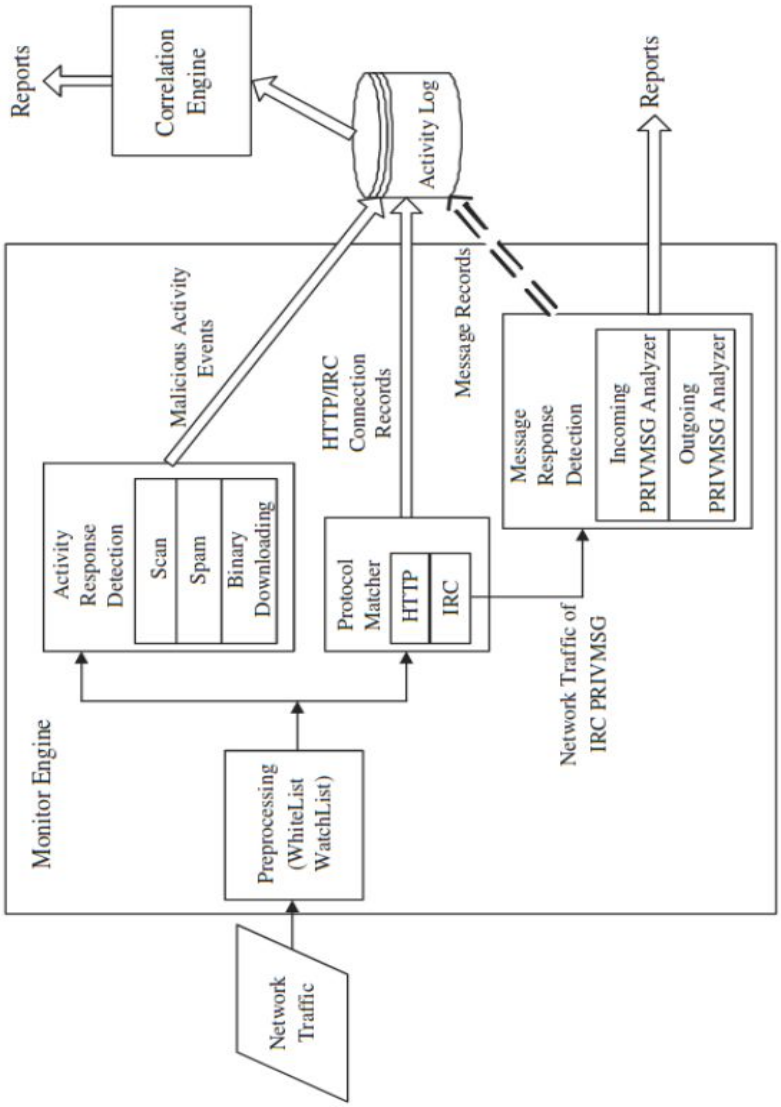

Figure 1 (Botnet Sniffer Architecture)

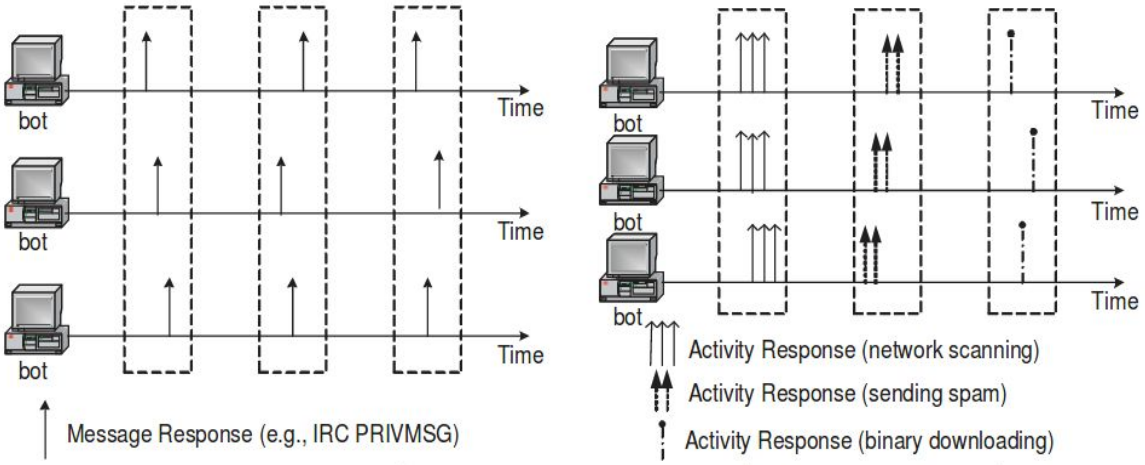

Figure 2 (Message and activity response)

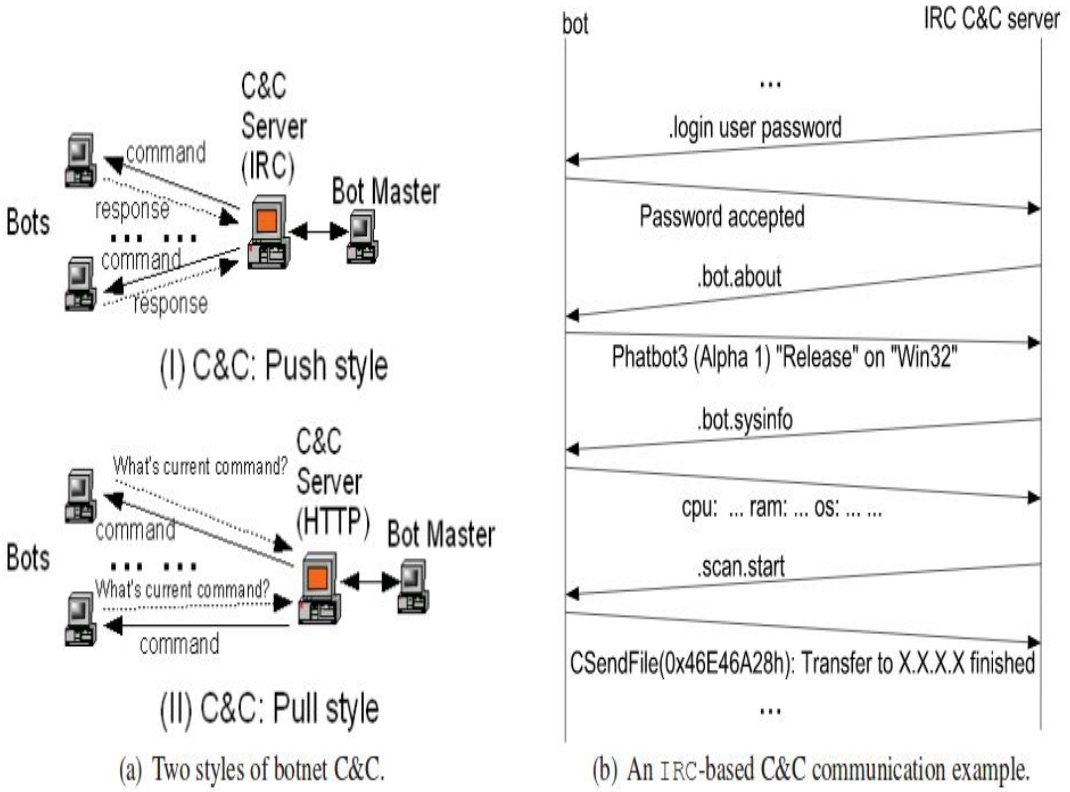

Figure 3 (Botnet Command and control)

2.1.2 BotMiner

The BotMiner detection framework [2] has three different phases of analysis. The first phase groups together host with similar activity patterns. The second phase groups together hosts that achieve similar attacking patterns, and the third phase groups together hosts on the basis of similarities from the other two phases. The first phase uses flow information such as the Internet Protocol (IP) addresses, ports and the network profile of the flow. It computes statistical measures such as flows per hour, packets per flow, average bytes per packets and average bytes per second. A two-step Xmeans clustering method is used to create clusters of hosts that behaved similarly in the network. The second phase uses the Snort IDS to extract attacks from each host and to group hosts by attack type. Among the malicious actions detected are SPAM, exploit attempts, port scans and binary download. The third phase computes a score for each attacking host on the basis of previous similarities and uses it to calculate a similarity function between hosts. Finally, a dendrogram is created using this function to find out the best cluster separation.The most important statistical characteristics used were the mail exchanger DNS queries to detect SPAM and the 25 TCP ports for SMTP. The proposal uses both virtualized and real botnet captures. Unfortunately, there is no information about how the botnet dataset was verified. The normal captures were verified using the BotHunter and BotSniffer software. In the preprocessing stage, some well-known Websites are whitelisted. However, the implications of the filter are not exposed. It also forces the method to maintain a list of well-known hosts, which could be error prone and time-consuming. The results seem encouraging. However, they could not be reproduced because the dataset was not made public. Comparisons with other papers were not carried out. Unlike other proposals, this work uses one novel idea to differentiate between botnets and manual attacks: botnets act maliciously and always communicate in a similar way, but the manual attacks only act maliciously.

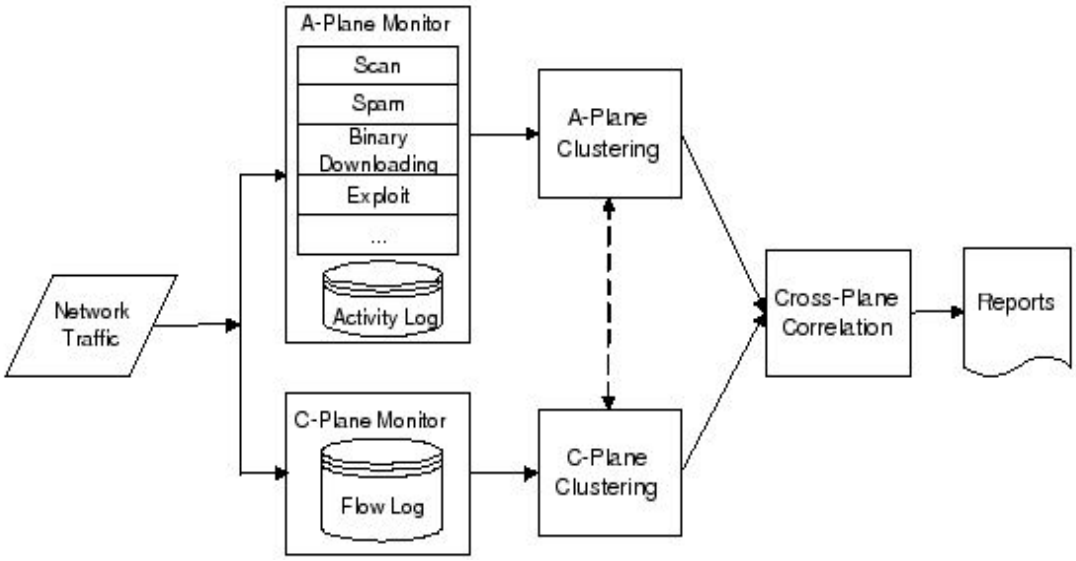

Figure 4 (BotMiner Architecture)

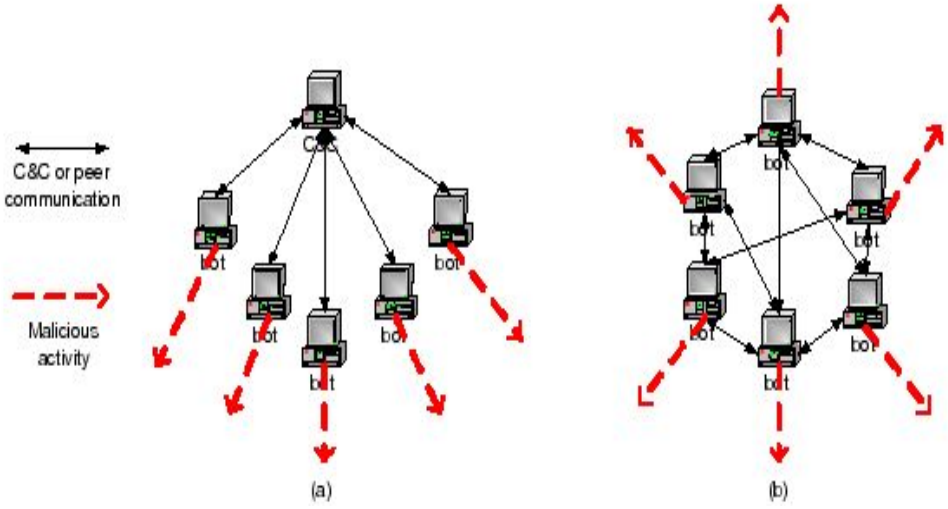

Figure 5 (Possible Botnet structures)

