January 2007

Design and Implementation of Parallel Anomaly Detection

Shashank Shanbhag
University of Massachusetts Amherst

DESIGN AND IMPLEMENTATION OF PARALLEL ANOMALY DETECTION

A Thesis Presented

by

SHASHANK SHANBHAG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

September 2007

Electrical and Computer Engineering

DESIGN AND IMPLEMENTATION OF PARALLEL ANOMALY DETECTION

A Thesis Presented

by

SHASHANK SHANBHAG

Approved as to style and content by:

Tilman Wolf, Chair

Weibo Gong, Member

Aura Ganz, Member

C.V.Hollot, Department Chair
Electrical and Computer Engineering

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Worms and malware pose an increasing risk to today's networks. The growing sophistication of today's systems has greatly increased the speed and damage potential of such attacks. To stop worms and malware, first you must know about them. In today's rapidly evolving networks, where attackers are often one step ahead of the products designed to thwart them, anomaly detection is an important innovation. Many vendors rely on signature detection to find network-borne threats. Customers often have to wait days to get a working signature for a new worm, leaving their networks vulnerable in the most critical period during a worm's release. Network behavior analysis is one of the most robust and scalable security technologies. At the core of network behavior analysis are anomaly-based algorithms used to identify emerging threats.

Network anomalies can arise due to various causes, some of which are network overload, malicious Denial of Service attacks, and network intrusions that somehow disrupt the normal delivery of network services. Typically, each of these disrupt the normal behavior of some network data. Normal network behavior is dependent on several factors such as dynamics of the network in terms of volume of traffic, type of data and the types of applications. Commercially available network management systems monitor a set of data to detect anomalies. Typically, a human network manager observes the alarm conditions to determine the status of the network. These conditions represent deviation from normal network behavior and can possibly occur during an anomalous event. This can result in degradation of performance in the

network. Thus, an anomaly can be associated with abrupt changes in the measured data, the duration of which varies with the nature of the anomaly. This is where anomaly detection finds application.

Anomaly detection is basically described as an alarm for strange system behavior. Dorothy Denning describes in her paper - An Intrusion Detection Model, a model for building an "activity profile" of normal usage over an interval of time. Once done, the profile can be compared against the present state in real time. Anything that deviates from the baseline, or the norm, is logged as anomalous. Anomaly Detection Systems are quite different from Intrusion Detection Systems although essentially both look for suspicious behavior. Finally, the result is the same - a suspicious event is flagged and sent to the administrator. IDS systems are analogous to Misuse Detection systems wherein there is a predefined set of rules or filters crafted to detect a specific, malicious event. However, an ADS operates only from the baseline of benign activity. The differences are there for all to see. An IDS is designed to catch events that are on its list. Anything outside this list will not be recognized. In contrast, an ADS can detect new, unknown and unlisted events. Figure 1.1 shows different types of IDS depending on the different techniques and characteristics.

Network anomalies can be broadly classified into two types.

1. Network failures : E.g. A web server could fail if there is an increase in the number of requests to the server.

2. Security Related: E.g. Denial of Service attacks and Network intrusions.

The main objective of the thesis is to show that multiple anomaly detection algorithms can be implemented in parallel to effectively characterize the type of traffic causing the abnormal behavior. The logs are obtained by running six anomaly detection algorithms in parallel on the Network Processor. The six Anomaly Detection Algorithms used to identify aberrant behavior in the network are:

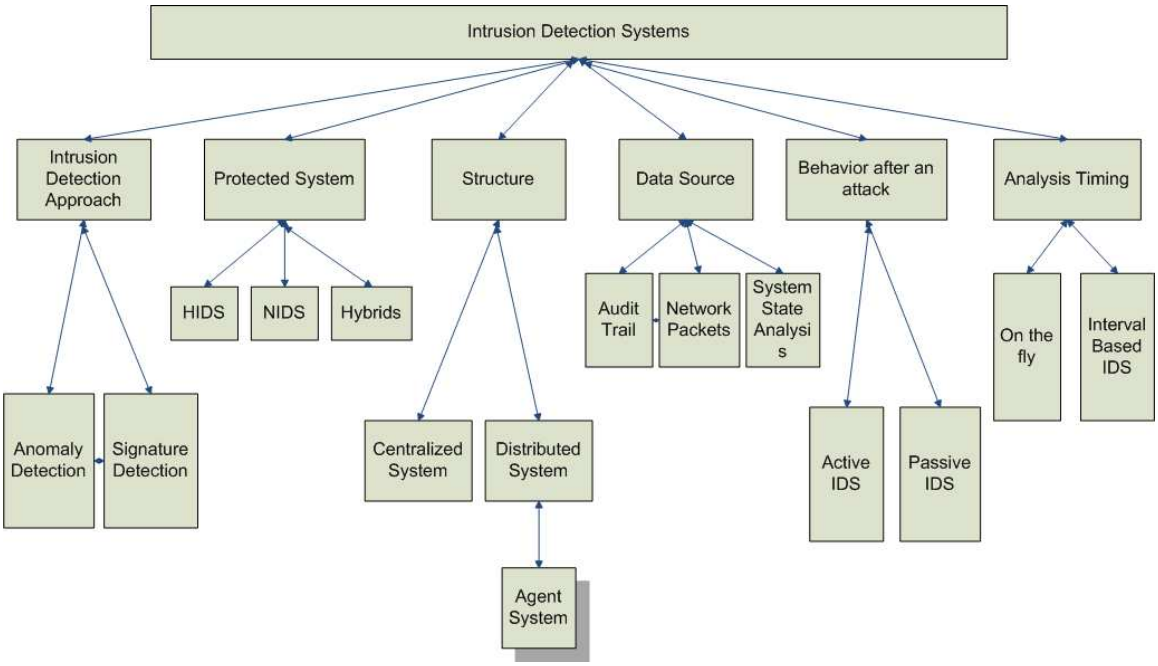

Figure 1.1. Intrusion Detection Systems

1. Holt Winter based Forecasting model proposed by Brutlag J.D. [4] which captures the history of the network traffic variations and predicts the future traffic rate in the form of a confidence band.

2. Behavior-based anomaly detection method proposed by Yu Gu [20] et al. that detects network anomalies by comparing the current network traffic against a baseline distribution.

3. Adaptive Threshold Algorithm [17], a straightforward and simple algorithm that detects anomalies based on violations of a threshold that is adaptively set based on recent traffic measurements.

4. Cumulative Sum Algorithm, [11], a proven statistical algorithm that maintains the cumulative sum of the deviations from a reference value, which is usually the mean of the time process.

5. Averaging Algorithm [16], another straightforward and simple algorithm that mainly averages the past 60 values. This is taken to be the predicted value which is compared to the present observed value.

6. Exponentially Weighted Moving Average, which applies weighting factors that decrease exponentially. Thus the most recent observations are given a greater weightage than the older observations.

CHAPTER 2

RELATED WORK

A variety of tools have been developed for the purpose of network anomaly detection. Some detect anomalies by matching the traffic pattern or the packets using a set of predefined rules that describe characteristics of the anomalies. Examples of this include many of the rules or policies used in Snort [1] and Bro [12]. The cost of applying these approaches is proportional to the size of the rule set as well as the complexity of the individual rules, which affects the scalability of these approaches. Furthermore they are not sensitive to anomalies that have not been previously defined. Our work is a behavior based approach and requires little computation. Deri et al [9] show that in every network there are some global variables that can be profitably used for detecting network anomalies, regardless of the type of users and equipment. The main idea is the design of an IDS that uses both Signature based and Anomaly based detection. Barford et al. [3] use wavelet analysis to remove from the traffic the predictable ambient part and then study the variations in the network traffic rate. Network anomalies are detected by applying a threshold to a deviation score computed from the analysis. Thottan and Ji [18] take management information base (MIB) data collected from routers as time series data and use an auto-regressive process to model the process. Network anomalies are detected by inspecting abrupt changes in the statistics of the data. Wang et al. [19] take the difference in the number of SYNs and FINs (RSTs) collected within one sampling period as time series data and use a nonparametric Cumulative Sum (CUSUM) method to detect SYN flooding by detecting the change point of the time series. [5] implements two of

the algorithms -Entropy and Holt Winter in parallel and online to know more about the types of anomalies generated and narrow down the traffic that is causing those anomalies. Zhang et al [2], describe the use of Change Point monitoring to detect Denial of Service Attacks. The objective of Change-Point Detection is to determine if the observed time series is statistically homogeneous, and if not, to find the point in time when the change happens. Non-parametric CUSUM is again used for the detection of DoS attacks. Zou et al [6], introduce a methodology for fast detection of internet worms called "trend detection". Its based on the fact that a worm, in an early stage, propagates exponentially with a constant, positive exponential rate. The system attempts to detect this trend.

CHAPTER 3

SYSTEM DESIGN

3.1 Motivation for online passive measurement

Measurements are important for managing and understanding computer networks. Measurements can provide insight into correct and faulty network behavior, and provide us with a basis for traffic and performance modeling. Measurement and monitoring tools are widely deployed in the Internet infrastructure. There are various approaches to monitor the network, two of the most common being passive and active measurement.

