A System Approach to Network Modeling for DDoS Detection using a Naive Bayesian Classifier

R Vijayasarathy
Society for Electronic Transactions and Security
Chennai, India
vijayasarathy@setsindia.net

Balaraman Ravindran
Department of Computer Science and Engineering
IIT Madras, India
ravi@cse.iitm.ac.in

S V Raghavan
Department of Computer Science and Engineering
IIT Madras, India
svr@cs.iitm.ernet.in

***Abstract*—Denial of Service(DoS) attacks pose a big threat to any electronic society. DoS and DDoS attacks are catastrophic particularly when applied to highly sensitive targets like Critical Information Infrastructure. While research literature has focussed on using various fundamental classifier models for detecting attacks, the common trend observed in literature is to classify DoS attacks into the broad class of intrusions, which makes proposed solutions to this class of attacks unrealistic in practical terms. In this work, the approach to a carefully engineered, practically realised system to detect DoS attacks using a Naive Bayesian(NB) classifier is described. The work includes network modeling for two protocols – TCP and UDP.**

I. Introduction

Denial of Service attacks are a major threat to the modern electronic society. Carefully crafted attacks of large magnitude, better referred to as Distributed Denial of Service attacks(DDoS) have the ability to cause havoc at the highest level, national information infrastructure. The ease at which such attacks can be performed today is startling given the number of free online tools available in the open(Trinoo[1] , Stacheldracht [2] etc.). An ideal situation would be to differentiate between good and bad packets as generally attempted and accomplished in the case of intrusion attempts, which in the case of DoS attacks is extremely hard as only the intent differs between a genuine user and an attacker. The difficulty in classifying good and bad packets brings another hard problem to solve – even if the attack were detected, one doesn't know which class of packets have to be filtered. While solutions to some of the attacks have been around for a while, they either deal with a small subset of the problem with reduced constraints(ex, Syncookies[3]), or are too simple to be effective(declare as attack if bandwidth is choked) in the case of sophisticated attacks.

The research community has taken two perspectives of solving the problem apart from simple measures of detecting attacks:

1) Converting the problem into that of a classification problem on network state(and not on individual packets or other units) by modelling normal and attack traffic and classifying the current state of the network as good or bad, thereby detecting attacks when they happen. Classical machine learning algorithms are used to solve the problem; and
2) Cryptographic solutions like client puzzles which discourage attack attempts. Some examples include hash pre-image based client puzzles as proposed in [4] and [5].

This work focusses on using machine learning techniques for detecting DoS attacks. The problem of attack detection using machine learning techniques is not new to literature. While signature detection techniques can detect attacks based on signatures of attacks already learnt by them, Anomaly detection techniques learn network traffic from a baseline profile and detect anomalies as ones which deviate significantly from the baseline profile. Signature detection techniques are effective against known attacks while anomaly detection has the ability to detect unknown(zero-day) attacks.

Various methods have been proposed to accomplish the above objective. The include statistical approaches, like [6], which proposes a Chi-Square-Test on the entropy values of the packet headers. [7] discusses the effects of multivariate correlation analysis on the DDoS detection and presents an example of SYN flooding. [8] uses the covariance matrix to present the relationship between each pair of network feature and identifies attacks. In [9] the CUSUM algorithm is used to detect the change in the amount of packets to destination. Different other techniques taken from pattern analysis and machine learning have also been proposed in literature. For example, Markov Models [10] consider application layer DDoS attacks and use hidden semi-markov models to describe browsing behavior of users for anomaly detection at the application layer. Other classification algorithms such as Support Vector Machines [11], Genetic Algorithms [12], Artificial Neural Networks (ANN) [13] [14] and Bayesian Learning [15] have also been applied. Hybrid modeling techniques such as [12] also provide interesting results. Hidden Markov Model based DoS attack solutions have also been proposed in [16][17]. A taxonomy of DDoS attacks and defence mechanims has been documented in [18]. Recent works [19] [20] have discussed use of bayesian classifiers towards intrusion detection in general, which includes DoS attacks.

In most of these works, a few drawbacks are generally

978-1-4244-8953-4/11/$26.00

observed. In these works, DoS attacks have been classified to fall into the broad category of Intrusion attacks, and so solutions have been oriented more towards formal intrusions, including root escalation, scripting attacks etc. A major factor that is often missed by classifying DoS attacks into intrusion attacks is the enormous volume that DoS detection solutions have to handle in comparison with intrusion attacks. This fact straightaway differentiates DoS attacks from other types of intrusion, and renders learning models such as SVMs, HMMs, ANNs impractical when it comes to real time attack detection. The detection mechanism is expected to be ultra light weight to support speeds in the order of at least a few hundred Mbps. This is also well complimented by the fact that the very nature of DoS attacks(the system can absorb some attack traffic unlike intrusions) makes it detectable using simpler notions than the ones which are input parameters to models described above.

Another practical drawback of supervised/unsupervised learning models for DoS detection is the accuracy of (supposedly normal) traffic supplied to learning[1]. In practical user sites, it is a very difficult proposition to be fully confident that the input to learning is absolutely normal. The system may cause false alarms if traces of the training traffic contained abnormalities. For making practical systems out of research outcomes, additional work has to be done to eliminate considering such abnormalities which may be present in traffic supplied to learning.

Another common problem to the entire research community on DoS attacks is the lack of training data. The DoS research community depends extensively on standard datasets (KDD dataset [21] and DARPA dataset [22])for the purpose of learning and analysis, which are better suited to analysis of intrusion attacks.