2.1.3 BotHunter

The BotHunter framework [3] recognizes the infection and coordination dialogue of botnets by matching a state-based infection sequence model. It captures packets in the network egress position using a modified version of the Snort IDS with two proprietary detection plug-ins. The proposal looks for evidence of botnet life cycle phases to drive a bot dialogue correlation analysis. IDS warnings are tracked over a temporal window and contribute to the infection score of each host. The host is labeled as a bot when certain thresholds are overcome. This proposal associates inbound scans and intrusion alarms with outbound communication patterns. An infection is reported when one of two conditions is satisfied: first, when an evidence of local host infection is found and evidence of outward bot coordination or attack propagation is found and, second, when at least two distinct signs of outward bot coordination or attack propagation are found. The BotHunter proposal is based on a statistical IDS, thus, some of the detections are static. For example, the IRC protocol is identified by means of the 6667 port, and some IP addresses of known botnet servers are embedded into the Snort configuration file. Among the bot statistical characteristics detected, the sequence of bytes in the binary download and the Snort statistical fingerprints are used by the proposal. The proposal did not publish the dataset used. This might be due to privacy issues or nondisclosure agreements. There are also no details about the Statistical Scan Anomaly Detection Engine and Statistical Payload Anomaly Detection Engine port scanning detection algorithms. The method has two major advances.

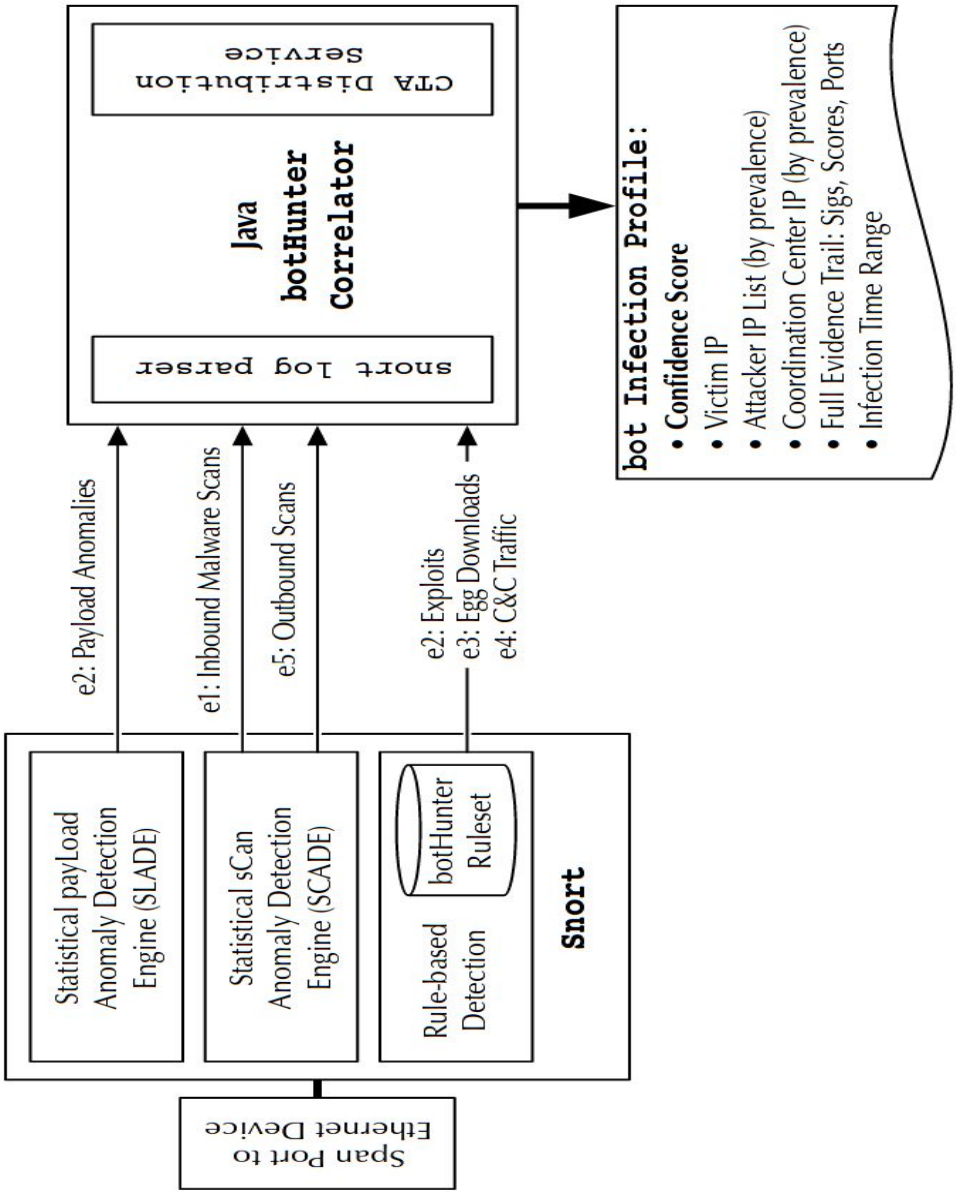

FIgure 6 (BotHunter Architecture)

2.1.4 N-Gram

The N-gram work [4] proposes an unsupervised, classification-based and IRC-based bot traffic detection method. It has two phases: classification of application traffic and detection of bots. The first phase has several stages. The first stage classifies traffic into known applications using signature-based payloads and known ports (signatures were generated previously). The second stage classifies unknown traffic from the previous step using a decision tree based on temporal-frequent characteristics. This tree differentiates known application communities (the proposal have the temporal-frequent characteristics of known flows). The third stage uses an anomaly-based approach to cluster the traffic. To differentiate malicious traffic from normal traffic, the proposal first creates profiles for known applications; these profiles use a one-gram technique that builds a 256-position vector holding the number of times that every byte appeared in the traffic payload. The real first stage, then, clusters the unknown traffic (which still remains unknown after the first phase) using the K-means, unmerged. The proposal assumed that bots reply to commands quickly, that bots communicate synchronously and that botnet IRC content is less diverse than normal IRC. The analysis shows that the training dataset might not be large enough to be considered representative of the problem. It has only one real botnet, one simulated botnet and two normal captures. The paper does not describe any verification of the normal captures. The analysis shows that the method is not completely unsupervised, because the centroids of the k means algorithm were manually selected. The method presents a novel and simple idea to detect botnet clusters automatically: to search for the cluster with the lowest standard deviation.

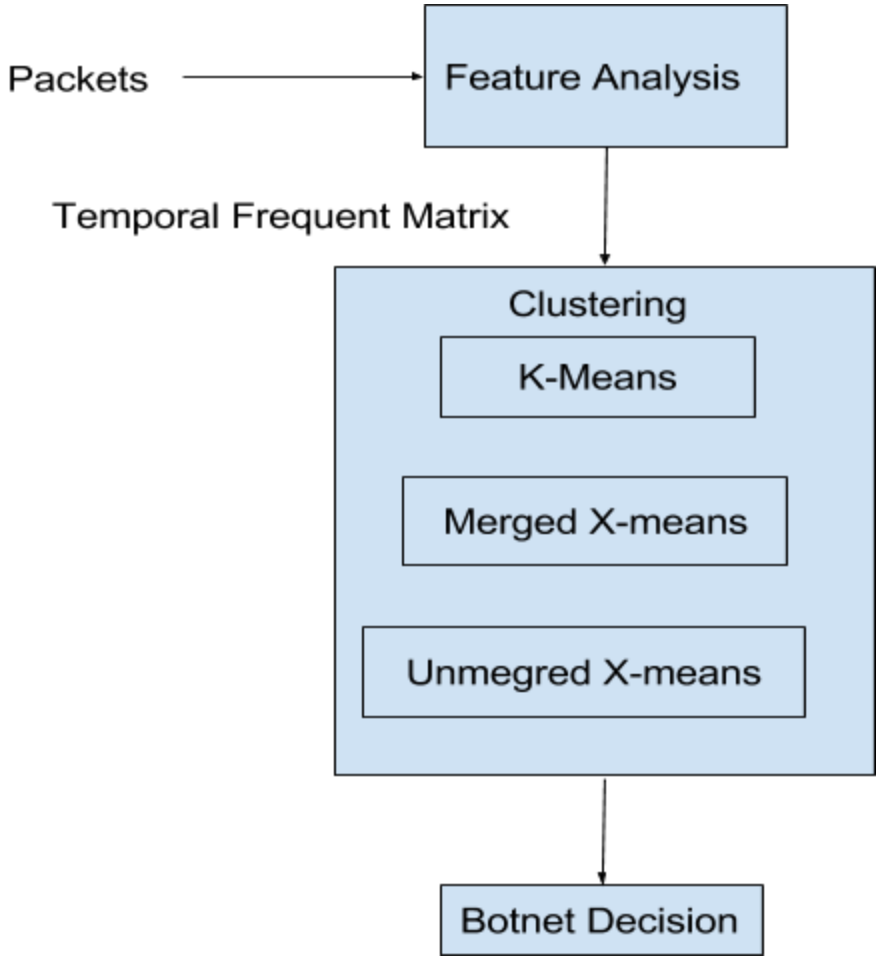

Figure 7 (N-gram architecture)

Unclean

The Unclean proposal [5] predicts future hostile activity from past network activity. The term spatial uncleanliness is defined as the propensity of bots to be clustered in unclean networks, and the term temporal uncleanliness the propensity of networks to remain unclean for extended periods. The proposal is divided into two phases.In the first phase, a dataset is used to evaluate both unclean properties. Although there is no detailed information about the capture procedure, the dataset has two types of information: the first type is external reports of phishing, port scans and SPAM activity, and the second type is some internal network captures. External reports from third parties are used as

ground truth for the experiments. The internal capture was performed by observing a public network. The spatial uncleanliness property is evaluated by comparing the population of IP addresses in an unclean report against the normal expected random popu-lation of the Internet. If the hypothesis is true, then the IP addresses of the bots should be more densely packed than the randomly selected addresses. The temporal uncleanliness property is evaluated by testing how an old report of unclean addresses predicts compromised IP addresses in the future. It is expected that unclean old reports are better predictors of future IP addresses than randomly chosen IPaddresses.In the second phase, the paper analyzes the performance impact of blocking these future bot IP addresses to differentiate between innocent, unknown and hostile IP addresses. An FP is considered each time an innocent IP address is blocked, and a TP is defined each time an unclean IP address is blocked. Unknown IP addresses are not used, probably adding a bias to the results. Results supported the hypothesis of spatial and temporal uncleanliness in the experiments.An analysis of the validation experiments for the first phase suggests that the results are incomplete. A TPR of90% was reported, but no FP, TN, FN, FPR or FNR were reported. Also, results are not compared with those of any other paper. Two assumptions were made: first, that bots are always used to attack and, second, that if a 5-month-old unclean report can predict current unclean activity, then a recent report should be more effective. The analysis of this proposal shows that the verification of the external reports was not described and that the dataset was not published. Also, it was not in the design of the proposal to differentiate between botnets and other types of attacks.