3.2 Passive Measurement

The passive approach uses devices to watch the traffic as it passes by. These are special purpose devices such as a Sniffer, or OCxMon, or they can be built into other devices such as routers, switches or end node hosts. The passive measurement devices are polled periodically and information is collected to assess network performance and status. One thing to note here is that this approach does not increase the traffic on the network for the measurements. It is also extremely useful in network troubleshooting, but is rather limited when it comes to emulating error scenarios or isolating the exact fault location. There are potentially two ways of deploying a passive measurement node.

1. Offline. In this method all the packets seen on the link are archived and then post-processed by running the application on the trace.

2. Online. In this method the packets are processed on transit and required statistics are collected/updated on a per packet basis.

The choice of going for an online or offline passive measurement node depends on several factors. Offline measurements create large traces that need to be post processed to get the required statistics.As link speeds increase, this only aggravates the problem. Also it is very processing intensive and time consuming to go through such big traces and extract the required statistics. Besides, Intrusion Detection applications like detecting DoS attack, anomaly detection etc., are online by nature and cannot be implemented in offline mode.

3.3 Active Measurement

The active approach essentially injects test packets into the network or sends packets to servers and applications, and measures the service obtained from the network. Thus, it creates extra traffic, the parameters being artificial. The volume and other parameters of the traffic are fully adjustable, thus providing explicit control on the generation of packets for measurement scenarios. Thus various parameters such as sampling techniques, timing, frequency, scheduling, packet sizes and protocols, statistical quality etcetera are under user control. Thus, active monitoring allows one to test what one wants, and when one needs it.

In this work, we consider measurement and monitoring to be *passive* and *online*. i.e., results are obtained by passively observing user traffic rather than by actively injecting probing traffic. The following statistics are monitored dynamically:

- Traffic Rate
- Number of TCP packets per second
- Number of UDP packets per second

- Number of TCP-Syn packets
- Number of TCP-Rst packets
- Number of Non-TCP and Non-UDP packets
- Targeted Ports divided into classes

3.4 Motivation for parallel implementation

Any DS, either based on signature detection or anomaly detection, is essentially a burglar alarm system for the network. It enables us to monitor the network for intrusive activities. When the intrusion occurs, the system generates an alarm to let us know that the network is possibly under attack. However, the DS can generate "false positives" or "false alarms".

A false positive occurs when the DS generates an alarm from normal user activity. If the system generates too many false positives, consequently there will be low confidence in the capability of the DS to protect the network. This can result in a "the boy that cried wolf" syndrome: When an actual attack is afoot, no one will respond because of all the previous false positives. Thus, it is important to minimize the number of false positives that a DS generates. *For instance, consider a DS system that generates a large number of alerts (say 15000). Also, lets say the number of false alarms is large as well, in thousands. Consequently, we find that this translates to the manual filtering and analyzing of the generated alerts. Thus, the administrator: loses confidence in the reliability of the DS, is overloaded with the task to manually analyze each alert, and might lower the defence levels to reduce the number of false positives.* Tuning the DS can solve some of the False Positives problem, but tuning is not the solution. We believe that FPs occur when the DS registers the legitimate sampled traffic as an attack. Thus, there is a need to confirm that an attack is taking place before an alert is raised.

Earlier, we briefly explained the different parameters that will be monitored in order to detect an anomaly. Thus, we will be looking at different parts of the traffic and running each of the six anomaly detection algorithms on them. For example, consider a DDoS attack on a Web Server through syn flooding. We have five anomaly detection algorithms viz. Holt Winter[4], Adaptive Threshold[17], CUSUM[11], Averaging and EWMA algorithms looking at syn packets among others. Concurrently, we have the Entropy based [20] AD system looking at the various ports including port 80. Consequently, in case of the above attack, there will be an increase in the number of syn packets targeting port 80, and all the algorithms monitoring the number of syn packets and the target port will fire an alert on the *type of traffic*, tcp and syn in this case, and the *port*, port 80 in this case. Thus, we can pinpoint the exact type of traffic that is causing the anomaly and the target port on the system.

The main goal of a network IDS is to guide the analyst or administrator toward network events that are malicious. The two major approaches as discussed above are misuse detection - based on pattern matching, and anomaly detection. The shortcomings of the former are false positives, false negatives, variants and overload. Furthermore, the signature based systems cannot detect new forms of attack. Anomaly detection is the solution to the problem of detecting new attacks, as they rely on traffic analysis rather than pattern matching to detect potential anomalies. Figure 3.1 illustrates our model of the ADS.

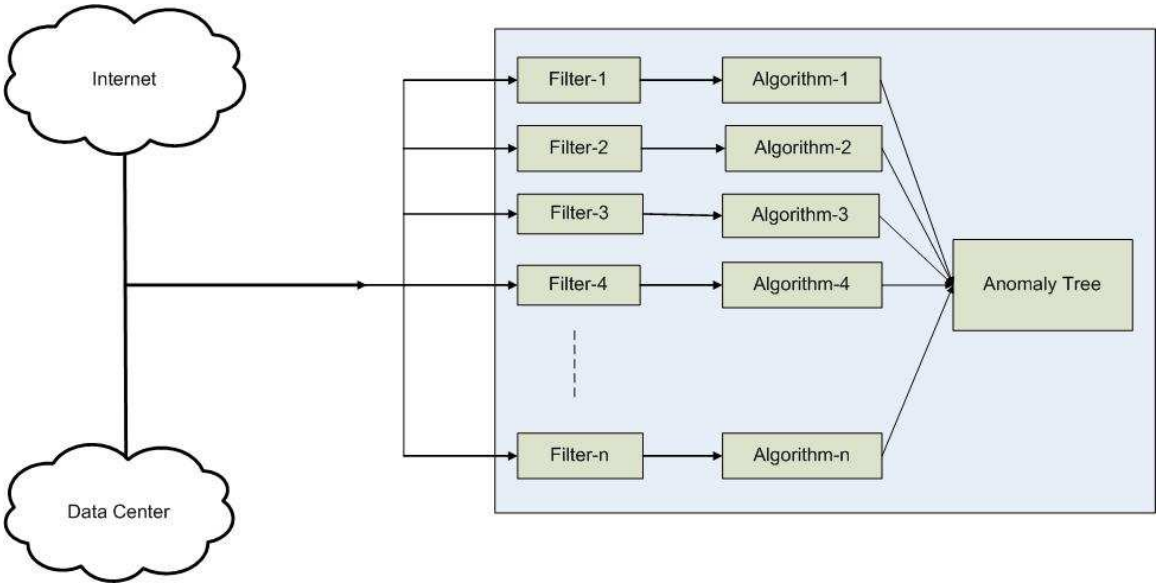

Figure 3.1. Anomaly Detection Model. Entropy only monitors ports

CHAPTER 4

THE ONLINE PASSIVE MEASUREMENT SYSTEM

4.1 Passive Measurement Node Architecture

For the experimentation and testing, the Network Processor-Based Network Measurement Node [14, 15, 13, 5] is being used. It is a passive measurement system which is capable of capturing packet traces and pre-processing them online on the measurement node. It is designed such that the statistics are updated dynamically during runtime. The system, shown in Figure 4.1, is currently installed on the Internet access link at the University of Massachusetts, Amherst.

4.1.1 Description

The Network processor based Network Measurement Node performs the following functions:

1. Packet capture and Header parsing Each of the packets is parsed to determine the sequence of headers present allowing us to consider nested protocol headers as well as different header sizes

2. Anonymization IP addresses are anonymized online during trace collection to protect the privacy of users.

3. Online Queries and Statistics Collection Packet preprocessing is done on the measurement node itself. Packets containing meta-data are prepared and pushed onto the central collection system over UDP.

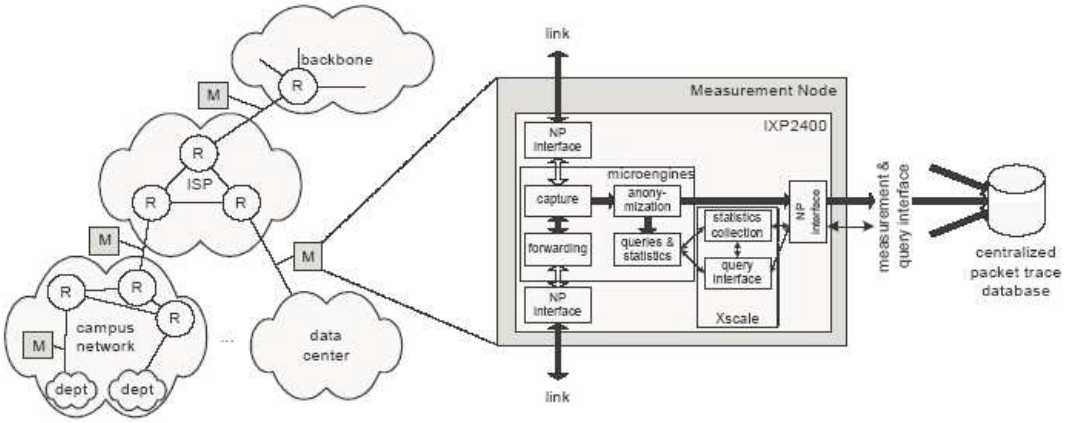

Figure 4.1. IXP2400 Based Network Measurement Node Architecture.