So a practical system using any machine learning algorithm to detect DoS attacks should be carefully engineered to be a) Light weight; b) Accommodative of practical difficulties in learning. Different systems for DoS detection and mitigation have different approaches to the problem based on their position in the network. The system could be placed either near the target, or the source, or anywhere at an intermediate point in the network[9].

A. Contribution of this work

This work proposes an anomaly based system developed to detect DoS attacks using a Naive Bayesian classifier approach, coupled with elegant tricks to make the system useable in real-time. The system is designed to be before the target. The focus of the system is on two transport layer protocols – TCP and UDP, both of which work on a common framework of mechanisms, although with different input parameters. While other protocols in the Internet stack may be different, the authors of this work believe that the basic solution framework can still be retained while working out solutions for other protocols. The mechanisms have been tested against standard DARPA dataset [22] and inbound data to the web server of SETS captured for a period of 7 days.

[1]During the training phase, the system does learning. Learning and training are used interchangeably

II. Technical Overview

A Naive Bayes(NB) classifier is a simple probabilistic classifier based on applying Bayes' theorem with naive independence assumptions. A naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods. The fact that NB classifiers can work with small amounts of training data, and can also accommodate a large number of attributes makes them a good choice for network modeling for DoS attacks.

TCP is taken as example for illustration. The classifier consists of two phases of operation. During the training phase, the probability distribution of TCP packets is estimated based on observation of a large number of "normal" packets. This will show the bias of particular TCP packets against a few others. If there are dependencies in the occurrence of specific TCP packets, a mechanism to factor these dependencies in the probabilities is in place. During the deployment phase, based on these probabilities, the probability of occurrence of a particular packet set shall be determined. If this probability were to be lesser than a threshold probability, then that particular sequence is said to be not seen or abnormal.

A. Windowing

Windowing essentially means splitting input traffic into traffic subsets, which fit into logical entities called windows. Analysis is done on every single window(traffic subset) to help arrive at a conclusion on the network state. Windowing is essential in order to have reasonable estimate and control over the reaction time of the system for attacks and better modeling from larger training datasets.

B. Packet based Windowing

Windowing may be done in two ways:

- Time Windows: Windows based on time slices, occurring every n time units.
- Packet count Windows: Windows based on packet count, occurring for every n network packets.

The important factor that drives the choice among the two is the fact that analysis during learning should help analysis to be done real time. Packet windows are considered a better choice of the two for the following reasons:

- Packet windows provide smaller reaction times during an attack situation, because of the fact that the system may have to wait wait for the time window to complete before deciding to flag an attack(or anomaly).
- Packet windows may provide for more accurate modeling, since the number of possible events to be considered is

bound to be limited, whereas the number of events to be considered may be large in time windows, since there is no control on how many packets could be received within a time window.

One obvious concern with a packet window is the fact that the periodicity of probability computations may not be known, since one doesn't know when n packets would be available for computing probabilities. This may happen particularly when there is high traffic rate at one time, and low traffic rate at another. But the server can easily handle low volume traffic and related probability computations in any case.

C. Parameters for Transmission Control Protocol(TCP)

TCP packet headers is used in the system. The following parameters were examined:

1) TCP flags
2) Payload size
3) Source/Destination Port number and count
4) Source/Destination IP number and count
5) Inter-packet time gaps
6) Total connection time till current packet
7) Number of packets in connection

It is assumed that the overall traffic is streamed for analysis based on destination details like IP address, port number etc. Also, source details are not considered because of the assumption that spoofing is the common trend among attackers.

Parameters 6 and 7 make sense only when complete connection sequence data is supplied. So if the system is utilised on-line, these attributes won't be available for on-line usage. Hence, those attributes are also excluded. Amongst the remaining 3 parameters, it was experimentally found that *TCP flags* are best suited among the rest of the attributes.

Hence, in a packet window, modeling is done based on the TCP flags set in the packet. The TCP flags field is a collection of 8-bits, each bit representing one flag. Individual flags or combinations of flags symbolise specific actions in TCP – For example, connection establishment, connection closure, requesting data etc. The different TCP flags are: SYN, ACK, PSH, FIN, URG, RST(standard TCP flags), ECE, CWR. Since combinations of these flags represent different actions with TCP, a maximum of 2^8 different combinations are to be considered to model entire TCP traffic. These combinations are referred to as observables.

It is seen that the last two flags are seldom used in IPv4 communication, and hence only 2^6 observables are considered. Each of the 64 observables could be considered separately, but on examination of traffic datasets, it is seen that packets with only very few observables among the 2^6 occur as inbound packets. This will obviously result in many zero probabilities if viewed separately, which in turn, may result in losing vital probability information because of a zero-probability event. Hence, these observables are grouped into observable groups in order to have lesser observables and events to be modeled, and to try and decrease number of zero probability events.

Based on experimentation, the following classification of packets in TCP was arrived at:

1) T_1: Packets with RST bit set(irrespective of other bits) – 32 packet types
2) T_2: SYN packets – 1 packet type
3) T_3: ACK packets – 1 packet type
4) T_4: FIN/ACK packets – 1 packet type
5) T_5: PSH/ACK packets – 1 packet type
6) T_6: Rest of the packets – 28 packet types. Includes seldom used packets and invalid packets.