Tight

The Tight proposal [6] aggregates traffic to identify hosts that are likely part of a botnet. The hypothesis is that some evidence of botnet IRC C&C communication activity can be found by monitoring IRC traffic at a core location.The training dataset is composed of 600 GB of wireless normal traffic from the CRAWDAD archive and 74 flows of a compiled botnet from an internal test bed. The proposed method has five steps. First, flows are mixed in a common dataset (unfortunately, there is no information about how the mixture was performed). Second, a whitelist and a blacklist are used to filter the flows. Third, flows are classified according to their chat-like characteristics.Fourth, correlations are used to find similar C&C flows.

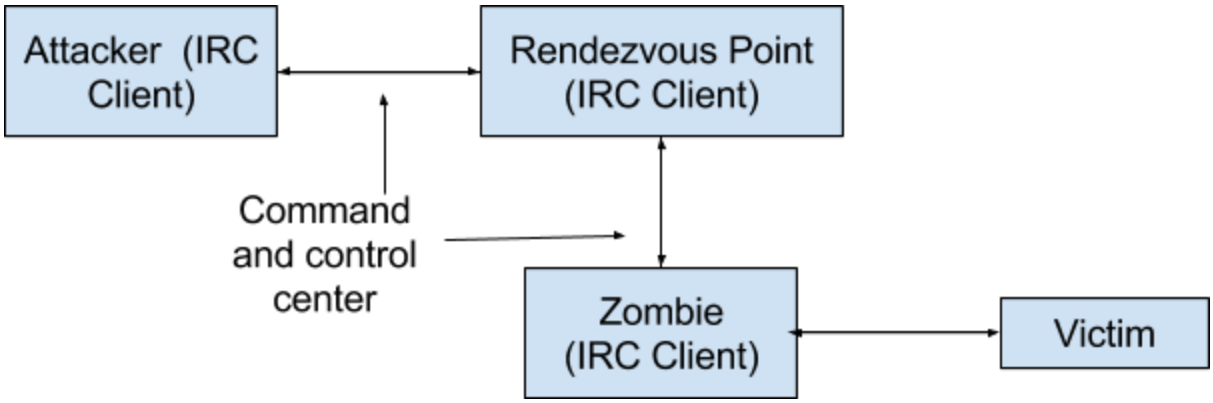

Figure 8 (Actors in IRC Botnets)

Fifth, a topological analysis of the flows is carried out to identify common connection endpoints and to manually determine which is the traffic between the controller and the rendezvous point.Several assumptions are made in the paper: first, that the IRC-based botnet command messages sent by the botmaster are brief and text based; second, that two different flows of the same application behave similarly; and, third, that botnet C&C channels do not sustain bulk data transfers.The analysis shows that some filters overfits

the data, such as deleting the high-bit-rate flows. However, these filters were not verified, and they seem to be created specifically to filter out the already-known botnet traffic.There is an issue regarding the dataset. When the botnet packets were obtained, the secure shell traffic generated from the test bed setup was also captured. This biases the capture. Almost every dataset has a bias, but it is important to enumerate and consider them. Furthermore, the normal captures were not verified, and the dataset was not published.Another bias is present in the results. Some results were obtained by applying the classifiers to one dataset; other results were obtained by applying the classifiers to another different dataset. It is commonly accepted that there should be at least three types of datasets: training, testing and validation. An evaluation methodology should also be used.The design of the proposal is considered to detect in real time, but unfortunately, no experiments were carried out to support this idea. A major highlight of the paper is that it focuses on novel behavioral features: a bot communicates with a C&C server, the server is controlled by a botmaster and the bots synchronize their actions.

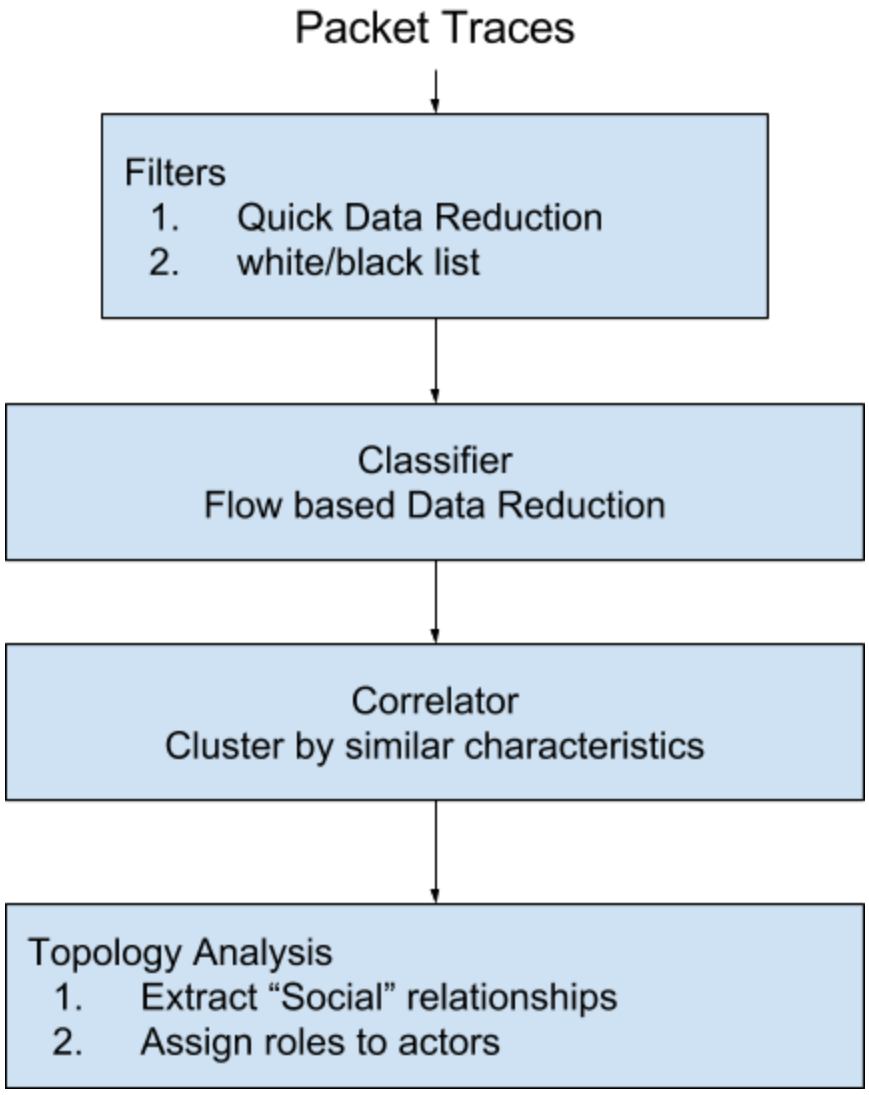

Figure 9 (Botnet detection processing pipeline)

2.1.5 Stability

In Stability [7], P2P botnets are detected using flow aggregation and stability measurements.

The hypothesis is that the proposal can further distinguish the structured P2P-based bot from normal clients by detecting the stability of I-flow.” The dataset was created by capturing background traffic from a university campus and later injecting real botnet traffic into it. The botnet traffic was captured in a honeynet, and thus, it is verified as malicious by the definition of a honeypot. This proposal is divided into two phases. In the first phase, the flow features are extracted. In the second phase, these features are used to compute the stability of the flows. Before the first phase, the dataset is divided into 1-h time windows. The first phase is composed of several steps. In the first step, protocol flows are aggregated on the basis of the double two-tuple IP addresses and ports. In the second step, for each aggregated flow, two novel values are computed: flows per aggregation flow (fpf) and average bytes per aggregation flow (abf). In the third step, an entropy value that uses the fpf as a random variable is computed. A high fpf and an abf lower than 300 bytes were used to detect bot flows. The aim of the second phase is to compute the flow stability with two sliding windows: one serves as a base- line and the other as a detection window. In the first step, the distance between the abf of both windows is compared against a predefined threshold. Every time the distance overcomes the threshold, a flow change is counted. In the second step, windows are moved forward, and the stability index of the flow is computed. If the stability index is above a threshold of 90%, then a botnet is detected. The proposal assumes that common DNS ports are not going to be used for C&C channels and that the common P2P ports are enough to detect that protocol. The analysis of the proposal shows that the preprocessing stage adds a bias to the results. The first bias is added when DNS traffic is filtered out after discovering that it has a stability index similar to that of P2P botnets. The second bias is added when port 28000 is filtered after discovering that it is being misdirected by the algorithm. Reproduction of the method is not possible because of two reasons. First, the paper uses several unspecified ad-hoc experimental values. Second, the dataset was not made public. Unfortunately, the normal traffic was not validated.

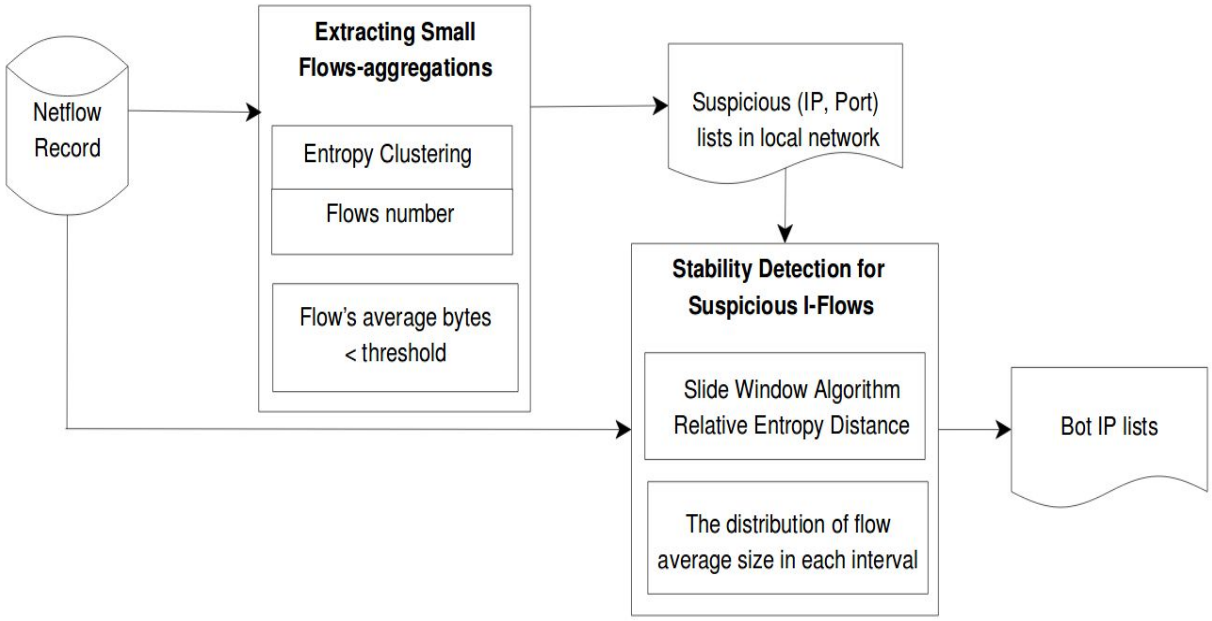

Figure 10 (Stability System Architecture)