4.2 Implementation

The implementation is based on the IXP2400 [7] network processor found in the Radisys ENP2611 [8] card. The IXP2400 has eight microengines that are highly optimized for packet processing in the data plane, each in turn having eight threads with zero overhead context swap. The XScale processor performs all the control plane related tasks. The implementation architecture is shown in Figure 4.2

The measurement path is separated from the fast path. The fast path is responsible for any packet processing function done by the router. The system has three ports - Port 0, Port 1 and Port 2. Ports 0 and 1 handle normal traffic along the fast path. The fast path simply forwards packets from Port 0 to 1 and vice versa. The packet is then enqueued in the measurement path. The packet is dropped if the queue is full. The measurement path consists of three microengines. The first performs a filtering operation based on the query. This microengine corresponds to the IP address and anonymization stages where the packet headers are also parsed and collected, and IP addresses are anonymized. The second microengine corresponds to metrics and statistics collection stage and performs measurement related processing and collects statistics. The statistics collected include packet counts for individual protocols, namely IP, TCP, UDP, etc and distributions of layer 3 protocols, packet

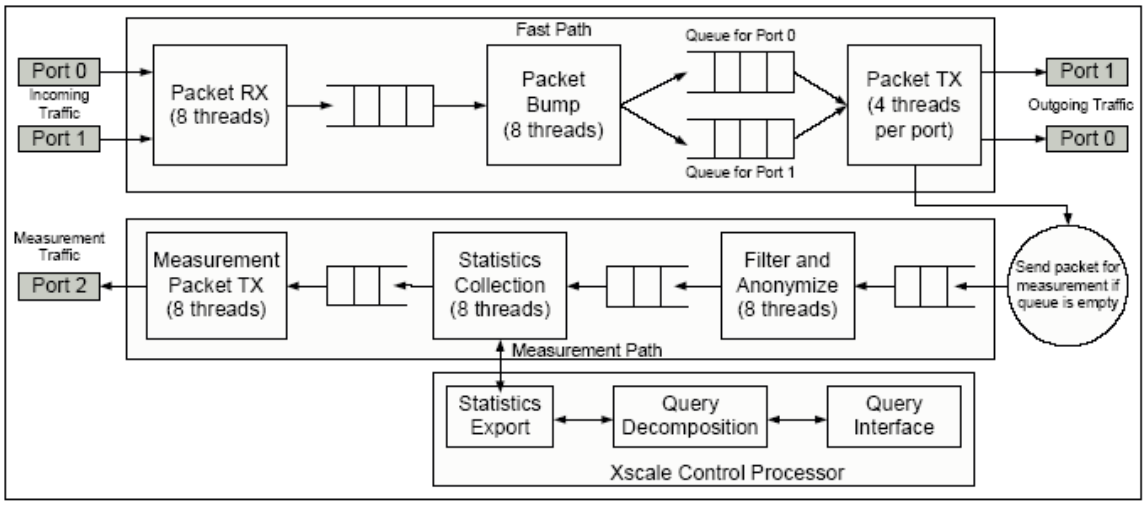

Figure 4.2. Measurement Node on the IXP 2400 NP.

size and the TCP port numbers. A "measurement packet" is generated for every packet that enters the measurement path. This packet contains a trace of the packet headers and some meta data. This is done by the third microengine. The meta data and captured headers are stored as a payload of an UDP over IP over Ethernet packet occupying 42 bytes in toto (14 bytes for Ethernet, 20 bytes for IP and 8 bytes for UDP).The format of the measurement packet is shown in Figure 4.3.

The meta data consists of the following:

1. **Node ID**: An identifier for the capture node that generated this measurement packet.

2. **Measure Length**: Stores the total length of the captured headers.

3. **Flags**: Stores the input interface number of the packet on the link that was measured.

4. **Sequence Number**: A sequence number is necessary to detect any packet loss in the network. This is because of the use of UDP to transmit the measurement traffic.

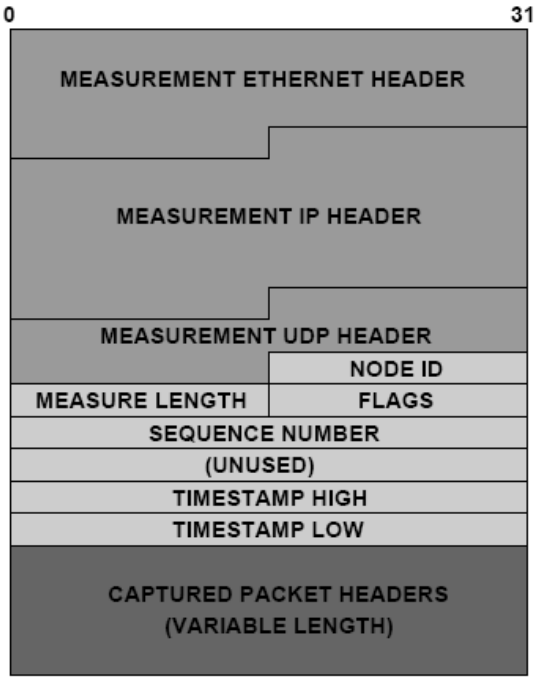

Figure 4.3. Measurement Packet.

5. **Timestamp (High and Low)**: Allows correlation of traffic between nodes. The DOME runs NTP to achieve clock accuracy in the millisecond range. The two fields store a 64-bit timestamp

In this architecture, four of the eight microengines are used to handle I/O and do some pre-measurement tasks. This leaves us with four microengines to do the actual application processing. Thus it is possible to build applications on top of it. The anomaly detection algorithms are programmed in one of the remaining microengines.

CHAPTER 5

ANOMALY DETECTION

Worms and malware pose an increasing risk toward today's networks. The growing sophistication of today's systems has greatly increased the speed and damage potential of such attacks. In today's rapidly evolving networks, where attackers are often one step ahead of the products designed to thwart them, anomaly detection is an important innovation. Many vendors rely on signature detection to find network-borne threats. Customers often have to wait days to get a working signature for a new worm, leaving their networks vulnerable in the most critical period during a worm's release.Network behavior analysis is one of the most robust and scalable security technologies. At the core of network behavior analysis are anomaly-based algorithms used to identify emerging threats. Basically there are three types of anomaly detection algorithms:

- **Protocol** - packets that are too short or have ambiguous options or violate specific application layer protocols.
- **Rate-based** - detects floods in traffic using a time-based model of normal traffic volumes. Most useful for detecting denial-of-service attacks.
- **Relational or behavioral** - detects changes in how individual or groups of hosts interact with one another on a network. For example, a normally quiet host that starts connecting to hundreds of hosts per second on the SQL port indicates a worm. Useful for a variety of threats, from worms and malware to insider misuse.

In addition, anomaly detection algorithms are also classified into:

- **Statistical Anomaly Detection Algorithms** - it determines "normal" network activity and then all traffic that falls outside the scope of normal is flagged as anomalous (not normal). These systems attempt to learn network traffic patterns on a particular network. This process of traffic analysis continues as long as the system is active. Assuming network traffic patterns remain constant, the longer the system is on the network, the more accurate it is.

- **Specification Based Anomaly Detection Algorithms** - this method uses a logic-based description of expected behavior to construct a profile. Thus, an administrator could construct a list similar to the rules and signatures. But instead of looking for misuse, these rules would ignore normal usage. However, anything outside of the specified behavior, would be marked as anomalous.

By applying these techniques, anomaly detection can identify zero-day worms, malware, and misuse.It also allows system administrators to separate the good from the bad or suspicious traffic allowing them to take preemptive action. An anomaly detection system can potentially detect an attack the first time it is used. The intrusive activity generates an alarm because it deviates from normal activity, not because someone configured the system to look for a specific stream of traffic as is the case with most intrusion detection systems.

However, the approach has three main drawbacks:

- It is too slow to detect fast spreading viruses and worms. The system is especially vulnerable during the training period where activity profiles are being generated. Most networks are diverse and are constantly changing. Thus, there is an added drawback. The ADS can be taught by intruders. For example, an attacker can send numerous SYN scans targeted at the network by using a

tool such as NMap.If continous, the system can flag this behavior as normal. Similarly with flood pings.

- The system is susceptible to an enormous number of false positives. A false positive occurs when the ADS generates an alarm from normal user activity. Anomalies can occurs at any time. As the Anomaly detection systems are looking for an anomalous event rather than an attack, it gives rise to the problem of false positives.

- Mitigation techniques are marginally effective, if any. Action is usually through zone segmentation to contain outbreaks.