Since a window is characterised by the occurence of packets belonging to either of these observable groups, the probability of a window is a function of the probability of these observable groups inside the window. Since the observable surrounding the occurrence of a number of TCP packet types is being modeled, the number of possible events depends on the size of the window, that is, the number of packets in a window. For example, if the window size is n, then there are $n+1$ events for every observable group, that is from number of packets of type T_i , $k = 0$ in that window(no packet of type T_i occurred in the window) to $k = n$(the window was full of packets of type T_i).

In order to avoid zero probabilities, laplacian smoothing is done over the data set so that there will be no zero probability event. The total number of values on which smoothing has to be performed is $6 * (n + 1)$. For typical values, it is seen that the simple smoothing technique adopted may be inappropriate under the assumption that the training data is very less. For example, for number of training windows $T = 1000$, and for window size $n = 100$, we may have to smooth 606 values, which is not acceptable given the value of T. This brings in the need for reducing the number of possible events per observable group $(n + 1)$ to a constant number of bands K, thereby reducing number of zero probability events, and also improving smoothing. This will bring down the total number of events to $6 * K$ from $6 * (n + 1)$. However while grouping events, it is to be ensured that only events which are as likely probable as each other are to be in the same group. These events may not be contiguous in nature – events $k = l$ and $k = l + 1$ may not fall into the same group.

For the purpose of implementation, it is assumed that events per group T_i have to be grouped to K such constant bands, and this number is independent of window size. For example, if SYN observable group was considered, and window size $N = 100$, then the bands for number of windows in which n SYN packets(n varies from 0 to 100) were found could be the following: First band $b_1 = \{k = 0, k = 23, k = 54, k = 13 \ldots\}$, second band $b_2 = \{k = 18, 27, 56, 99 \ldots\}$ and so on. It is to be noted that these bands are disjoint sets, and the union of these sets results in the event space. These indices may also vary from site to site.

In summary, the following are the primary goals of the training phase:

- To determine the optimal band ranges(of events) for each of the TCP observable groups. For example, if window size $n = 100$, then there are 101 possible events which have to be examined. Events which were to have been found equally likely during the learning phase are

grouped into bands and examined. A constant number of such bands is assumed and grouping is done accordingly. The bands may not be contiguous in nature.
- To learn from input traffic based on the above bands for the Naive Bayesian(NB) model and determine the probabilities for each of these bands.
- To determine threshold probability for the stream.

D. Parameters for User Datagram Protocol(UDP)

The objective of monitoring protocol headers is to evolve a distinguishing pattern in them, which will subsequently help detecting DoS attacks on the protocol. The feature about UDP is that it is connection-less, and hence ensures speedy communication. However, the header of the UDP protocol does not contain fields like flags, which will determine the state of the communication, unlike TCP, where TCP flags determine which state the connection is in. Since UDP is not connection-oriented, most DoS attacks performed on UDP are only bandwidth based attacks, essentially trying to exhaust bandwidth resources available to connect to the server. In line with this view of thought, it can be seen that header information may not be a critical parameter for detecting DoS attacks on UDP.

Hence, the parameter considered here is the Window Arrival Time(WAT) of a packet window. WAT of a packet window is the duration in which a packet window has arrived. Technically, it is the difference in time between packets P_1 and P_N of a window, where N is the size of the window and P_i is the time of arrival of packet i of the window. Due to implementation limitations in considering overlapping windows, only non-overlapping windows are being considered. During training phase, the WATs of windows are monitored, and a model evolved to accommodate these events(WATs). During the deployment phase, the probabilities of these models are used to determine the probability of an incoming window. If the probability is less than a particular threshold probability(determined out of the learning input itself), then the window is classified as abnormal.

These WATs are grouped into a constant number of bands defined by time bounds. Hence, each band is a collection of a contiguous range of WATs. The WATs of Incoming UDP traffic are collected, which helps in the creation of a constant number of time bands(defined by upper and lower bounds) inside which these WATs can be viewed. With this setup, the probability of WAT falling under each band is computed and used during detection. During detection, the WAT of each window is computed, and the probability of the window is the probability of the band inside which the WAT falls. If this probability were to be lesser than a threshold probability, then the window is considered abnormal. The threshold probability is determined by adopting methods similar to that done in TCP.

The primary goals of the training phase with respect to UDP are the following:

1) To compute per UDP stream, the upper and lower intervals of each band of intervals of WATs, given that a constant number of bands(K) exist. For example, if window size N = 100, then the different bands could be B_1 = (0–10) seconds[2] , B_2 = (11–235) seconds, B_3 = (236–499) seconds and so on. It is to be noted that the band intervals shall span the entire time.
2) To compute per stream, the probabilities of each of these bands(for example, probability of WAT in B_2 etc) at the end of the training phase.
3) To compute per stream the dynamic threshold probability associated with the stream. This threshold probability is the probability below which the window will be considered abnormal.

III. Practical Design Considerations

The most important factors driving practical design issues are the following:

1) **Availability of Training Data**: One of the most important challenges for establishing confidence over a statistical model is to have large volumes of test data. This in this context amounts to having large volumes of training data to determine what traffic patterns are part of a good network state. This is generally impractical, considering the time for which learning has to take place before the system can be deployed online, particularly in sites where the traffic is not expected to be very high at any point of time. So the system should be able to work with small training data.
2) **Authenticity of Training Data**: The general assumption in a learning based system is that the data supplied to training is generally tagged and completely normal. This will help the system to completely take all patterns seen during training as normal and determine anomalies accordingly. But in practical situations, it is very hard to guarantee such purely normal training data. The training data might even contain data pertaining to unsuccessful attack attempts, in which case, these attempts will never be classified as anomalies during the testing phase. So the system should be able to absorb some proportion of abnormal data in the training data and still be able to classify them as abnormal.
   To work around the problem, the error in "normal" traffic is quantified into the system in the form of a parameter called the *Error Proportion(t)*. This proportion indicates the proportion of training traffic(supposedly normal) that is likely to be abnormal. From experiments in limited datasets(tagged and untagged), it is seen that the proportion varies from 1% in tagged datasets like the DARPA dataset to close to 10% in other untagged datasets.
3) **Dealing with sparse training data**: In many user sites, the training data is observed to be very sparse, that is, the data contains occurrence of very less events out of a large number of possibilities. In this case, a large number of probabilities remain zero, due to which there

[2]Unit of time is assumed to be seconds.

is a possibility of losing crucial probability information because of one zero probability event.