Incremental

In Incremental [8], the botnet activity is monitored on the basis of similarities between the traffic feature streams of several hosts.The proposal is divided into five steps. The first step captures both normal and botnet traffic from an internal network. P2P botnet traffic is obtained from a third party. The captures in this step are simulated, as the internal network did not have access to the Internet.The second step reduces the volume of normal data by applying some filters. Non-TCPs are filtered out, several whitelists and blacklists of common Web sites are applied and packets with more than 300 bytes are deleted. The third step creates the feature streams. Traffic is divided into 5-min time windows, and two features are computed:average bytes per packets and packet amount.The fourth step computes a distance measure between streams using an incremental discrete Fourier transformation technique. The novel characteristic of this technique is that it does not need to recalculate all the coefficients everytime a new value arrives.In the fifth step, the Snort IDS is used to manually verify the activity of the

suspect hosts (hosts with high similarity streams). In this step, it is also decided whether this host is part of a botnet or not. Unfortunately, no information about this step is given.The proposal assumes that simulated botnet traffic behaves like real botnet traffic and also that the traffic from their network is normal. The proposal was not designed to differentiate between botnet and other attacks.The analysis shows that the results are unclear. No FNs were computed, and a perfect detection accuracy is reported. The paper also states that the “false positive rate is still relative low because we will analyze their activities to confirm the final result.” Unfortunately, the confirmation is not in the proposal. The results are not compared with those of other papers.The proposal uses several filters in the preprocessing step. However, it does not seem that the bias added by these filters would have been evaluated.Different distance measures are used in the proposal.However, there is no information about the distances called protocol distance measure and start time distance measures. Consequently, it is difficult to reproduce the method.During the validation experiments, normal traffic and botnet traffic were mixed. Unfortunately, the mixture process is not explained.This proposal has two highlights. First, it seems capable of monitoring botnets in real time using the time series-like feature streams. Second, the feature stream implementation, although not novel, is an important improvement in the area

Models

The Models [9] proposal identifies single infected machines using previously generated detection models.These models are composed of two parts. The first part extracts the strings received by the bot on the network to find the commands. The second part tries to find the responses (attacks) to the command sent. Models are stored for later use in real networks. The novel idea is to search for command signatures (string tokens) within the network traffic and then search for the responses to these commands. This technique correctly separates botnets from other attacks.The proposal is separated into five phases.The first phase consists of several steps. In the first step,traffic from 50 bots is captured. In the second step, bot responses are located within the traffic. This is

carried out by separating the traffic into 50-s time windows and by computing eight features for each time window. The features are the number of packets, cumulative size of packets in bytes, number of unique IP addresses contacted, number of unique ports contacted, number of non-ASCII bytes in the payload, number of UDP packets, number of HTTP packets and number of SMTP packets. In the third step, the features are normalized. In the fourth step, the change point detection algorithm uses these features to detect where the traffic changes. This algorithm uses the cumulative sum method to detect the changes.Once the change point is detected, in the fourth step,data are extracted from the traffic to create the profiles.Bot command profiles are composed of string snippets extracted from the 90-s time windows where the changepoint detection took place: the number of UDP packets,number of HTTP packets and number of SMTP packets and IP addresses. The final response profile is the average of these vectors over the total time of the bot response.In the second phase, models for commands and responses are created. A model has two parts: the strings for command detection and a network-level description for response detection.In the first step of the command model creation, snippets that contain the same tokens are grouped together using the longest common subsequence algorithm. In the second step, response traffic payloads are clustered together using hierarchical clustering. In the third step,snippets from the first-step clusters are grouped according to the second-step traffic clusters (snippets likely to generate the same response). In the fourth step, the precision of the method is improved by deleting the tokens found in previously generated and unverified normal captures.Still, in the second phase, in the first step of the response model creation, the element-wise average of individual profiles is computed on each cluster. In the second step,some FPs are filtered out using a threshold. If none of the following thresholds are overcome in 50 s, then the response is discarded: 1000 UDP packets, 100 HTTP packets, 10 SMTP packets and 20 different IP addresses.Once the models are created, the proposal implements them. In the third phase, previous models are mapped into the Bro IDS. Bro is configured to work in two stages. In the first stage, if a Bro command signature is matched,then Bro changes to stage 2. In stage 2, a bot detection is

achieved if any of the following traffic thresholds are overloaded: the number of UDP packets, number of HTTP packets, number of SMTP packets or number of unique IPaddresses.In the fourth phase, an evaluation is performed to obtain the performance metrics. A total of 416 binary bots (18 bot families) were obtained from the Anubis service along with 30 non described storm captures. The evaluation process is performed in several steps. In the first step, detection models are created using a training set composed of 25% of the total traces. In the second step, results are obtained using a testing set composed of 75% of the total traces.In the fifth phase, two real-world experiments are carried out. First, the already configured Bro IDS is used in a university campus network, and later, the same Bro IDS is used in an organizational network.The HTTP is detected by filtering the TCP port 80, and the SMTP is detected by filtering the TCP port 25.The analysis shows that the bot families are separated by hand while creating the models, probably because, once created, the models are useful for a long time. On the other hand, the FPs are manually verified. These issues make the verification of the method difficult.The proposal seems to use a good dataset, but the information included is not enough to verify its diversity. Also,the third-party Anubis dataset is not verified. Moreover,there is no information about the preprocessing of the dataset.A highlight of this proposal is that the results are compared with those of the BotHunter proposal using the Models proposal [31] dataset.

2.1.6 FluXOR

The FluXOR proposal [10] detects and monitors fast-flux-enabled domains. Given a fully qualified domain name, its goal is to verify whether it concealed a fast-flux service network and, in such a case, to identify all the agents that are part of the network. Although not strictly a botnet detection technique, it is one of the few proposals that could detect fast-flux botnet domains. Suspicious domains are detected when traces of fast-lux characteristics are found in the DNS and WHOIS information of the domain. A fast-flux behavior is detected with nine features that describe the properties of the domain, such as the degree of availability and the heterogeneity of the hosts in the DNS

A records. The experiment setup includes a collector module, which harvests domains from different sources; a monitor module, which monitors domains; and a detector module, which feeds the classifier algorithm from the Waikato Environment for Knowledge Analysis suite . This proposal does not have a detailed performance metrics analysis, so it is not possible to evaluate how well it performed. A 0% FP was reported; however, no other results were given. The analysis shows that the malware domains are extracted from SPAM mails and normal domains are extracted from Web history. However, there is no description about the verification of the domains. The design of the proposal does not include real-time detection capabilities. However, under some conditions, it could be possible to detect fast-flux botnets within a very short time.

2.1.7 Tamed

In the Tamed proposal [11], it is hypothesized that “even stealthy, previously unseen malware is likely to exhibit communication that is detectable.” The proposal aims to“identify infected internal hosts by finding communication aggregates, which consist of flows that share common net-work characteristics.” This is the first proposal that does not try to detect botnets themselves but presents a coherent and useful set of features in which future work can rely their detection methods on. The work is divided into two phases: the definition of their aggregate feature function and the evaluation of experiments.In the first phase, three different aggregation functions are defined: destination aggregation function, payload aggregation function and common platform function.The destination aggregate function finds suspicious destination subnetworks for which there are a large number of interactions with the internal network (and the IP addresses involved). This is performed by comparing a baseline past network record with the current network record. The function is composed of two steps. First, the past normal traffic is used to remove the periodic communications. Second,the principal component analysis algorithm is used to find the most important components, and then, a clustering technique is used to find the hosts that connect to the same combinations

of destinations.The payload aggregate function uses the edit distance with substring moves to output a normalized ratio. This ratio indicates how many distances are below a given threshold between two payloads (and thus need training).The proposal also contributes an algorithm for approximating the fraction of relevant record pairs that satisfy this distance (using a variant of the k-nearest neighbor).The common platform function uses two heuristics for fingerprinting host operating systems passively: time-to-live fields and communication characteristics (such as Windows computers connecting to the Microsoft Time Server).This last function returns the largest fraction of internal hosts that can be identified with the same operating system.The proposal assumed that the C&C server can be in a different network and that bots will not use the Tor*network.The analysis shows that the normality of the traffic captured at the university is not verified but granted asnormal.In one of the experiments, the traffic from one bot-net was merged with traffic from the university. The IP addresses of the bots were substituted with the IP addresses of hosts in the university. The intention was to have IP addresses with normal behavior patterns and botnet behavior patterns at the same time. However, no analysis was performed about how this change could affect the common platform aggregate function. This issue might be the reason of some FNs during one experiment. Unfortunately, the dataset was not made public, so it is difficult to reproduce the method.The proposal stated that a 100% TPR was achieved with the exception of one experiment. In this isolated excep-tional experiment, an 87.5% TPR was reported. The results of all the experiments should be computed together to calculate the performance metrics. There cannot be any experiment exceptions while computing the results.The proposal was not designed to detect bots using only the traffic from one host. Also, it was not designed to differentiate between botnets and other types of attacks.

Markov

Port scan activities of botnets are detected in [12]. The scanning phases are represented with text symbols, and a hidden Markov model (HMM) is used for training and detection. The work is divided into four phases. In the first phase, a system is used to capture information from the network and generate network logs. Unfortunately, this system is not described. Data were captured for 1 month in11 C-class university networks. Incidentally, an unknown amount of hosts was found to be infected.In the second phase, the amount of TCP packets present in certain time windows with SYN and ACK bits is computed. Unfortunately, no information about the time windows is given. The amount of packets is plotted, and each distinct shape of traffic is assigned a different text symbol.This phase supposedly ends up with a string of letters (one gram) that represent each port scan (observations).In the third phase, the training phase, the Baum–Welch algorithm is used for 6 hours to find a Markov model that maximizes the probability of each sequence of observations.In the fourth phase, strings are extracted from the rest of the traffic (supposedly of unknown type). However, there is not much information about this. Then, the Viterbi algorithm is used to find the most probable sequence of states in an HMM from each group of observed states, that is, to find the model that better explains the observations. The result of the Viterbi algorithms is used as a type of score. We suppose that they selected the sequence of stored port scan strings that maximizes the score. At some point, a comparison is made between a botnet pattern and an unknown pattern using a similarity function. Finally, the early detection of port scans is defined by these similarity values.However, no information about any threshold or decision boundary is given.The proposal assumes that the botnets only scan TCP ports.The analysis shows that the dataset used is not verified.There is no clear description of the dataset. Moreover, it is stated that the

dataset contains both unknown data and botnet data, but it is not labeled. Also, results were not compared with those of other papers.Unfortunately, it is difficult to completely understand the method. There is no information about how the portscan detection is performed. Also, the proposal states that the method behaves fine with Monte Carlo simulations, but there is no information about this. The experimental setup is not described. Furthermore, a bigger detection system is described, with botnet policy management and classifications steps, but there is no information or explanation about it. The information given about the automatic detection of botnets does not have enough details.

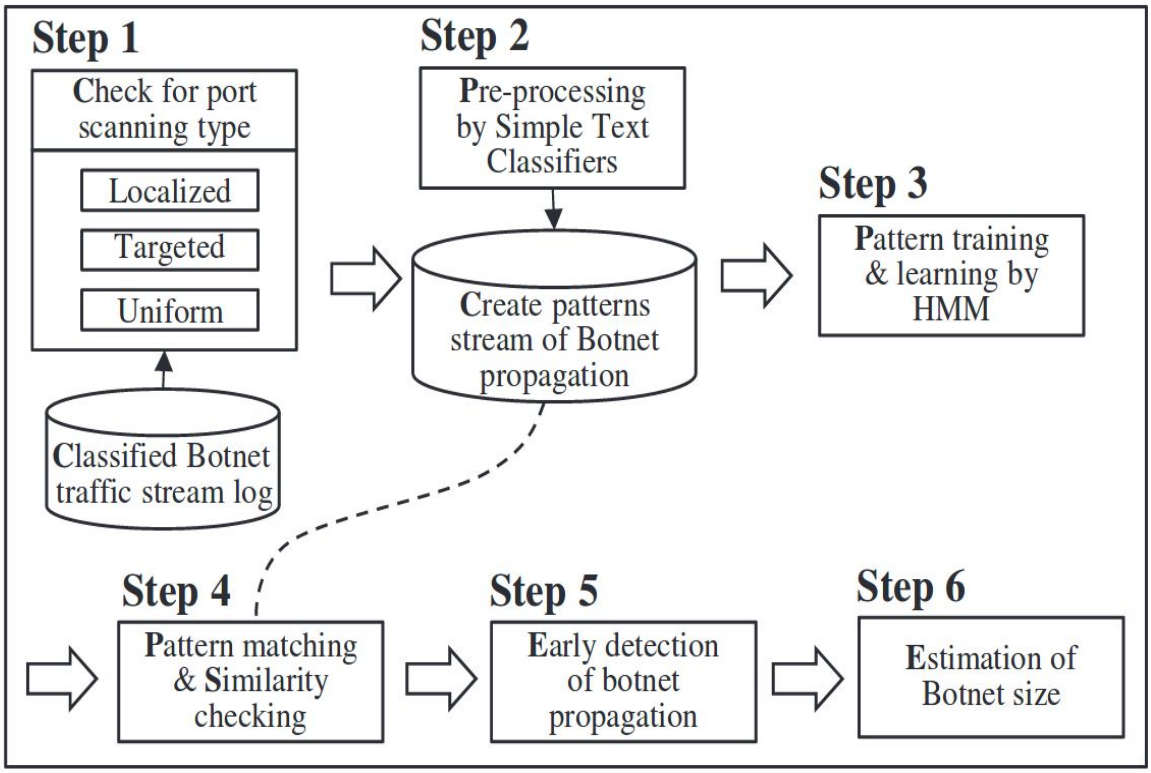

Figure 11

(Overview of botnet-propagation early-detection. HMM, hidden Markov model)