One of the main motivations for this work is to resolve the problem of false positives and narrow down the traffic that is causing the anomaly. False positives are mainly the reason why anomaly detection systems are not as prevalent when compared to intrusion detection systems. In this thesis, we have tried to implement six anomaly detection algorithms in parallel on the Passive Network Measurement Node. This not only allows us to flag the possible anomalies in real time but then also allows us to post process the data and classify and characterize the anomaly. We have implemented the Holt Winter based forecasting model, Maximum Entropy based Anomaly Detection Algorithm, Adaptive Threshold Algorithm, the Averaging Algorithm, Cumulative Sum and the Exponentially Weighted Moving Average techniques on the measurement node. Each of these implementations has been described in the following sections.

5.1 Holt Winter Based Forecasting

In this section the Holt Winter Forecasting Model for detecting aberrant behavior detection given by Brutlag J.D [4] is explained. Many service network variable time series exhibit the following regularities (characteristics) that should be accounted for by a model:

- A trend over time (i.e., a gradual increase in application requests over a two month period due to increased subscriber load).
- A seasonal trend or cycle (i.e., every day bytes per second increases in the morning hours, peaks in the afternoon and declines late at night).
- Seasonal variability. (i.e., application requests fluctuate wildly minute by minute during the peak hours of 4–8 p.m., but at 1 a.m. application requests hardly vary at all).
- Gradual evolution of regularities (1) through (3) over time (i.e., the daily cycle gradual shifts as the number of evening daylight hours increases from December to June).

In addition to modeling time series regularities, model design must consider the real-time monitoring context. Complicated statistical models are unlikely to be understood by network technicians and unlikely to be feasible computationally in a real-time context.

Aberrant behavior detection is decomposed into three pieces, each building on its predecessor:

- An algorithm for predicting the values of a time series one time step into the future.
- A measure of deviation between the predicted values and the observed values.

- A mechanism to decide if and when an observed value or sequence of observed values is too deviant from the predicted value(s).

The proposed model is an extension of Holt-Winters Forecasting, which supports incremental model updating via exponential smoothing. Let $y_1....y_{t-1}, y_t, y_{t+1}....$ denote the sequence of values for the time series observed at some fixed interval. Let m denote the period of the seasonal trend (i.e., the number of observations per day). Exponential smoothing is used to predict the next value. It is a simple algorithm for predicting the next value in a time series given the current value and the current prediction. Let y_{t+1} denote the predicted value for time $t+1$, then:

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t \tag{5.1}$$

The prediction is actually a weighted average of all past observations in the time series. The premise of exponential smoothing is that the current value is most informative for prediction of the next value, and that the weight of an older observation decays exponentially as the observation moves further into the past. It is an incremental algorithm because the next prediction is obtained by updating the current prediction with the current observed value. α is the model parameter and $0 < \alpha < 1$. It determines the rate of decay $(1-\alpha)$ and the weight the current value is given during the incremental update.

Holt-Winters Forecasting is a more sophisticated algorithm that builds upon exponential smoothing. Holt-Winters Forecasting rests on the premise that the observed time series can be decomposed into three components: a baseline, a linear trend, and a seasonal effect. The algorithm presumes each of these components evolves over time and this is accomplished by applying exponential smoothing to incrementally update the components.

The prediction is the sum of the three components: The update formulas for the three components, or coefficients a, b, c are:

- Baseline ("intercept"): $a_t = \alpha(y_t - c_{t-m}) + (1 - \alpha))(a_{t-1} + b_{t-1})$
- Linear Trend ("slope"): $b_t = \beta(a_t - a_{t-1}) + (1 - \beta)b_{t-1}$
- Seasonal Trend: $c_t = \gamma(y_t - a_t) + (1 - \gamma)c_{t-m}$

As in exponential smoothing, the updated coefficient is an average of the prediction and an estimate obtained solely from the observed value y_t, with fractions determined by a model parameter (α, β, γ). Suppose that m is the period of the seasonal cycle; so the seasonal coefficient at time t references the last computed coefficient for the same time point in the seasonal cycle. The new estimate of the baseline is the observed value adjusted by the best available estimate of the seasonal coefficient (c_{t-m}). As the updated baseline needs to account for change due to the linear trend, the predicted slope is added to the baseline coefficient. The new estimate of the slope is simply the difference between the new and old baseline (as the time interval between observations is fixed, it is not relevant). The new estimate of the seasonal component is the difference between the observed value and the corresponding baseline.

The parameters α, β and γ are the adaptation parameters of the algorithm and $0 < \alpha, \beta, \gamma < 1$. Larger values mean the algorithm adapts faster and predictions reflect recent observations in the time series; smaller values means the algorithm adapts slower, placing more weight on the past history of the time series. Note that the update formulas imply that an implementation need only store the current values of the slope and intercept, and a single period of seasonal coefficients, as these stored values are replaced at each iteration. Holt-Winters Forecasting can also predict a time series further than a single time step in the future. This multi-step prediction provides a mechanism to handle missing data. Confidence bands measure deviation for each time point in the seasonal cycle; this mechanism models seasonal variability. The measure of deviation is a weighted average absolute deviation, updated via exponential smoothing.

$$d_t = \gamma(y_t - \hat{y}_t) + (1 - \gamma)d_{t-m} \tag{5.2}$$

Here d_t is the predicted deviation at time step t. The update formula for d_t is similar to that of c_t. They even share the same adaption parameter, γ. The confidence band is simply the collection of intervals $(\hat{y}_t - \delta_- \cdot d_{t-m}, \hat{y}_t + \delta_+ \cdot d_{t-m})$ for each time point y_t in the series.

Parameters δ_+ and δ_- are scaling factors for the width of the confidence band. Often, a symmetric confidence band is desired and $\delta_+ = \delta_-$. In this case, denote the common parameter δ. Given some assumptions and statistical distribution theory, sensible values of are between 2 and 3.

A simple mechanism to detect an anomaly is to check if an observed value of the time series falls outside the confidence band. However, this mechanism often yields a high number of false positives. A more robust mechanism is to use a moving window of a fixed number of observations. If the number of violations (observations that fall outside the confidence band) exceeds a specified threshold, then trigger an alert for aberrant behavior. Formally, define a violation as an observation y_t that falls outside the interval:

$$(\hat{y}_t - \delta_- \cdot d_{t-m}, \hat{y}_t + \delta_+ \cdot d_{t-m}) \tag{5.3}$$

Finally, define a failure as exceeding a specified number of threshold violations within a window of a specified number of observations (the window length).

At a cost of adding some additional overhead to the implementation, the model performs temporal smoothing within a cycle for the seasonal coefficients and deviations. The smoother used is an equal-weight moving average, with a window of 0.05m. The model parameters need to be set and tuned for the model to work well. There is no single optimal set of values, even restricted to data for a single variable. This is due to the interplay between multiple parameters in the model.

For example, consider two observations in sequence, y_t and y_{t+1}. The intercept (a), slope (b), and seasonal (c) coefficients all 'absorb' some part of the difference between y_t and $y_(t+1)$ during the exponential smoothing update. It is safe to assume some of the difference is noise, so updates to the coefficients need not account for all of the difference between y_t and y_{t+1}. The values of α, β and γ determine the relative share of the difference assigned to a changing baseline, a changing linear trend, and a changing seasonal coefficient.

Here are some guidelines for setting parameters:

1. Parameter α: At least one of α, β and γ should allow adaptation in a short time frame. As seasonal updates occur infrequently for each coefficient (once per cycle), and the goal of β is to capture a slowly changing linear trend, the most logical choice is α. Use exponential smoothing weights to make an educated choice for α. The sum of the most recent n weights is $1 - (1 - \alpha)^n$ and of course the sum of all weights is 1 (ignoring initialization). These facts can be manipulated to choose α using the formula:

$$\alpha = 1 - \exp(\frac{\ln(1 - \text{total weights as percentage})}{\text{number of points}}) \tag{5.4}$$

   For example, if one wants observations in the last 15 seconds to account for 95 percent of the weights, and observations occur at one second intervals (fifteen timepoints), then the formula yields $\alpha = 0.181$.

2. Parameter β: As the purpose of β is to capture a linear trend longer than one seasonal cycle, it is logical to choose β such that one seasonal cycle does not account for a majority of the exponential smoothing weights. The formula discussed previously applies with β replacing α. For example, if the period of the cycle is twenty seconds, at one observation every second (20 time points),

then setting $\beta = 0.00346$ will guarantee that observations within the last day account for less than 50 percentage of the smoothing weights.

3. Parameter γ: The seasonal adaptation parameter can also be selected using exponential smoothing weights using a variation of the previous formula. Note this single parameter controls both seasonal coefficient and deviation adaptation, on the assumption that seasonal trend and variability evolve together over time at roughly the same rate.

4. Parameter δ: As noted in confidence bands section, the scaling factor of the confidence bands can be chosen by appealing to statistical distribution theory. Reasonable values fall in the interval [2, 3]. Choosing 2 detects more failures (which may just mean a higher rate of false positives).