IV. Algorithm

Further sections describe the important engineering methods used to deal with the above problems. Due to space limitation, TCP is taken as the model for description, and algorithms are explained in terms of TCP. It is to be noted that the design for UDP is more or less similar.

The system design consists primarily of two phases of operation:

1) (**Training Phase**) The system takes stream information and traffic statistics corresponding to the stream as input, and produces a data structure S which consists of the NB model probabilities for the different TCP events. This model is used in the deployment phase to determine the probability of a window(function of probabilities of individual flags). *It is to be noted that operations done during training phase are entirely offline.*
2) (**Deployment Phase**) The actual operations phase in which the system takes the stream S containing the NB model probabilities for the stream as input, and determines the state of the network at any point in time. In real time, the system takes input traffic and determines if the network state is normal or anomalous.

V. Training Phase

A. Inputs

Inputs to the training phase are the following:

1) Stream information – containing the following:
   a) Target IP address
   b) Target (canonical) host name
   c) Target port number(application layer)
   d) Target port name(canonical)
   e) Window size(N)
   f) Error proportion t of learning traffic.
2) TCP Stream traffic statistics(per stream): Traffic statistics denote the number of packets for each of the 6 TCP packet classes in a window of the stream. For example, if $N = 100$, then a typical example of statistics for one window could be the tuple (2 5 46 43 4 0).

B. Pseudo-code

Every TCP stream is captured into a single data structure called stream S, which captures all the above mentioned values. For the model probabilities, S maintains a 2-D array W which represents the actual learnt probabilities of each event for each packet type. For example, S.W [3][7] denotes the probability of observing 7 packets of type T_3 in a window. Variable C is a 1-D array which represents local count of packet types per window. W is updated based on the elements of C. Variable K denotes the number of bands of events per packet type.

Algorithms 1 and 2 describe the training phase function for TCP.

Complexity: The complexity of the algorithm is determined by the inputs to the algorithm[3], namely, a) Number of windows W for the learning phase, b) Window size N. The complexity of the algorithm, hence, is O($W * N$). For practical purposes, even N could be considered very small when compared to W, and hence the complexity of the algorithm could be generalised to O(W).

Algorithm 1: Training phase functionality for TCP

Input: Stream Information SF, Traffic statistics TF, window size N, number of bands K, error proportion t.

Output: Updated Stream Data Structure S with learnt probabilities and threshold probability for stream.

```
Initialise S with the number of windows N and error proportion t;
/* Update model probabilities                    */
S ← TCPTrain(SF,TF,S.N,K);
/* Determine Threshold probability for the stream */
S.threshold probability ← determineThresholdProbability(S.N,TF,S.t);
return S;
```

C. Smoothing

Smoothing is a common technique used in situations where the system has to deal with sparse training data. Smoothing is the mechanism of artificially injecting a trivial non-zero value (for example, assuming even before the beginning of the experiment that every event has occurred once) to every element in the data matrix in order to avoid ending up with a sparse matrix(resulting out of sparse data). Hence, smoothing is very important in the design of practical systems modeling network traffic for DoS attacks. Further information about how smoothing is performed on TCP is available in later sections.

D. The Distribution – An overview

The probability distribution considered is broadly described by the following:

1) Observables – Grouped into 6(T_1 to T_6).
2) Discrete Random Variable X_i per observable group i = number of packets of type T_i observed in a window.
3) Outcome set O_i = {0,1,2,3...N}, where N = window size.
4) The probability of observing c_i packets of type T_i in a window is $P(X_i = c_i)$.
5) $P(X_1 = c_i)$ = $\frac{number\ of\ windows\ with\ c_i\ packets}{T}$, where T = Total number of training windows.
6) Probability of a window W(c_1 packets of T_1, c_2 packets of T_2 ... c_6 packets of T_6($c_1 + \cdots c_6$ = window size

[3]Number of bands K is generally assumed to be constant and doesn't impact the complexity of the algorithm.

Algorithm 2: Function *TCPTrain(SF,TF)* – Learning Module Algorithm for TCP

Input: Stream Information SF, Traffic Statistics TF, Window size for stream N, number of bands K.
Output: Updated Stream Data Structure S with learnt probabilities.

```
1  Initialise stream S with destination IP, destination port, etc. from SF;
2  for Every window in TF do
3      Populate S.C_i s for each of the 6 T_i s;
4      for i = 1 to 6 do
           /* Update corresponding event occurence. */
5          Increment S.W[i][S.C_i];
6      end
7      Increment S.total windows in learning;
8  end
   /* Determine Optimal bands for each row of W, compute probabilities */
9  for i = 1 to 6 do
10     band ← determineOptimalBands(S.W[i],S.N,K);
       // Determine band ranges for row i
11     S ← updateProbabilities(S,K,band,i);
       // Determine probabilities for row i
12 end
13 return S;
```

$N)) = P(X_1 = c_1) * \cdots P(X_6 = c_6)$, by Naive Bayes independence assumptions.