2.1.8 Synchronize

Network synchronization and clustering techniques are used to detect bots in [13]. There are five phases. In the first phase, data are captured. Three different test beds were used to capture five datasets. Three of the datasets correspond to botnet captures and two to non botnet captures.The non botnet captures include a manual port scan and normal traffic. Verification of the captures is carried out for both botnet and non botnet datasets. The botnet binaries are verified with the VirusTotal Web service and the EUREKA! automated Malware Binary Analysis Service. The botnet traffic is manually verified by experts. The non botnet traffic is verified by a fresh installation of the operating system, antivirus programs and the Snort IDS.In the second phase, the TCP flows are extracted using the tcptrace tool.In the third phase, flows are divided into 1-s time windows, and three features are used to aggregate them: the amount of unique source IP addresses in a time window,the amount of unique destination IP addresses in a time window and the amount of unique destination ports in a time window. Each instance represents a time window with the aggregated features.In the fourth phase, the expectation–maximization algorithm is used to cluster the instances. A dataset with both types of traffic was obtained by merging botnet and non-botnet data. Four experiments were conducted: first, to separate between botnets and port scanning activities; second, to separate between botnet and non botnet traffic from one host; third, to separate between botnet and non botnet traffic in an unbalanced dataset; and fourth, to separate between a bot and a network of non botnet hosts.The fifth phase analyzes the results. The proposal considers the usual definition of FPs and FNs. However, it also includes a novel definition of error metrics based on the administrator perspective, that is, to compute an FP only when a non botnet IP address is detected as a botnet at least once during a period and also to compute an FN when a botnet IP address is always detected as a non botnet in a time window.The proposal assumes that botnets tend to

generate new flows almost constantly and that botnets' most typical characteristics are maliciousness, being remotely managed and synchronization and also that botnets only use the TCP protocol.The analysis shows that the dataset included non botnet data and manual attacks. However, it is rather small, and it does not cover a large proportion of botnet behaviors. Only five captures were made, and none of them was made in a large network.Unfortunately, there is no justification of the 1-min time window used. The results were not compared with those of other papers.One of the highlights is that all the tools and datasets used were made public on the Internet. These datasets might allow other researchers to verify the method.Another highlight is the differentiation between botnet and port scanning traffic.Unfortunately, no automatic botnet detection method was established, meaning that the final decision is left to experts.

3. Research Approach:

3.1 Datasets:

We used two Datasets from two different sources for this project.

1. UNB ISCX Intrusion Detection Dataset
2. CTU-13 by Malware capture facility, CTU university

3.1.1 UNB ISCX Intrusion Detection Dataset

UNB ISCX dataset is ideal for training any Botnet detection system. Some of the important properties are:

1. Realistic network and traffic: This dataset resembles real life traffic as it does not contains any artificial post traffic insertion. It contains both normal and malicious type of data while keeping the consistency of the dataset.
2. Labeled data: As the data generation was done inside the lab with complete control it was easy to label the dataset.
3. Network traffic is kept unsanitized so as to preserve the traffic in its natural form.
4. This dataset comprises of many types of attacks which are very close to the recent trends.

3.1.2 CTU Dataset

1. This is a botnet traffic dataset captured by CTU university, Czech republic in 2011. The goal of this dataset was to have a large capture of real botnet traffic along with normal traffic and background traffic. The CTU dataset consisted of total 13 captures of different botnet samples. On each scenario they executed a specific malware, which used several protocols and performed different actions. Each scenario was captured in a pcap file that contains all the packets of the three types of traffic. These pcap files were processed to obtain other type of information, such as NetFlows, WebLogs, etc. The first analysis of the CTU-13 dataset, that was described and published in the paper "An empirical comparison of botnet detection methods" (see Citation below) used unidirectional NetFlows to represent the traffic and to assign the labels. These unidirectional NetFlows should not be used because they were **outperformed** by our second analysis of the dataset, which used bidirectional NetFlows. The bidirectional NetFlows have several advantages over the directional ones. First, they solve the issue of differentiating between the client and the server, second they include more information and third they include **much more detailed labels**

Id	IRC	SPAM	CF	PS	DDoS	FF	P2P	US	HTTP	Note
1	√	√	√							
2	√	√	√							
3	√			√				√		
4	√				√			√		UDP and ICMP DDoS.
5		√		√					√	Scan web proxies.
6				√						Proprietary C&C. RDP.
7									√	Chinese hosts.
8				√						Proprietary C&C. Net-BIOS, STUN.
9	√	√	√	√						
10	√				√			√		UDP DDoS.
11	√				√			√		ICMP DDoS.
12							√			Synchronization.
13		√		√					√	Captcha. Web mail.

Figure 11 (Traffic Distribution in CTU Dataset)

3.2 Models

For differentiating between botnets from the normal one, we are using two different algorithm. First a set of simple machine learning algorithm, other being the Neural Network approach. We will talk about both separately.

3.2.1 Machine learning approach

This approach is a inspired by {research paper.} [14] and implemented. The proposed flow in the paper is as follows.

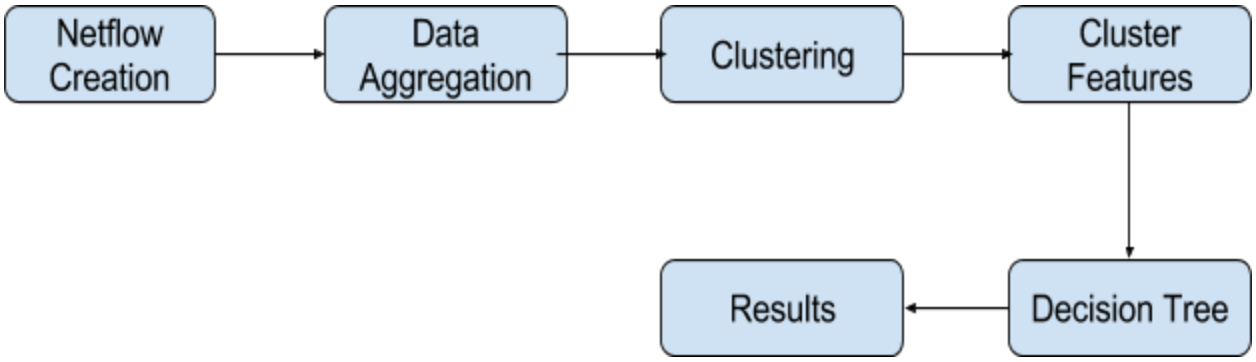

Figure 12 (workflow as suggested by research paper)

3.2.1.1 Problems in above proposed workflow:

The algorithm used here make use of decision tree on the features extracted from clusters for to predict whether a cluster is malicious or not. Problem comes when you try to try to label the flows, about their cluster and whether they fall in malicious cluster or not. That is we cannot cluster the incoming traffic and label them as the clustering is very unstable. To solve this problem we are proposing a different flow model as shown below.

3.2.1.2 Training flowgraph

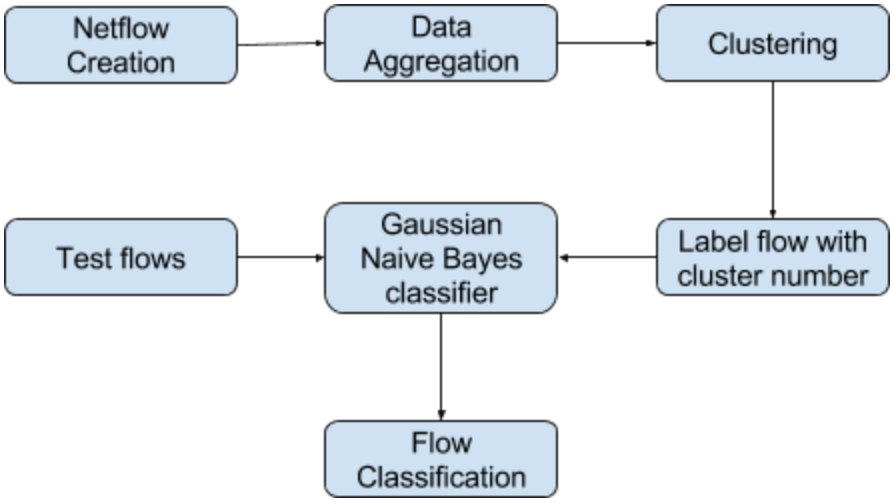

Figure 13 (Proposed Workflow)

3.2.1.3 Netflow Creation

We are using Argus Toolkit for conversion of pcap file to NetFlow due to its several advantages over the NetFlow standard. The differences between the NetFlow standard and the Argus flows are that the NetFlows are restricted to the version 4 of the IP protocol, and that they are unidirectional. For Argus, a flow is defined as all the packets that share the source IP address, the source port, the destination IP address, the destination port and the protocol. This format stores the flows in a binary file, it includes tools to process the flows, to apply labels, to filter flows, to plot flows, and it can be fed with live or offline pcap packets.

3.2.1.4 Data Aggregation:

In Data aggregation we divide the whole data in 1 minute time windows. This is being done due to the huge amount of data that need to be processed. A shorter time windows gives us a more power to process this data. Also 1 minute time window is optimum as it is not too short to let any flow pattern get broken and not too long as it becomes difficult to process. These small 1 minute time intervals are called as buckets. Inside each time window, the Net Flows are aggregated by unique source IP address.

The resulting features on each aggregation window are:

1. Source IP address
2. Number of unique source ports used by this source IP address
3. Number of unique destination IP addresses contacted by this source IP address.
4. Number of unique destination ports contacted by this source IP address.
5. Number of NetFlows used by this source IP address.
6. Number of bytes transferred by this source IP address.
7. Number of packets transferred by this source IP address

3.2.1.5 Clustering

The clustering of aggregated NetFlows is performed using Gaussian Mixture Model(GMM). A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. The GMM object implements the expectation-maximization algorithm for fitting mixture-of-Gaussian models. Expectation–maximization algorithm is an for finding or estimates of in, where the model depends on unobserved. The EM iteration alternates between performing an expectation step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization step, which computes parameters maximizing the expected log-likelihood found on the step. These parameter-estimates are then used to determine the distribution of the variables in the next step.

3.2.1.6 Label flow with cluster number

After the clustering of flow we assign a cluster number to each flow, that is where that particular flow falls among cluster. This label is also a indicator of whether a cluster and the flows falling within those cluster are malicious or not by looking at the number of malicious flow falling in that cluster.

3.2.1.7 Classifier

Once the labeling of flows are done. We train a Gaussian Naive Bayes classifier to classify the flows and the tag of a cluster being malicious or not. This trained model is used for further classifying test dataset netflows, and label them as Malicious or not. This way of classification can allow us to make this model in soft real time as we can buffer 1 minute time windows netflow from some network tool and enable us to predict whether to let this flow pass through or not. This way we can even block the malicious netflow from entering or leaving the network making a machine learning enabled firewall or Intrusion detection system resulting in a near real time system.