5. Window length and threshold: Given the goal of real-time monitoring, the window length should be at most on the order of 20 seconds (i.e., for one second intervals, choose a window length between 15 and 20). A higher threshold will make the model robust to false positives, but perhaps at the cost of missing true failures. These parameters are probably the most difficult to set a priori.

5.1.1 Implementation of Holt Winter Forecasting Model

The algorithm outlined above was implemented on the network processor setup and run for an eight hour long interval. The microengine of the Network Processor maintains various statistics of the packets seen in the link. The observed value for that particular statistic is calculated every second. The next expected value is calculated based on the Holt Winter Forecasting model. Confidence intervals are calculated as described in the previous section, and a violation is indicated when the observed value falls outside the confidence band. The results are shown in the Figures 5.1, 5.2, 5.3,

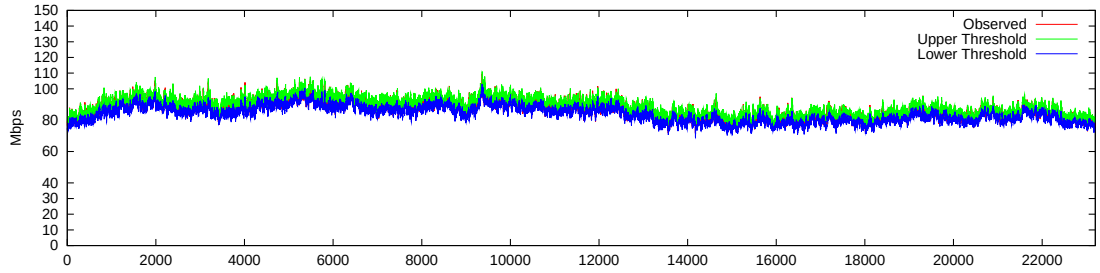

Figure 5.1. Per second Observed and Predicted Rates for Holt Winter model

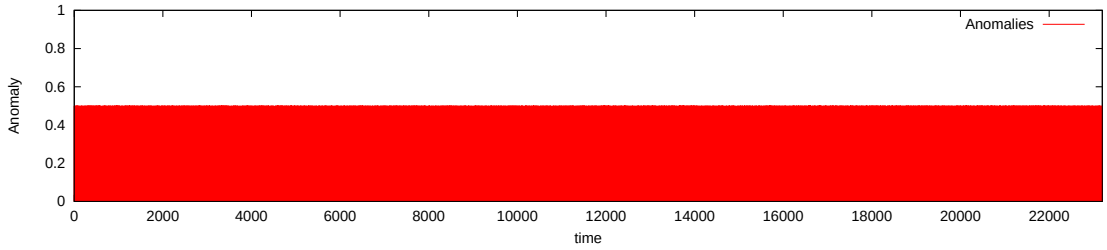

Figure 5.2. Per second interval Anomalies for Holt Winter model

5.4, 5.5 for the Holt Winter model monitoring the traffic rate in Mbps. On the x-axis, we have time in seconds.

- Per second observed and predicted rates according to the Holt Winter Forecasting model is shown in Figure 5.1. On the y-axis, we have the rate in Mbps.
- Per second violations as shown in Figure 5.2. As is evident, there are a large number of false positives.

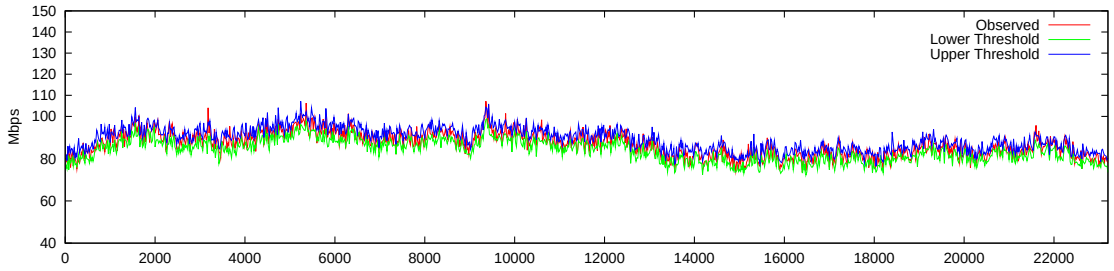

Figure 5.3. 20 second interval Observed and Predicted Rates for Holt Winter model

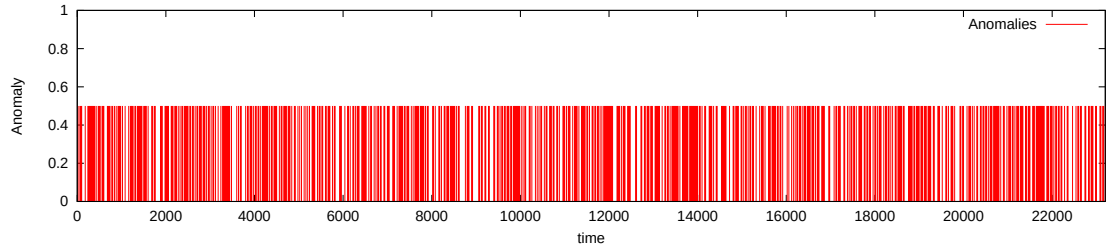

Figure 5.4. 20 second interval Anomalies for Holt Winter model

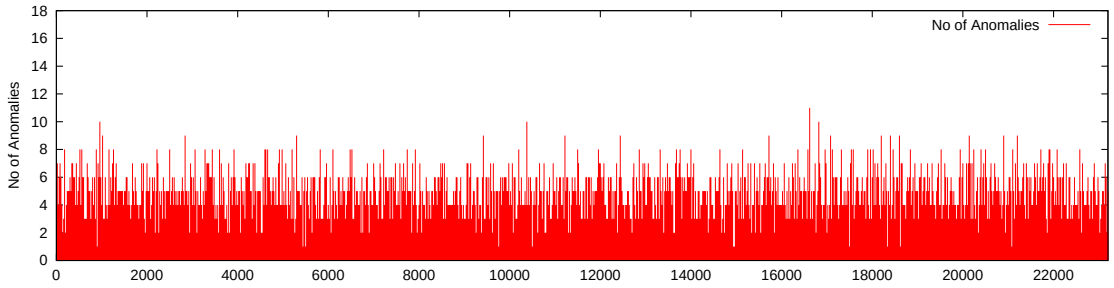

Figure 5.5. Total number of threshold violations in the 20 second interval

- Figure 5.3 shows the 20 second averaged values of the observed and predicted rates. On the y-axis we have the rate in Mbps.

- Figure 5.4 shows anomalies at instances where the number of violations in the twenty second interval, as in Figure 5.5 exceeds the threshold value.

5.2 Maximum Entropy based Anomaly Detection

5.2.1 Description

The Maximum Entropy based Anomaly Detection scheme was proposed by Yu Gu et al [20]. In this approach all the packets in the network traffic into a set of packet classes. In order to study the distribution of these packets, they divide them into a set of two-dimensional classes according to the protocol information and the destination port number in the packet header. This set of packet classes is the common domain of the probability spaces. In the first dimension, packets are divided into four classes according to the protocol related information. First, packets are divided into the

classes of TCP and UDP packets. Two other classes are further split from the TCP packet class according to whether or not the packets are SYN and RST packets.

In the second dimension, packets are divided into 506 classes according to their destination port numbers. Port numbers often determine the services related to the packet exchange. According to the Internet Assigned Numbers Authority, port numbers are divided into three categories: Well Known Ports (0 – 1023), Registered Ports (1024 – 49151), and Dynamic and/or Private Ports (49152 – 65535). Packets with a destination port in the first category are divided into classes of 8 port numbers each. Since packets with port number 80 comprise the majority of the network traffic, they are separated into a single class. This produces 129 packet classes. Packets with destination port in the second category are divided into 376 additional classes, with each class covering 128 port numbers. Packets with destination port numbers larger than 49151 are grouped into a single class. Thus, in this dimension, packets are divided into a total of $129 + 376 + 1 = 506$ classes.

Altogether, the set of two-dimensional classes consists of $4 \cdot 506 = 2024$ packet classes. These packet classes comprises the probability space. The distribution of different packets in the benign traffic according to this classification, and use it as the baseline distribution to detect network traffic anomalies.

Empirical distribution of the packets is obtained once every time slot based on the percentage of packets seen in that class to the total packets seen. The relative entropy shows the difference between the distribution of the packet classes in the current network traffic and the baseline distribution. If this difference is too large, it indicates that a portion of some packet classes that rarely appear in the training data increases significantly, or that appear regularly decreases significantly, which corresponds to an anomaly.

5.2.2 Implementation

The packet counts in each class are obtained once every second. The packet count in each class divided by the total packets seen in that time interval gives the empirical distribution of that packet class. The relative entropy shows the difference between the distribution of the packet classes in the current network traffic and the baseline distribution. If this difference is too large, it indicates that a portion of some packet classes that rarely appear in the training data increases significantly, or that appear regularly decreases significantly. In other words, this serves as an indication of the presence of an anomaly in the network traffic..