E. Band Grouping and Optimal Band Determination

TCP is taken as the model for explaining the concept of band grouping while it applies equally on UDP.

Description: During the process of collection of statistics, the count of observable types T_1 to T_6 per window are computed. Since certain events corresponding to each of these packet types are close to improbable for normal traffic(for example, the event SYN=windowsize is close to improbable in a normal situation, as this generally indicates a SYN flooding attack), probabilities have to be smoothed in order to preserve other window related information while detection is on. In order to overcome the concerns of using simple smoothing over smaller training set, events are grouped together and modeled, so that smoothing may be done on lesser number of values. Observables are grouped into bands, and probabilities of each of these bands determined. Input to the algorithm is a collection of statistics denoting number of windows observed against each event, from 0 to window size, per TCP packet type.

Problem Statement: Given an array of numbers(size of array equivalent to window size+1) $A = a_0, a_1, a_2, a_3, \ldots a_N$, where N =window size, Divide the array into K groups, such that each element is closest in likelihood to every other element in the group.

Methodology: Initially, grouping contiguous events was considered for want of convenience during hardware based detection – If the events were non-contiguous, extra efforts have to go into determining which band this particular event belongs to. However, this kind of partitioning may result in inaccurate probabilities since a single high or low probable event will create imbalance and bias to all other events in the group. Hence, only events which are as equally likely as each other are to be grouped.

In order to group these numbers, the numbers are first sorted. Sorting these numbers brings like events closer. It is to be ensured that the like events fall into the same group. However, the number of such bands/groups is also limited. This lets only a reasonable such grouping since the best grouping in terms of like events may have bigger number of groups. In order to do efficient grouping by accommodating fixed number of groups, each group is characterised by a jump (in values) that it has with the last element of the previous group. With the sorted array, the jumps between successive numbers(it is to be seen that estimating jumps between successive numbers will clearly isolate surges into a single group, since one big number between small number neighbours will automatically induce two jumps. This situation is not desirable.) is computed. The job on hand is to estimate the boundaries of fixed number of partitions in the sorted sequence in accordance with the jumps. If j bands are allowed, then the top $(j-1)$ jumps are taken and partitions are drawn in the sorted sequence in front of them. The indices corresponding to the original sequence of numbers in the sorted sequence in each partition will form the elements of the band. The number of windows for each band is computed as the average number of windows occurring for events in the band. The probability is computed as the average number of windows by the total number of windows.

The methodology described above is presented formally in Algorithm 3.

Example: Let window size $N = 10$.

Let the set representing number of windows during learning against each event(from $n = 0$ to $n = 10$) be $A = \{500, 291, 271, 36, 222, 111, 1211, 3, 1, 31, 1\}$, corresponding to packet type T_i. Let the fixed number of bands $K = 5$.

Sorted array $A_s = \{1211, 500, 291, 271, 222, 111, 36, 31, 3, 1, 1\}$, which brings like events closer.

Jump array $J = \{711, 209, 20, 49, 111, 75, 5, 28, 2, 0\}$.

Top 4 jumps $J_s[0] \ldots J_s[3] = \{711, 209, 111, 75\}$.

Index array $I = \{0, 1, 4, 5\}$.

Sorted Index array $I_s = \{0, 1, 4, 5\}$.

Band groups: Bg_1 : $A_s[0]$(1 value), Bg_2 : $A_s[1]$(1 value), Bg_3 : $A_s[2] - A_s[4]$(3 values), Bg_4 : $A_s[5]$(1

Algorithm 3: Function *determineOptimalBands(S.W[],K)* – Optimal Band Determination Algorithm for TCP

Input: Array $A = \{a_0, a_1 \dots a_N\}$, window size for stream N,number of bands K.
Output: 2-dimensional array $band$, containing indices of per group elements in A.

$J[N] \leftarrow \{0\}$;// stores jumps between consecutive values of array A
$J_s[K-1] \leftarrow \{0\}$;// stores the top $(K-1)$ jumps.
upper,lower,nelements $\leftarrow 0$;
$I[K-1] \leftarrow 0$, $I_s[K-1] \leftarrow 0$;// stores indices in A of top $((K-1))$ jumps.
Sort array A in descending order and copy into array A_s;
/* Determining jumps between consecutive values of A_s */
for *i = 0 to N* **do**
    $J[i] \leftarrow A_s[i] - A_s[i+1]$;
end
Determine the top $(K-1)$ jumps(largest numbers) from array J. Store them in $J_s[0], J_s[1] \dots J_s[K-2]$;
Store indices of $J_s[0] \dots J_s[K-2]$ in J into array I;
Sort array I in ascending order and store in I_s;
for *i = 0 to* $K-1$ **do**
    upper $\leftarrow I_s[i]$;
    nelements $\leftarrow$ upper - lower + 1;
    k $\leftarrow$ 0;
    band[i][k++] $\leftarrow$ nelements;
    for *j = lower to (lower+nelements)* **do**
        band[i][k++] $\leftarrow$ index of $A_s[j]$ in A;
    end
    lower $\leftarrow$ upper + 1;
end
band[i][K++] $\leftarrow$ remaining elements of A;
return *band*;

value),$Bg_5 : A_s[6]A_s[10]$(5 values).