3.2.1.8 Gaussian Naive Bayes

Gaussian Naive Bayes methods is a of supervised learning algorithm based on applying Bayes' theorem of probability with the "naive" assumption that each feature is independent of other features in the classification process. To predict an output variable 'y' form a set of dependent input features x1,x2...xn. Bayes' theorem states the following relationship.

$$P\left(y \mid x_1, x_2, x_3 ... x_n\right) = \frac{P\left(y\right) P\left(x_1, x_2, x_3 ... x_n \mid y\right)}{P\left(x_1, x_2, x_3 ... x_n\right)}$$

Using the naive independence assumption that

$$P\left(x_i \mid y, x_1 ... x_{i-1}, x_{i+1}, ... x_n\right) = P\left(x_i \mid y\right)$$

for all i, this relationship is simplified to

$$P\left(y \mid x_1, x_2, x_3 ... x_n\right) = \frac{p\left(y\right) \prod_n^{i=1} P\left(x_i \mid y\right)}{P\left(x_1, x_2, x_3 ... x_n\right)}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$P\left(y \mid x_1, x_2, x_3 ... x_n\right) \propto p\left(y\right) \prod_{n}^{i=1} P\left(x_i \mid y\right)$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set. The different Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$. In spite of their apparently over-simplified assumptions, Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They requires a small amount of training data to estimate the necessary parameters. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\pi\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood in the implementation There are other Naive Bayes algorithms available too like multinomial naive bayes, and Bernoulli Naive Bayes. And can be used when the distribution of the

data seems to follow a particular pattern. In our case we tend to use the Gaussian naive bayes because for clustering we used Gaussian mixture models.

3.2.1.9 Gaussian Mixture models

Gaussian Mixture models are probabilistic model used to model the normally distributed subpopulation among the whole population. A Gaussian Mixture Model is a probability density function made up of weighted sum of individual Gaussian component densities functions. The Gaussian Mixture model make use of the expectation-maximization algorithm for fitting mixture models. Usually, Gaussian Mixture models cluster data by assigning data points to the multivariate normal functions that maximize the component's posterior probability given the data points. Gaussian Mixture models can also deal with clusters that have different sizes and correlation within them. Due to this reason Gaussian mixture models are more suited for this purpose, Other than this python libraries implements a very neat and elegant implementation of gaussian mixture models

3.2.1.10 Test Flows to Trained Models

After the model is trained with train data, we input our test flows to output in which cluster a particular flow will fall. By predicting in this way we don't need to cluster the incoming data. As clustering is very unstable on incoming data which was hindering our efforts to make this system real time. Now we can just use a tool like argus or tcpdump etc to get flows and input the flows into the model for prediction. After the flow classification has been done we can block the identified traffic by deploying an external system. This can also be included in future work.

Results

Confusion Matrix

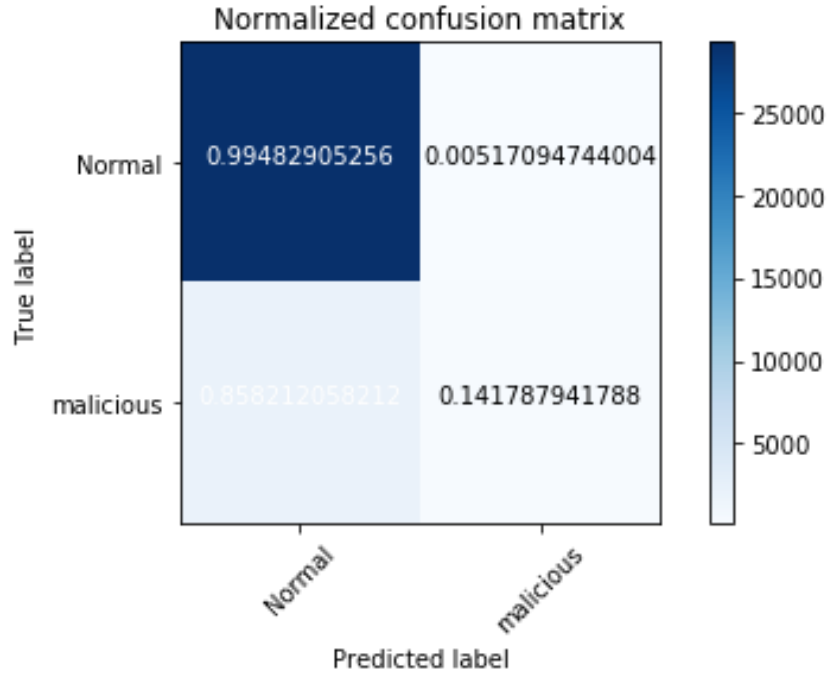

Figure 14 (Normalized Confusion Matrix, clustering method)

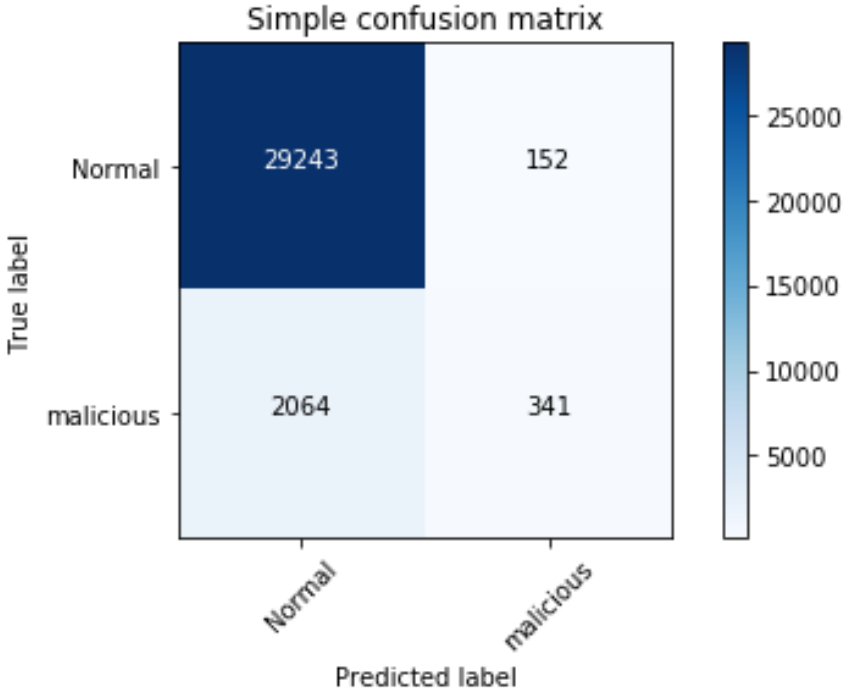

Figure 15 (Confusion Matrix, Clustering method)

Measures

Sensitivity	0.9341	TPR = TP / (TP + FN)
Specificity	0.6917	SPC = TN / (FP + TN)
Precision	0.9948	PPV = TP / (TP + FP)
Negative Predictive Value	0.1418	NPV = TN / (TN + FN)
False Positive Rate	0.3083	FPR = FP / (FP + TN)
False Discovery Rate	0.0052	FDR = FP / (FP + TP)
False Negative Rate	0.0659	FNR = FN / (FN + TP)
Accuracy	0.9303	ACC = (TP + TN) / (P + N)
F1 Score	0.9635	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.2924	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

3.2.2. Neural Network approach

3.2.2.1 Neural Network:

Artificial neural networks (ANNs) are a mathematical model used in research disciplines, which is based on a large collection of simple interconnected neural units (artificial neurons), loosely analogous to the observed working of a biological brain's neurons. Each neuron is connected with many others neuron, and this interconnection can increase or decrease the activation function\ output of connected neural units. Each individual neuron calculates using addition function. There are threshold function or limiting function on each connection between neuron and on the neuron itself, the signal must overcome the bound before propagating to next neurons. These neural structures are self-learning and self-trained, rather than programmed, and perform best where the solution or feature selection is quite difficult to express in a conventional computer program. Neural networks consist of multiple layers, and the signal travels from the input, to the output layer of neural network. Back propagation is the use of forward

simulation to modify the weights on the "front" neural network and this is done in addition with training where the correct final result is known. Modern networks are a bit more relaxed in terms of stimulation with connections interacting in a much more unorganised and complex fashion. Dynamic neural networks are the more advanced, in that they dynamically alter based on rules, form new inter connections and know new neural network while removing others. The aim of the neural network is to solve problems similar to that of human brain would, although several neural networks are more abstract. Modern neural network projects typically work with a few thousand to a few million neural units and millions of connections, which is still several orders of magnitude less complex than the human brain and closer to the computing power of a worm. The backward propagation of error or backpropagation, is a method of training artificial neural networks and often used with a optimisation method such as gradient descent, stochastic gradient descent and batch gradient descent etc. The algorithm consist of two steps propagation and followed by weight updation. When a input is fed into the network the value is propagated throughout the entire network layer by layer until the output layer. Then the output is compared with the required expected result, then an error term is calculated for each neuron in the network to adjust their activation functions. This process is carried out several times until the error converges to a small value. At this point the test netflows are fed into the trained model.

Features Selection:

In this method we used CTU's Dataset's folder number 1,2,3 which is available online. We selected only a handful of features as opposite to using many features. This is done as many features can not be one hot encoded like IP address. Neural network takes a vector of float as input, but the features such as source ip, destination ip and payload etc. The following features were used:

1. Duration of the flow

Duration of the flow can be found by converting the starting and ending time in Posix time style which is the number of the seconds that has been elapsed since 1 January 1970 till that time. Then we can just subtract the starting Posix time from ending posix time

2. One hot encoded Protocol vector of flow

   One hot encoding is a method which converts the categorical variables into a vector of ones and zeros. Suppose we need to one hot encode a categorical variable with 20 types of values then corresponding to each value in column a new vector is constructed with 20 components and having values as either one or zero depending whether the value whoc one hot vector is to be constructed has which value. So the component corresponding to that value (category) will have one and rest will have zero. This transformation of data is very useful when used in classification and regression analysis.

3. Source Port

   Every flow has a source port associated with them This port servers as unique identifier for the service running and sending and receiving data. This ensures the proper handling of incoming and outgoing traffic is done.

4. One hot encoded Direction Vector of flow

   This feature allow us to identify the direction of the flow being analysed. As the direction can have many values we are using one hot encoding to transform it into a vector which can be used in neural network.

5. Destination port

   Similar to source port destination port server the same purpose as source port but for the destination.

6. Total number of the packets transfer

   This gives us the total number of the packets transferred in the flow.

7. Total number of the bytes transfer

   Similar to the total number of packets transferred in the flow but it shows the total number of bytes although it has high number of correlation with total number of packets but this data filtering step has been skipped as we want the neural network to learn this and do it automatically.

Data Preprocessing

The data frame contains a very skewed ratio of number of normal and malicious example. So we grouped all the malicious netflow examples and similarly grouped all the normal one together. Then we took equal number of malicious and normal flows together and and combined them in a single data frame for further splitting it in training and testing data frames. This Data was then reshuffled so as to avoid any introduced systematic error due to this. After shuffling the whole data frame was divided into three parts for training, testing and validation of the model. These data frames were then converted to numpy matrix as the tensorflow graph only takes Tensors/matrix as input.