5.3 Adaptive Threshold Algorithm

5.3.1 Description

The Adaptive Thresholding Algorithm [17] is a straightforward and simple algorithm. It detects anomalies based on violations of a threshold that is adaptively set based on recent traffic measurements. Seasonal variations and trends are taken care of by using an adaptive threshold whose value is set based on an estimate of the mean number of the packets under consideration or the rate, either of which are computed from recent traffic measurements. If x_n is the observed value in the n-th time interval, and $\bar{\mu}_{n-1}$ is the mean estimated from measurements prior to n, then the alarm condition is

$$If \quad x_n \geq (\alpha + 1)\bar{\mu}_{n-1}, \quad then \quad Alarm \quad signalled \quad at \quad time \quad n. \tag{5.5}$$

where $\alpha > 0$ is a parameter that indicates the percentage above the mean value that we consider to be an indication of anomalous behavior. The mean $\bar{\mu}_n$ can also be computed over some past time window *or* using an Exponentially Weighted Moving Average of the previous measurements as given below.

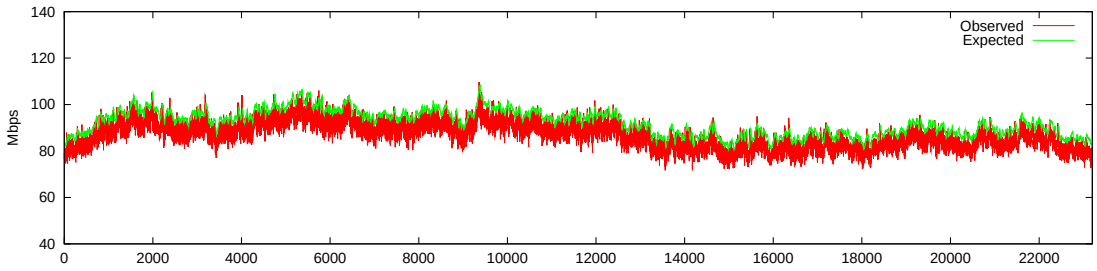

Figure 5.6. Per second Observed and Predicted Rates for Adaptive Threshold

$$\bar{\mu}_n = \beta\bar{\mu}_n - 1 + (1-\beta)x_n \tag{5.6}$$

where β is the EWMA factor.

If we apply the above algorithm directly, then it is obvious that it will yield a high number of false positives. To counter this, we can trigger an alarm only after a minimum number of consecutive violations of the threshold. Thus, the new alarm condition is given by

$$If \quad \sum_{i=n-k+1}^{n} 1_{x_n \geq (\alpha+1)\bar{\mu}_{n-1}} \geq k, \quad then \quad Alarm \quad at \quad time \quad n. \tag{5.7}$$

where $k > 1$ indicates the number of consecutive intervals the threshold must be violated for an alarm to be raised.

The tuning parameters of the adaptive threshold algorithm are the amplitude factor α for computing the alarm threshold, the number of successive threshold violations k before signalling an alarm, the EWMA factor β, and the length of the time interval over which traffic measurements (number of individual packets and the rate) are taken. The following values were considered for the parameters $\alpha = 0.5$, $k = 5$, and $\beta = 0.98$. The results are shown in the Figures 5.6, 5.7, 5.8, 5.9. On the x-axis, we have time in seconds.

- Per second observed and predicted rates according to the Adaptive Threshold algorithm is shown in Figure 5.6. On the y-axis, we have the rate in Mbps.

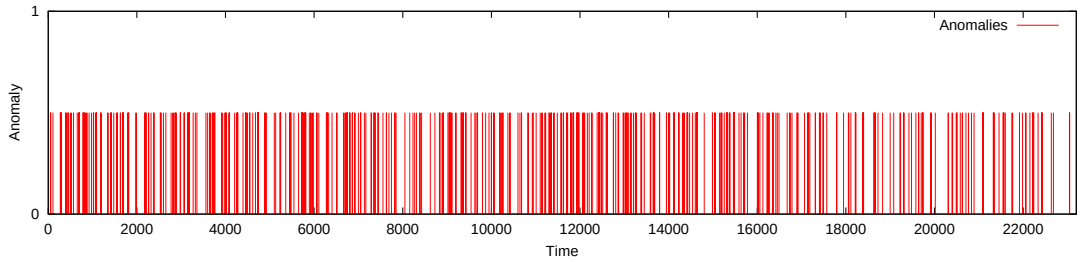

Figure 5.7. Per second interval violations for Adaptive Threshold

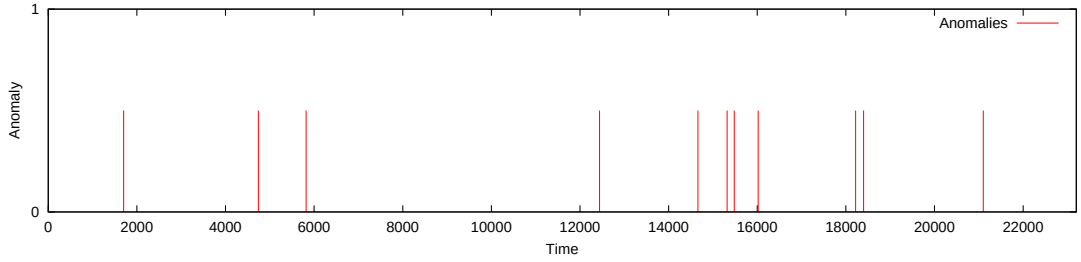

Figure 5.8. 20 second interval Anomalies for Adaptive Threshold

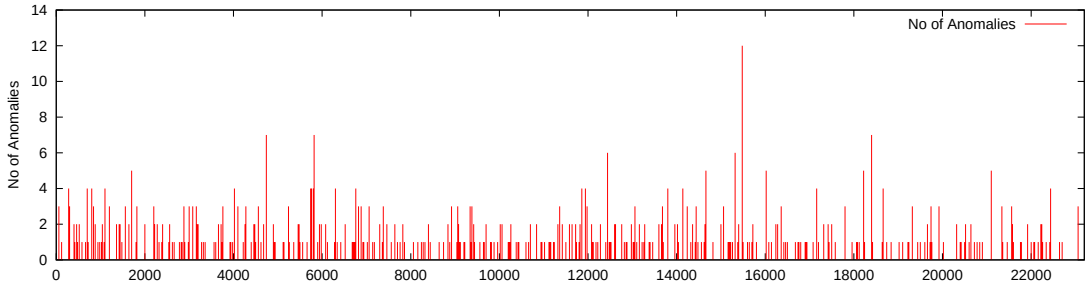

Figure 5.9. Total number of threshold violations in the 20 second interval

- Per second violations as shown in Figure 5.7. As is evident, there are a large number of false positives.

- Figure 5.8 shows anomalies at instances where the number of violations in the twenty second interval, as in Figure 5.9 exceeds the threshold value.

5.4 Cumulative Sum Algorithm

5.4.1 Description

The CUSUM technique is a sequential technique for detecting change points. Data is input to the algorithm one point at a time decision on whether a change point has occurred is made as each new piece of data is received. As its name suggests, CUSUM maintains a cumulative sum of deviations from a reference, μ , which is the mean of the process estimated in real time and periodically updated. If x_n is the n-th observation, and S_i is the i-th cumulative sum, the cumulative sum, S_i can be calculated as follows:

$$S_k = \sum_{i=1}^{k}(x_1 - \mu) = (x_k - \mu) + S_{k-1} \tag{5.8}$$

If the observations x_n are close to the mean, then the cumulative sums S_i will be around zero. However, once a shift around the mean occurs, the S_i values will increase or decrease quickly. In the above equations, we have mentioned μ to be the mean value. μ can be calculated as an Exponentially Weighted Moving Average of the previous observations as given below

$$\mu_n = \beta\mu_n - 1 + (1 - \beta)x_n \tag{5.9}$$

where β is the EWMA factor. Choose a parameter δ that is the upper bound of μ. Also, define

$$\tilde{S_n} = S_n - \delta \tag{5.10}$$

$\tilde{S_n}$ has a negative mean during normal operation. However, when an attack occurs, $\tilde{S_n}$ will suddenly become a large positive. Also, let the increase in the mean of $\tilde{S_n}$ be lower bounded by a value h.

$$Let \qquad y_n = (y_{n-1} + \tilde{S_n})^+ \tag{5.11}$$

where

$$x^+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

and y_n is defined as:

$$y_n = U_n - \min_{1 \le k \le n} U_k \tag{5.12}$$

and where

$$U_k = \sum_{i=1}^{k} \tilde{S}_k \quad and \quad U_0 = 0 \tag{5.13}$$

Thus, y_n is nothing but the maximum continuous increment until time n. A large y_n is a strong indication of an attack. Now, let $d_N(\cdot)$ be the decision at time n: 0 for normal operation and 1 for an attack. Let N be the *flooding threshold*. Then, we have

$$d_N(y_n) = \begin{cases} 0 & \text{if } y_n \le N \\ 1 & \text{if } y_n > N \end{cases}$$

The effect of introducing δ is to offset the possible positive mean in X_n, so that y_n will be reset to zero frequently and will not accumulate in time.