Bands[indices]: $B_1 : \{6\}$, $B_2 : \{0\}$, $B_3 : \{1, 2, 4\}$, $B_4 : \{5\}$, $B_5 : \{3, 9, 7, 8, 10\}$.

Complexity: The function determineOptimalBands() is dominated by the sorting of numbers, and hence has a theoretical complexity of O($NlogN$), although for practical implementations, N will be small and the complexity in that case would be O(N^2).

F. Probability Updation

Problem: Given 2-dimensional array band of K number of rows(that is, K number of bands), determine the probability for all possible events for the packet type.

Methodology: After determining the band indices, determining probabilities is done by the following philosophy: Probability of all events within a group should be the same, and should be the mean probability of all events. However, smoothing needs to be done prior to determining probabilities.

The method explained above is described in Algorithm 4.

Algorithm 4: Function *updateProbabilities(S,K,band,L)* – Probability determination algorithm

Input: TCP Stream structure S, 2-D array $band$ containing band indices of groups of A, number of bands K, Index of packet type L(for writing into appropriate index at S.W)
Output: Probabilities of all events updated into corresponding W index in S

avgwcount $\leftarrow$ 0;
for *j = 0 to K* **do**
    nelband $\leftarrow$ band[j][0];
    for *k = 1 to (1+ nelband)* **do**
        avgwcount $\leftarrow$ avgwcount + S.W[L][band[j][k]];
    end
    /* Smooth the event group and compute mean */
    avgwcount++;
    avgwcount $\leftarrow \frac{avgwcount}{nelband}$;
    /* Write the mean value into the corresponding W indices of the packet type. */
    for *k = 1 to (1+ nelband)* **do**
        S.W[L][band[j][k]] $\leftarrow$ avgwcount;
    end
    /* Compute probabilities */
    for *k = 1 to (1+ nelband)* **do**
        S.W[L][band[j][k]] $\leftarrow \frac{S.W[L][band[j][k]]}{S.total\ windows\ in\ learning}$;
    end
    avgwcount $\leftarrow$ 0;
end
return *S*;

Complexity: The complexity of the function updateProbabilities() is dominated by operations done on individual elements of each band, whose sum total is the window size N. Hence, the complexity of the function is O(N).

G. Thresholding

Problem: Given learning traffic statistics for a site, determine the threshold probability occurrence below which an event will be considered abnormal.

Methodology: For the purpose of determining a threshold probability, a technique called *Cross Validation* is adopted. The following is the philosophy behind cross validation: While there is variability in traffic in terms of some events appearing more likely than a few others, it is assumed that this variability is reflected in the learning traffic also. So, the idea is to simulate the fact that detection actually has to work on unseen data, by making some of the training data itself unseen, and get trained with the seen data. For the assumed unseen data, the detection module estimates probabilities, and a suitably

low probability estimated in detection becomes a candidate for threshold probability.

An important design consideration is to eliminate the effect of bad traffic that may be present in the training traffic. During cross validation, there is a possibility that most of "unseen" traffic set(in spite of randomly picking the traffic for the unseen set) might be bad traffic, in which case the threshold probability will be lesser than an ideal choice(since the probability of most unseen windows would be very low as estimated by the model built with good traffic). This is undesirable. So the unseen traffic should be sampled at random a number of times before a threshold probability is arrived at. The algorithm mentioned here does 10 such iterations.

A single experiment consists of training the model using a group G consisting of $\frac{9}{10}$ th of learning traffic(picked at random from the complete traffic), and estimating probabilities of a blob B consisting of the remaining $\frac{1}{10}$ th of traffic. The learning traffic(that is, window statistics) is split into 10 groups(10-fold cross validation[4]) of traffic(that is, windows) at random. The system is trained on every possible 9-group combination, and probabilities computed on the remaining group of traffic. This will result in a number of probabilities, from which a suitably low probability will be chosen as the threshold probability.

The error proportion(t) is taken as an input to the algorithm. If the error proportion were x percent, then the lower x percent of probabilities calculated in the learning-cum-estimation phase described above will be eliminated. The threshold probability will be the least probability in the set after x percent of probabilities are eliminated. The algorithm uses a Random Number Generator Rng() which will be used for determining which group a particular window statistic shall belong to.

Complexity: The number of operations to be performed during the execution of the function depends on the total number of training windows W, and is dominated by the sorting function which sorts W values. Hence the complexity of the algorithm is O($WlogW$). This appears slightly intensive in practical applications but doesn't affect the effciency since *all training phase operations are one-time operations done completely offline.*

VI. Deployment Phase

The deployment phase is very similar in functionality to the training phase in terms of computing statistics. The system takes as input the data structure S containing the NB model probabilities for the given stream, and determines if the network state for the stream is normal or anomalous at any point in time. It is important to have a light-weight deployment phase activity for the system since this phase is the online phase. The system is designed to only perform preliminary window statistics computation and a few look ups to determine the probability of a window.

[4]cross validation done by splitting input dataset into n groups is called n-fold cross validation.