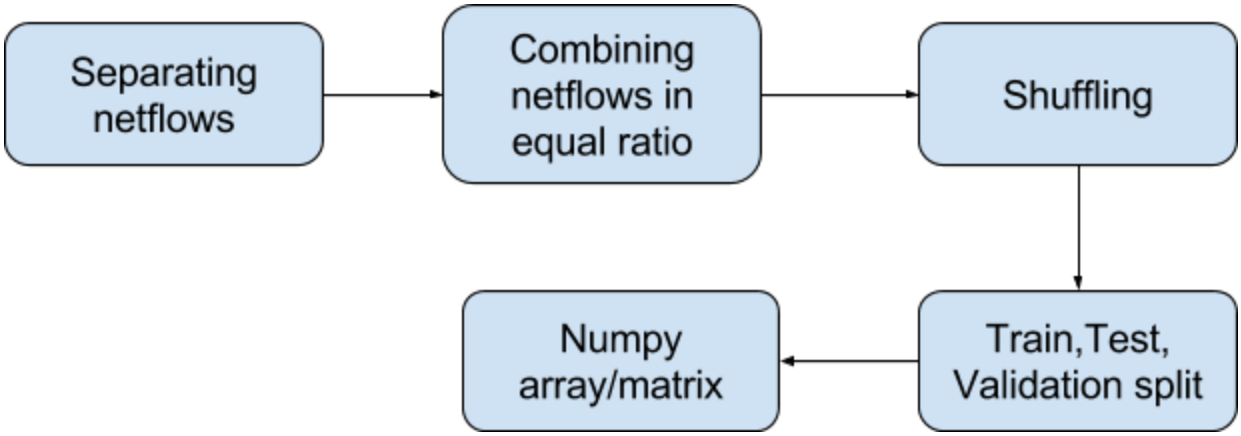

Figure 16 (Data Processing flowchart)

Neural network Structure

We used a simple back propagation neural network structure. The input layer was equal to number of input variable. The two hidden layer used has 16 neurons in each layer which are fully connected layer. Then we are using a dropout layer with 0.5 probability of dropping a particular value from a neuron. In the final output layer there are only two neuron output giving the probability of any netflow being malicious or not.

Dropout layer

Dropout layer is introduced in the network to incorporate a mechanism to fight overfitting. Dropout layer just drops certain values completely from propagating to next layer, in this way we are preventing from overfitting the data in the network. The dropout ratio determines the number of total value to be dropped, for example a value of 0.2 will drop one in five features randomly. A value of zero will let all the value propagate though and a value of one will block all the values. We have kept this value to be 0.5 so as to limiting only half of total values to propagate through the next layer.

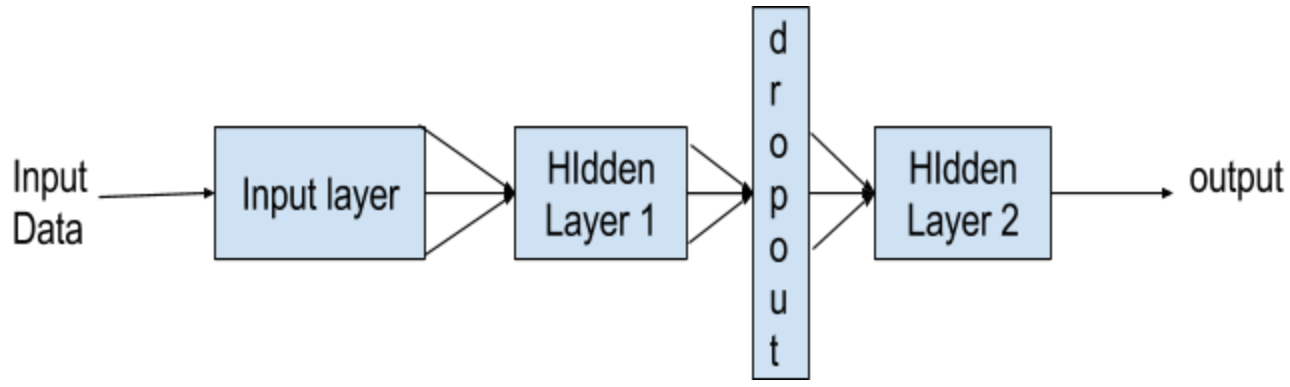

Figure 17 (Neural Network architecture)

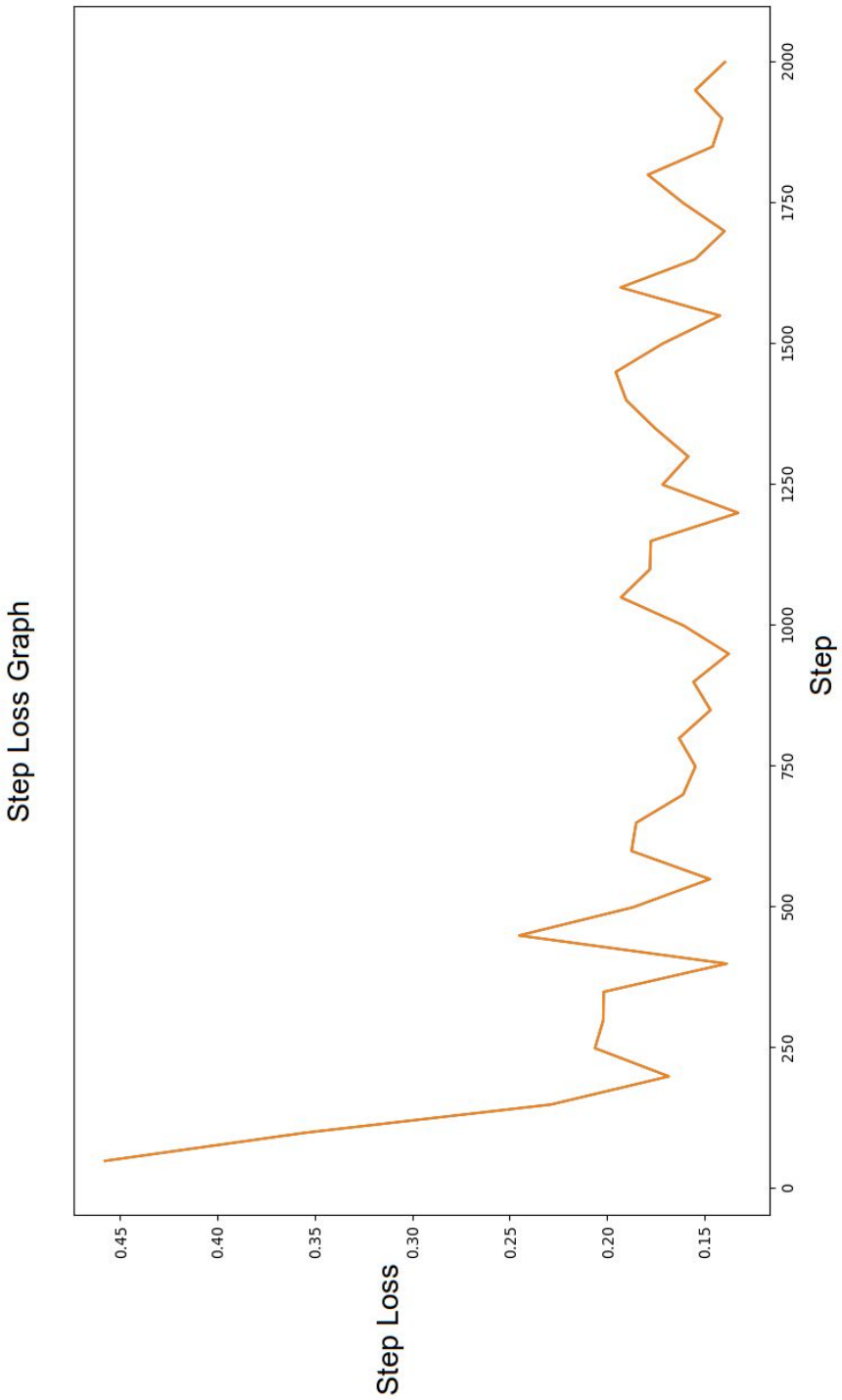

Figure 18 (Loss Graph vs Steps)

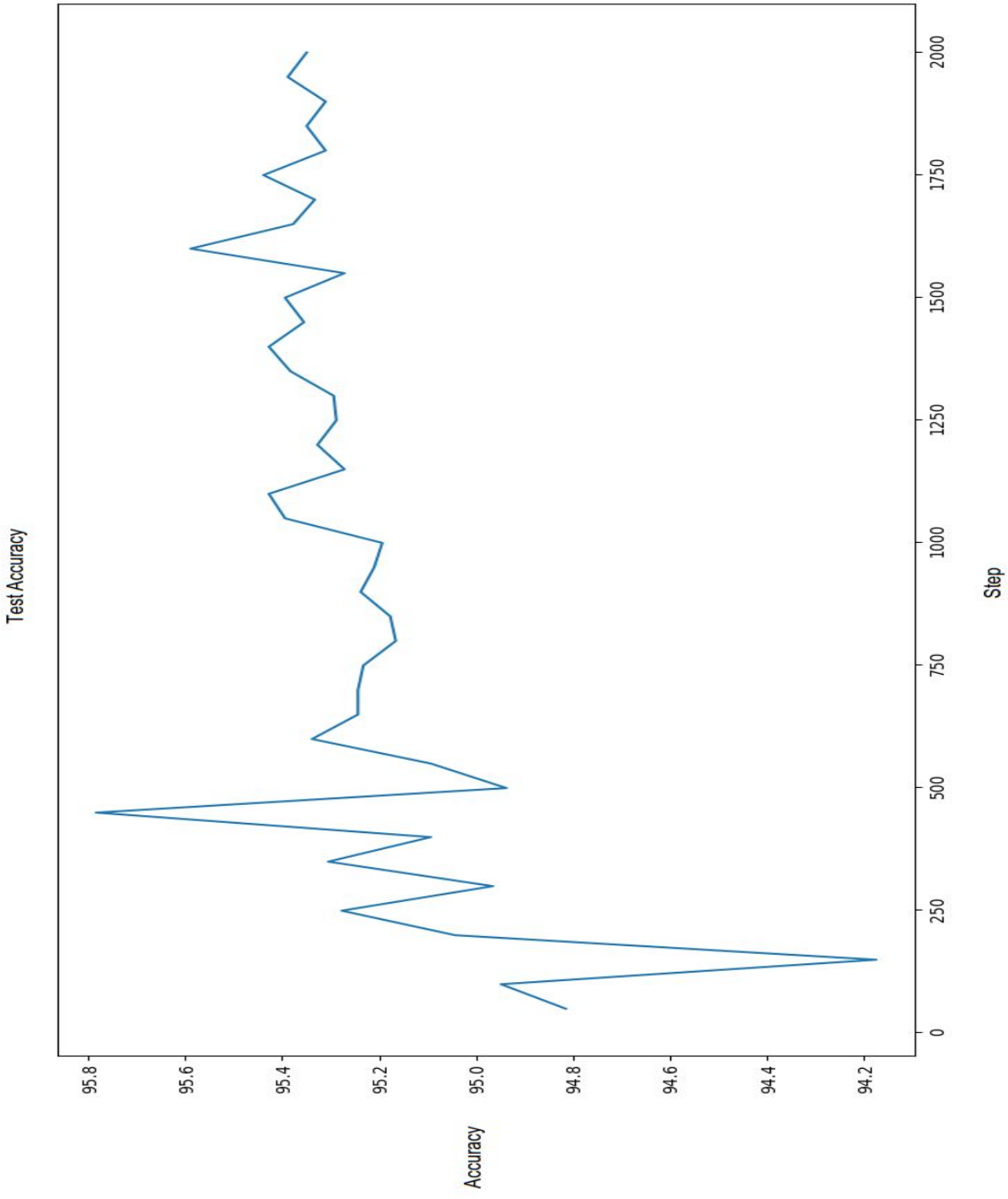

Figure 19 (Accuracy vs Step graph)