5.5 Average over past n prediction

The average over past n algorithm is a simplified version of the Adaptive threshold algorithm. Here, the seasonal variations are accounted for by taking the average of

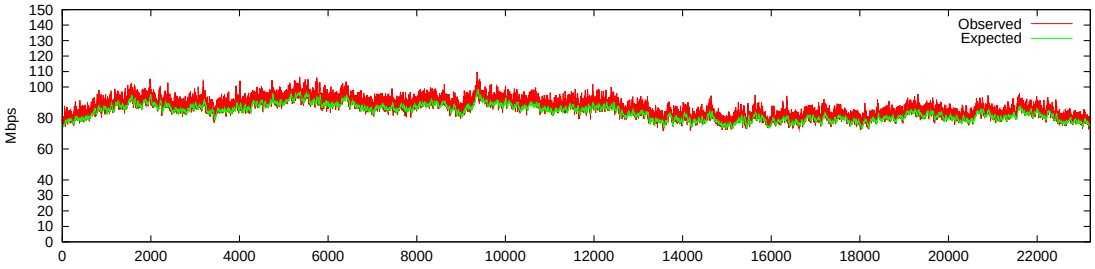

Figure 5.10. Per second Observed and Predicted Rates for Averaging Algorithm

the past n values. Thus, if x_n is the observed value in the n-th time interval, and $\bar{\mu}_n$ is the mean estimated from measurements prior to n, then the alarm condition is:

$$If \quad x_n \geq \bar{\mu}_n, \quad Alarm \quad at \quad time \quad n. \tag{5.14}$$

Again, it can be seen that this technique is susceptible to a high number of false positives if directly applied. Thus, we choose a window period over which we keep a count of the number of violations. An alarm is raised only when the number of violations over this window exceeds the threshold value set for that window. Thus, the new condition is

$$If \quad \sum_{i=0}^{t} 1_{x_n \geq \bar{\mu}_n} \geq k, \quad then \quad Alarm \quad at \quad time \quad n. \tag{5.15}$$

where t is the window period in seconds, and $k > 0$ is the minimum number of threshold violations that should be registered within the window period for the alarm to signal. The results are shown in the Figures 5.10, 5.11, 5.12, 5.13. On the x-axis, we have time in seconds.

- Per second observed and predicted rates according to the Averaging model is shown in Figure 5.10. On the y-axis, we have the rate in Mbps.
- Per second violations as shown in Figure 5.11. As is evident, there are a large number of false positives.

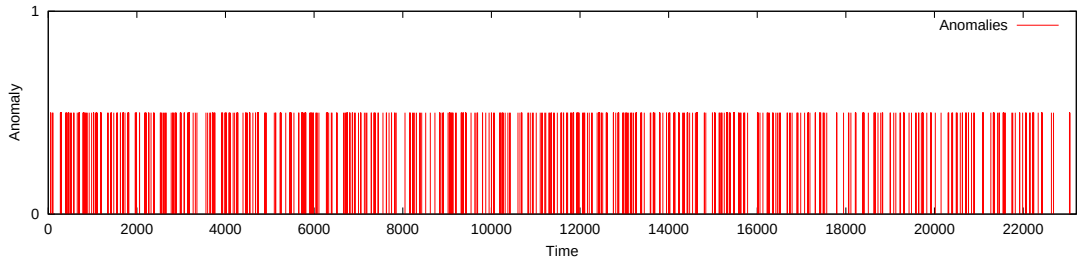

Figure 5.11. Per second interval violations for Averaging Algorithm

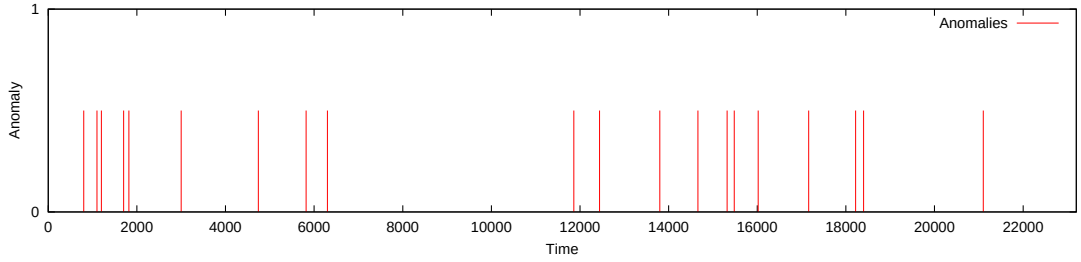

Figure 5.12. 20 second interval Anomalies for Averaging Algorithm

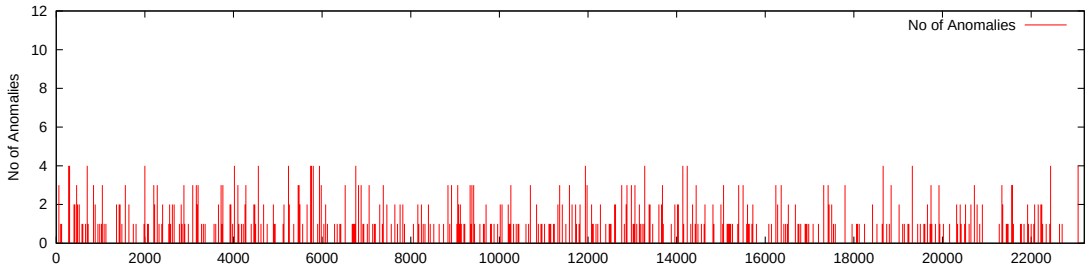

Figure 5.13. Total number of threshold violations in the 20 second interval

- Figure 5.12 shows anomalies at instances where the number of violations in the twenty second interval, as in Figure 5.13 exceeds the threshold value.

5.6 Exponentially Weighted Moving Average

An exponentially weighted moving average (EWMA) applies weighting factors which decrease exponentially. The weighting for each older data point decreases exponentially, giving much more importance to recent observations while still not discarding older observations entirely. The graph at right shows an example of the weight decrease. The degree of weighing decrease is expressed as a constant smoothing factor β, a number between 0 and 1. β may be expressed as a percentage, so a smoothing factor of 10% is equivalent to $\beta = 0.1$. The equation that defines the EWMA is

$$\bar{\mu}_n = \beta\bar{\mu}_n - 1 + (1 - \beta)x_n \tag{5.16}$$

where $\bar{\mu}_n$ is the exponentially weighted moving average of the past measurements.

5.7 Parallel implementation of Anomaly detection schemes on the IXP2400

Earlier, I briefly explained the different parameters that will be monitored in order to detect an anomaly. Thus, we will be looking at different parts of the traffic and running each of the six anomaly detection algorithms on them. For example, consider a DDoS attack on a Web Server through syn flooding. We have four anomaly detection algorithms viz. Holt Winter, Adaptive Threshold, CUSUM, Averaging and Next=Current prediction looking at syn packets among others. Concurrently, we have the Entropy based AD system looking at the various ports including port 80. Consequently, in case of the above attack, there will be an increase in the number of syn packets targeting port 80, and all the algorithms monitoring the number of

syn packets and the target port will fire an alert on the *type of traffic*, tcp and syn in this case, and the *port*, port 80 in this case. Thus, we can pinpoint the exact type of traffic that is causing the anomaly and the target port on the system. Thus, we can see that there is a need of a correlation engine. Essentially, a single data point often results in an alarm. However, a correlation engine looks at multiple data points. A network anomaly, is not enough evidence to initiate an immediate response. Once it correlates with an alert from the other algorithms over the sampling period, can the alarm be issued with a high degree of accuracy. This results in fast and accurate detection which can also initiate stoppage of attack traffic, as is typically done, without negatively impacting critical operations.

5.7.1 Implementation details

To have a better understanding of the algorithms and for experimentation we needed to recreate the traffic. Consequently, the six anomaly detection algorithms were run on a Lab setup on publicly available traces the details of which are given in detail in the following section. The lab setup consisted of a host machine with the ENP2611 card installed in it, and a replay machine which replayed the stored pcap traces. Tcpreplay was used to replay the traces over the network. Tcpreplay is aimed at testing the performance of a Network Intrusion Detection System by replaying real background network traffic in which to hide attacks. Tcpreplay allows you to control the speed at which the traffic is replayed, and can replay arbitrary libpcap traces. Unlike programmatically-generated artificial traffic which doesn't exercise the application/protocol inspection that a NIDS performs, and doesn't reproduce the real-world anomalies that appear on production networks (asymmetric routes, traffic bursts/lulls, fragmentation, retransmissions, etc.), tcpreplay allows for exact replication of real traffic seen on real networks.

5.7.2 Anomaly Trees

The basic premise of all the algorithms explained in the previous section is that a violation or an anomaly is registered when the observed value is too deviant from the calculated or predicted value for that time instant. Our technique uses this fact thus only concentrating on the degree by which an the observed value deviates from the prediction as calculated by an algorithm. All six algorithms monitor each subset of traffic. This not only enables us to pinpoint the exact subset of traffic that is causing the anomaly but also to come up with a simple score that characterizes the degree of the anomaly. For this purpose, we create what we call the *Anomaly Tree* for every time instance.