Algorithm 5: Function determineThresholdProbability(S)– Dynamic Threshold determination algorithm

Input: Window size for stream N, Traffic statistics TF, Error proportion t percent
Output: Threshold probability P_t for S

```
/* Initialisation                                   */
Seed Rng();
parray[] ← 0;
/* Grouping Learning traffic                        */
Split traffic TF uniformly at random into 10 groups G_1, G_2 ... G_10;
for i = 1 to 10 do
    G ← TF − G_i;
    B ← G_i;
    /* Learn from 9/10th of traffic                 */
    S_temp ← TCPTrain(SF,G);
    /* Compute probability of remaining
       1/10th of traffic                            */
    for every window W in B do
        P ← determineProbability(W,S_temp);
        add P to parray;
    end
end
/* Sieving lower probabilities                      */
Sort array parray;
result ← parray[t/100 * parraylen + 1];
return result
```

A. Probability Computation

Probability computation involves computing statistics of a window and determining the probability at which the window would have occurred in the training phase. It is described in Algorithm 6.

Complexity: The detection mechanism involves a set of constant, light weight operations. More precisely, it includes a constant number of additions for updating TCP flag statistics(N of them) until a window of packets is received, 6 lookups to determine probabilities of individual events in the window, 5 multiplications(or additions in the case of logspace operations) to determine the probability of the window, and one comparison operation to arrive at a verdict.

B. Attack Detection

Flagging an abnormal situation as an attack is a function of a series of abnormal windows, not necessarily consecutive. For example, if attack were to be flagged after observing 5 consecutive abnormal windows(say), then the attacker can cleverly escape attack detection by keeping attack traffic just below 5.

To overcome this problem, a step based mechanism is used. At any point(until a reasonable time out period) in the course of deployment, if the number of abnormal windows goes beyond a particular number (quantified as a parameter called Abnormal Window Count(AWC)), then the system flags an

Algorithm 6: Function *determineProbability(W,S)* – Probability of a window W for stream S

Input: Stream with NB probabilities S, packet window statistics W
Output: Probability P of window W

Initialise P to 1;
Determine counts of packets with flags of all 6 groups $T_1 \dots T_6$ and update counts $C_1 \dots C_6$;
for i = *1 to 6* **do**
    P = P * S.W[i][C_i];
end
return P;

attack. This will ensure that attacks such as the one explained above will be caught. The function of the system during deployment phase is formally described in Algorithm 7.

Algorithm 7: Function *TCPDeploy(S,IT,AWC)*

Input: Stream S, Input Traffic IT, Abnormal Window Count(AWC)

/* Initialisation */
$A \leftarrow 0$;
for *every packet window* W *in* T **do**
    $P \leftarrow$ determineProbability(W,S);
    if $P <$ *Threshold probability of* S **then**
        increment A;
    else
        decrement A;
    end
    if $A \geq AWC$ **then**
        return *attack*;
    end
end

VII. Experimental Results

- Experiments were conducted on the following training datasets[5]:
  - DARPA dataset(TCP): Consisted of hundreds of streams with number of inbound TCP packets varying from a few hundreds to seventy thousand. The target stream(telnet server) with the most number of inbound packets(70,600) was taken for training.
  - SETS dataset(TCP): Consists of inbound TCP traffic to the webserver of SETS, India collected over a period of 15 days. The traffic consists of roughly half a million packets.
  - SETS dataset(UDP): Consists of inbound UDP traffic to local DNS server at SETS, India, collected over a period of 2 days. The traffic consists of roughly 64,300 packets.

[5]The system was trained using these datasets.

  For attacks, datasets were synthesised for each of these above streams.
- Two kinds of tests were performed for the purpose of determining four parameters – True and false positives, true and false negatives. The tests were:
  - False Positive Test(FPT): Dataset is fed into both the training phase and the deployment phase and the number of windows classified as abnormal is observed, to determine false positives.
  - False Negative Test(FNT): Dataset is fed into the training phase. During the deployment phase, attack traffic(anomalous event) is provided as input to system, and the number of windows classified as normal is observed, to determine false negatives.
- Critical system parameters, namely, Accuracy(Acc), False Alarm Rate(FAR), and Miss Rate(MR) were computed based on the number of true and false predictions. To understand the effect of window size on the problem, experiments were conducted with different window sizes. The results are tabulated in Table II.

WS	DARPA at t =1%			SETS at t =5%		
	Acc	FAR	MR	Acc	FAR	MR
50	98.3%	2.3%	0	97%	7%	0
100	98.6%	2%	0	97.4%	6.2%	0.06%
200	98.7%	1.8%	0	97.1%	5%	1.1%

TABLE I
Experimental Results for TCP

WS	SETS at t =1%		
	Acc	FAR	MR
100	99.5%	0.4%	0.1%
200	99.2%	0.5%	0.6%

TABLE II
Experimental Results for UDP

With the tagged DARPA dataset(at an error proportion of 1%), there is a uniform trend in performance. With increasing window size, the accuracy becomes better, false alarms lesser. The miss rate is zero, that is, no attacks were missed. But with the untagged SETS dataset, it is safe to assume a slightly bigger error proportion of 5%, and the performance is slightly different from the DARPA dataset. With increasing window size, the false alarms become lesser but the miss rate increases marginally. Although untagged datasets show a slightly different pattern in performance, it could be concluded that *larger window sizes are bound to improve system performance*.