BackPropagation Tabular Result

step	Minibatch Accuracy	steploss	Test Accuracy	Validation Accuracy
50	91.11328125	0.457613647	94.81481481	62.13682354
100	92.48046875	0.3547400236	94.94848232	78.45539775
150	93.65234375	0.2287197709	94.1743247	94.71827717
200	95.21484375	0.1682162434	95.04316346	95.43302701
250	93.9453125	0.2061072141	95.27708159	96.01782233
300	94.04296875	0.2018313855	94.96519075	95.16383551
350	93.26171875	0.2016212642	95.30492899	95.56298153
400	95.60546875	0.1385124922	95.09328878	95.28450757
450	92.67578125	0.2450364828	95.78390421	96.09208206
500	95.01953125	0.1866013855	94.93734336	95.14527058
550	95.41015625	0.1471265554	95.09328878	95.21024784
600	93.65234375	0.1873557568	95.33834586	95.55369906
650	95.80078125	0.1849553883	95.24366472	95.34020236
700	94.82421875	0.1608996987	95.24366472	95.37733222
750	94.82421875	0.1545391232	95.23252576	95.34020236
800	95.01953125	0.1629344821	95.16569201	95.28450757
850	95.01953125	0.1468147635	95.17683097	95.3030725
900	95.01953125	0.1555558145	95.23809524	95.32163743
950	95.41015625	0.1374755502	95.21024784	95.32163743
1000	95.3125	0.1606083363	95.1935394	95.29379003
1050	93.65234375	0.1927562803	95.39404066	95.42374455

Updated Neural network

In earlier back propagation neural network we simply selected a set of features for model training and testing. But that model was too naive, still performing good but it would have been better if we can do some feature engineering that will help the neural network perform a little better. The feature preprocessing steps are as follows:

1. Group all the netflow rows by Source Address.

   In this step we gather all the netflows of a particular source IP address and club them together. This step is similar to clustering solely on the basis of the source IP address.
2. For every unique source IP address we count the number of unique destination IP address. In other terms we are getting the number of destinations a particular source IP address has contacted.
3. For every unique source IP address we count the number of unique destination port used. Similar to destination IP address the destination ports are also counted.
4. Number of netflow count for a unique source IP address.
5. Number of unique protocols used by a source IP address.
6. Whether the source IP uses UDP port.
7. Whether the source IP uses TCP port
8. Total number of bytes used by a source IP
9. Total number of packets used by source IP
10. For all flow in the group if there exist any malicious flow our modified Neural network is defined as

```
def multilayer_perceptron(x, weights, biases,keep_prob):
        # Hidden layer with RELU activation
        layer_11 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
        layer_11 = tf.nn.sigmoid(layer_11)
        layer_11 = tf.nn.dropout(layer_11, keep_prob)
        layer_12 = tf.add(tf.matmul(x, weights['h2']), biases['b2'])
        layer_12 = tf.nn.tanh(layer_12)
        layer_12 = tf.nn.dropout(layer_12, keep_prob)
        layer_1 = tf.multiply(layer_11, layer_12)
        layer_1 = tf.nn.sigmoid(layer_1)
        layer_1 = tf.nn.dropout(layer_1, keep_prob)
        out_layer = tf.matmul(layer_1, weights['out']) + biases['out']
        return out_layer
```

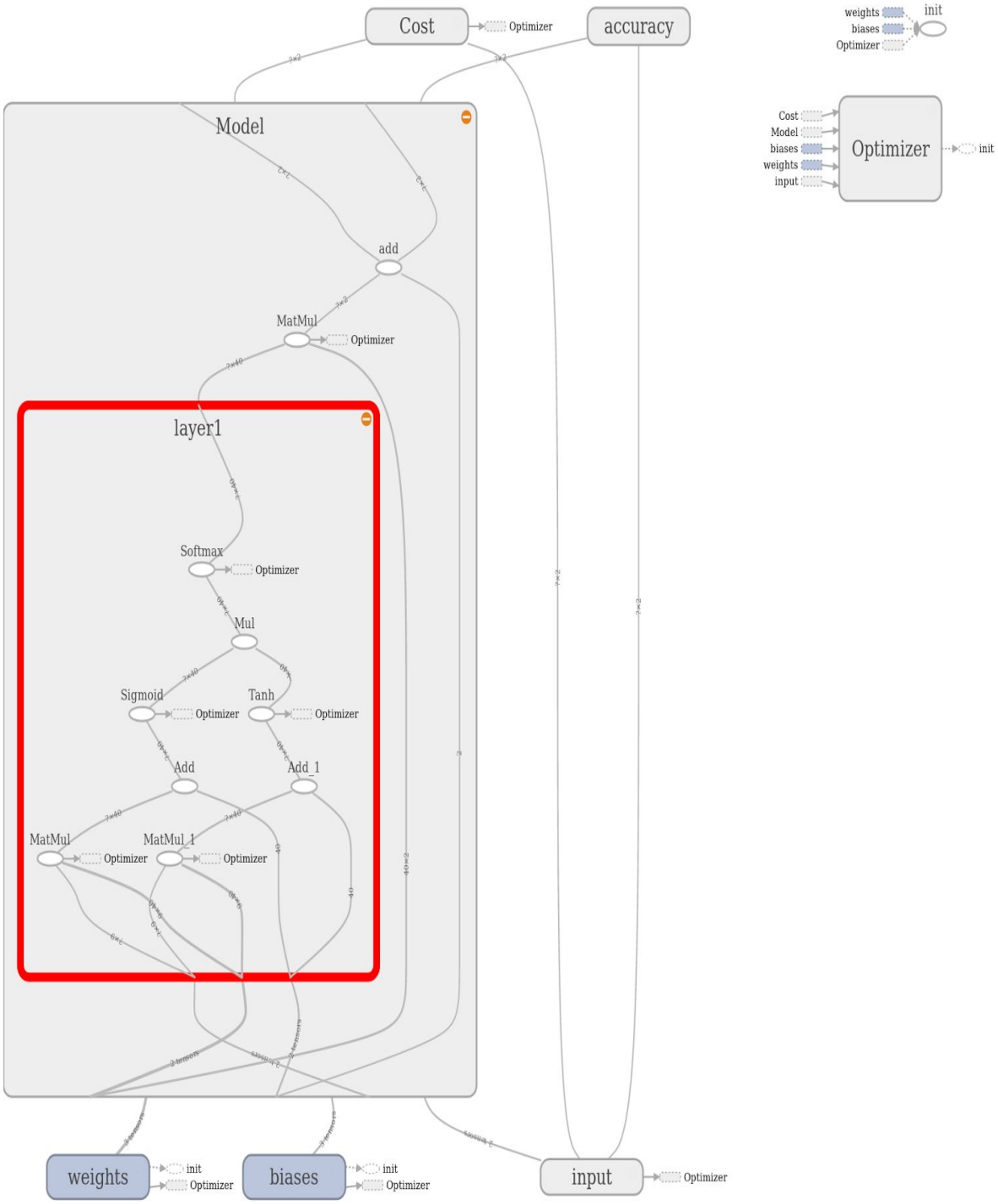

Figure 20 (TensorGraph)

Figure 21 (Tensorgraph Simplified)

Results

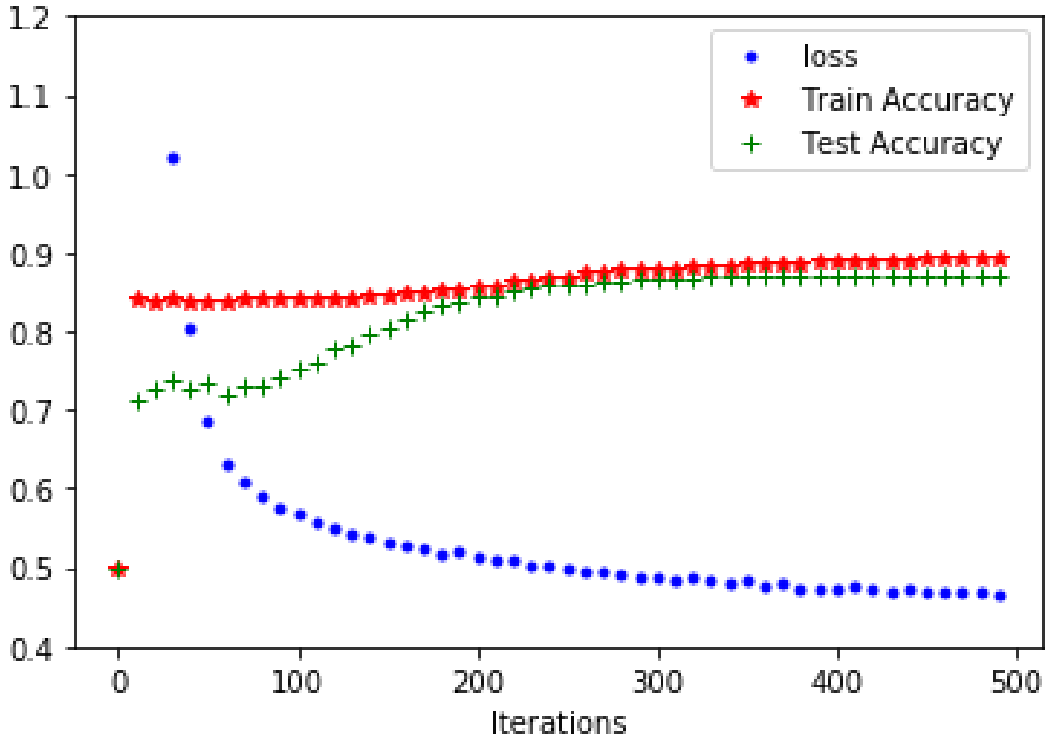

Figure 22 (Accuracy vs Iterations)

Scenario 1

When complete CTU dataset is used

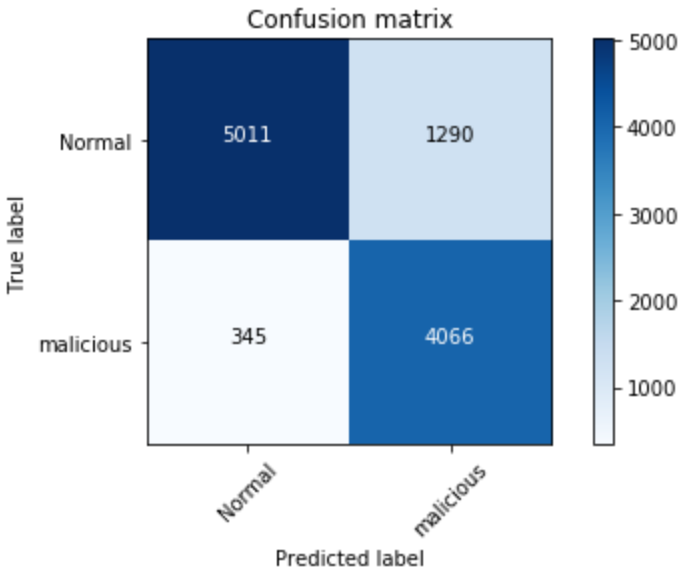

Figure 23 (Full Dataset Confusion Matrix)

Measure	Value	Derivations
Sensitivity	0.933	TPR = TP / (TP + FN)
Specificity	0.8051	SPC = TN / (FP + TN)
Precision	0.8272	PPV = TP / (TP + FP)
-ve Predictive Value	0.9231	NPV = TN / (TN + FN)
False Discovery Rate	0.1949	FPR = FP / (FP + TN)
False Discovery Rate	0.1728	FDR = FP / (FP + TP)
False Negative Rate	0.067	FNR = FN / (FN + TP)
Accuracy	0.869	ACC = (TP + TN) / (P + N)
F1 Score	0.8769	F1 = 2TP / (2TP + FP + FN)

Matthews Correlation Coefficient	0.7442	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Conclusion

In this project we identified the bots from a group of computer using 3 different machine learning methods by trying to learn the pattern behind the traffic flow from the bots. We concluded that Neural network method performed better than the simple machine learning methods. As expected the neural network method learns the complicated non-linear pattern which are hard to identify and model by hand. In order to remove the feature engineering by hand this neural network strategy proved to be very effective. In the first method we get an accuracy of 93 %, while comparing this to neural network model which gives an accuracy of 95%. This 2% accuracy increase is due to self learning of features in the network.

Future Work

Future work includes real time implementation of this method, which will include firmware programming for the router to gather the data and send it to the cloud, large scale implementation and optimization of the models proposed.

Table of figures