Experiments were later conducted on a partially tagged DARPA dataset[6] containing predominantly normal traffic mixed with some attack traffic on the above TCP stream. While the training dataset remained the same, the new dataset was used as the deployment dataset. Around 7.4% of windows(consisting of various attacks) were detected as anomalies

[6]Attacks on a particular stream are not known although prevalent attacks on the dataset on certain classes of IPs is mentioned.

even though the system was not specifically tuned for them. While an AWC of 5 to 10 works for flooding attacks in general, *in order to detect a variety of attacks, the AWC may have to be tuned appropriately.*

Remarks

- For the purpose of analysing real-time latency, the detection algorithm was implemented on a Virtex 4 FPGA operating at 65Mhz. It is observed that the maximum per-packet latency is 4 micro seconds, and the attack detection time is roughly 3 seconds for an AWC of 5.
- In an attempt to compare the performance of the proposed method with other methods in literature, it was observed that different works have been done in different testing environments, including datasets used, number of features, performance parameters reported, constraints on efficiency reports etc. Here are a few excerpts of the variety of environments in which different works are conducted:
  - [20] does Naive Bayes modeling on the KDD99 dataset, utilising 41 attributes and reports for DoS attacks a detection rate of 99.75% and a false positive rate of 0.04%.
  - [17] does HMM based training on the DARPA99 dataset, testing on a mix of DARPA99 dataset traffic and DDoS traffic injected at (supposedly)arbitrary times, reports a detection rate of 100% with zero false alarms, subject to the condition that the length of the input observation sequence in the constant time t is 120, while their Discrete Reinforcement Learning algorithm reports 79.2% Detection Rate.

  Owing to the above, it is understood that comparison efforts of proposed system with other systems may be inaccurate.

VIII. Conclusion and Future work

A practical approach to network modeling for DoS attacks using Naive Bayesian Classifiers has been proposed. The resulting system has been prototyped independently and is verified to be light weight and working close to line speeds(Only testing limitations don't let the authors claim that the system works at line speeds). The system has been tested with limited number of datasets. An avenue of research lies in making the user parameters more friendly – For example, only a seasoned network administrator will be able to provide a reasonable value to the error proportion t. In more critical applications where even successful attacks of small magnitude can cause catastrophic effects, it is important to improve the system performance to catch literally all attack traffic. Another important area of concern is addressing an attack situation – In terms of making a robust and fool proof practical system, attack mitigation is a wide topic of research.

Acknowledgment

The authors would like to thank Rangadurai Karthik for useful debugging exercises on software, and the hardware implementation team at SETS for implementing the detection algorithm in FPGA.

References

[1] M. Yamamura, http://service1.symantec.com/sarc/sarc.nsf/html/W32.DoS.Trinoo.html.

[2] D. Dittrich, "The stacheldraht distributed denial of service attack tool," http://staff.washington.edu/dittrich/misc/stacheldraht.analysis, December 1999.

[3] D. Bernstein, "Syn cookies," http://cr.yp.to/syncookies.html.

[4] A. Juels and J. G. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *NDSS*, 1999.

[5] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *Lecture Notes in Computer Science*. Springer-Verlag, 2000, pp. 170–177.

[6] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to ddos attack detection and response," in *Proc. DARPA Information Survivability Conference and Exposition, 2003*, vol. 1, Apr. 2003, pp. 303–314.

[7] S. Jin and D. Yeung, "A covariance analysis model for ddos attack detection," in *Communications, 2004 IEEE International Conference*, China, Jun. 2004, pp. 1882–1886.

[8] S.-Y. Jin and D. Yeung, "Ddos detection based on feature space modeling," in *Proc. of 2004 International Conference on Machine Learning and Cybernetics*, vol. 7. IEEE Press, Aug. 2004, pp. 4210–4215.

[9] Z. Zhou, D. Xie, and W. Xiong, "A novel distributed detection scheme against ddos attack," *Journal of Networks(JNW)*, vol. 4, no. 9, pp. 921–928, Nov 2009.

[10] Y. Xie and S.-Z. Yu, "A novel model for detecting application layer ddos attacks," in *IMSCCS '06: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - Volume 2 (IMSCCS'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 56–63.

[11] J. Seo, C. Lee, T. Shon, K.-H. Cho, and J. Moon, "A new ddos detection model using multiple svms and tra," in *Machine Learning and Cybernetics*, vol. 3823. Springer, 2005.

[12] T. Shon, Y. Kim, C. Lee, and J. Moon, "A machine learning framework for network anomaly detection using svm and ga," in *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, Nagoya, Japan, Jun. 2005, pp. 176–183.

[13] D. Gavrilis and E. Dermatas, "Real-time detection of distributed denial-of-service attacks using rbf networks and statistical features," *Comput. Netw. ISDN Syst.*, vol. 48, no. 2, pp. 235–245, 205.

[14] Y. Xiang and W. Zhou, "Mark-aided distributed filtering by using neural network for ddos defense," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, Nov. 2005, p. 5pp.

[15] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *KDD 02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 376–385.

[16] J. Kang, Y. Zhang, and J. bin Ju, "Detecting ddos attacks based on multi-stream fused hmm in source-end network," in *International Conference on Cryptography and Network Security*, 2006.

[17] X. Xu, Y. Sun, and Z. Huang, "Defending ddos attacks using hidden markov models and cooperative reinforcement learning," in *PAISI'07: Proceedings of the 2007 Pacific Asia conference on Intelligence and security informatics*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 196–207.

[18] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, 2004.

[19] K.-C. Khor, C.-Y. Ting, and S.-P. Amnuaisuk, "From feature selection to building of bayesian classifiers: A network intrusion detection perspective," *American Journal of Applied Sciences 6 (11)*, pp. 1949– 1960, 2009.

[20] D. M. Farid, N. Harbi, and M. Z. Rahman, "Combining naive bayes and decision tree for adaptive intrusion detection," *CoRR*, vol. abs/1005.4496, Apr. 2010.

[21] "Kdd cups 1999 data," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, October 1999.

[22] "Darpa intrusion detection datasets," http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html.