The Anomaly tree (Fig. 5.14) represents a hierarchical relationship between the various subsets of traffic as observed in the network. At the highest level is the traffic as seen in the network. A level below are the TCP, UDP and Other (Non-TCP and Non-UDP) traffic. TCP is further divided into TCP-Syn, TCP-Rst and TCP(Rest)[1]. UDP, TCP-Syn, TCP-Rst and TCP(Rest) are further divided into 506 classes each according to the Entropy algorithm [20] explained earlier.

The Anomaly tree is updated every time instance, and maintains the score for every subset of traffic calculated from the algorithm outputs. Every algorithm outputs values depicting the degree of deviation on a different scale. This necessitates the use of normalization techinques so as to enable us to compare all algorithm outputs on a similar scale. Of various normalization techniques available, we chose *Min-max* normalization. Min-max normalization subtracts the minimum value of an attribute from each value of the attribute and then divides the difference by the range of the attribute. These new values are multiplied by the new range of the attribute and

[1]TCP(Rest) is the Non-Syn and Non-Rst TCP traffic

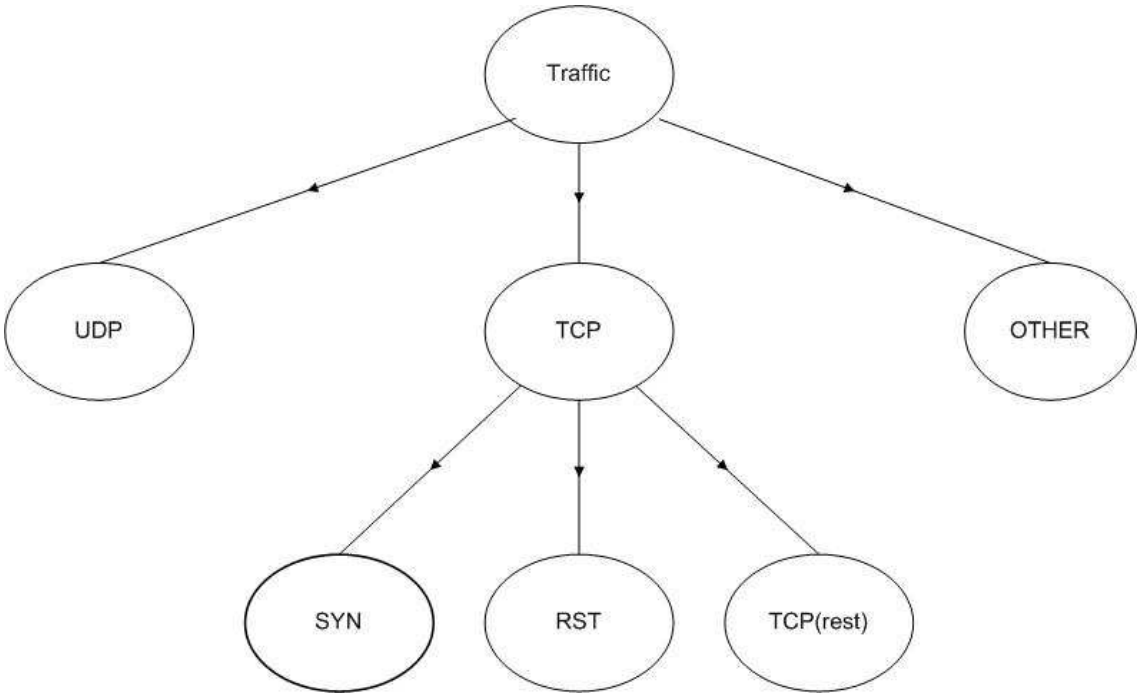

Figure 5.14. Anomaly Tree

finally added to the new minimum value of the attribute. These operations transform the data into a new range, generally [0,1].

Every node maintains the current normalized degree as output by the various algorithms. To ensure that we do not end up with six trees for every time instance for the six algorithms in parallel, but instead we only have one tree per time instance, the nodes maintain either the *Average* or the *Median* of the six outputs at every node. The observed statistics and the differences between maintaining the Average and the Median are explained in the following section.

CHAPTER 6

OBSERVATIONS

The algorithms were run on four different traces with different characeristics and varied attack traffic. This section explains in detail the observations for each of those traces along with the status of the Anomaly tree nodes at selected time instances during the experiment. The experiments are repeated for the Median and the Average case.

6.1 Trace 1

The trace was taken from the peering link at the Los Nettos network of the University of Southern California, LA, USA. The trace contains Syn portscan and portsweep traffic of varying magnitudes between 50 and 250 seconds into the trace. The entire trace lasts for about 441 seconds. The trace was repetitively replayed a number of times using tcpreplay.

6.1.1 Average of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the average of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.2. The experiment shows a correlation between the algorithm outputs and the behavior of the traffic in the trace. The aggregation model is successful in detecting the syn traffic in the trace between time instants 50 and 250 seconds and also outputs a rough metric of how deviant the observations are. The behavior of various algorithms as seen at the Syn node in the anomaly tree is shows in Figure. 6.1.

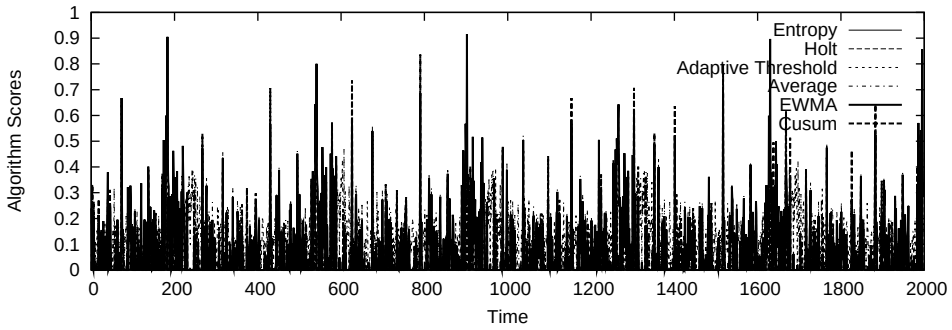

Figure 6.1. Output behavior of Algorithms as seen at the Syn node in the Anomaly Tree.

6.1.2 Median of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the median of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.3

6.1.3 Anomaly Tree

Figure 6.4 shows the status of the anomaly tree nodes before and after the change at three sets of instances. An increase in the normalized output of the algorithms as seen at a particular indicates that the node is anomalous. The anomalous nodes are shaded in with a darker shade representing a higher level of anomaly. For instance, at times 898s(before) and 903s(after the change), the Syn node value increases by 0.4 units whereas UDP node value increases by 0.1 unit. Thus, Syn node is shown in a darker shade as compared to the UDP node.

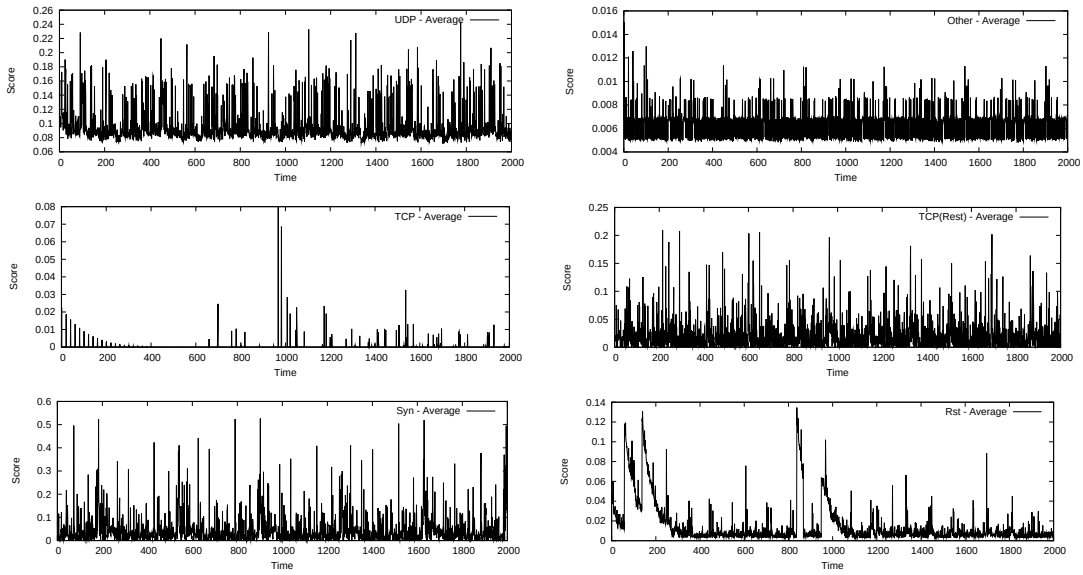

Figure 6.2. Trace 1: Average of the Algorithm outputs at various Anomaly tree nodes.

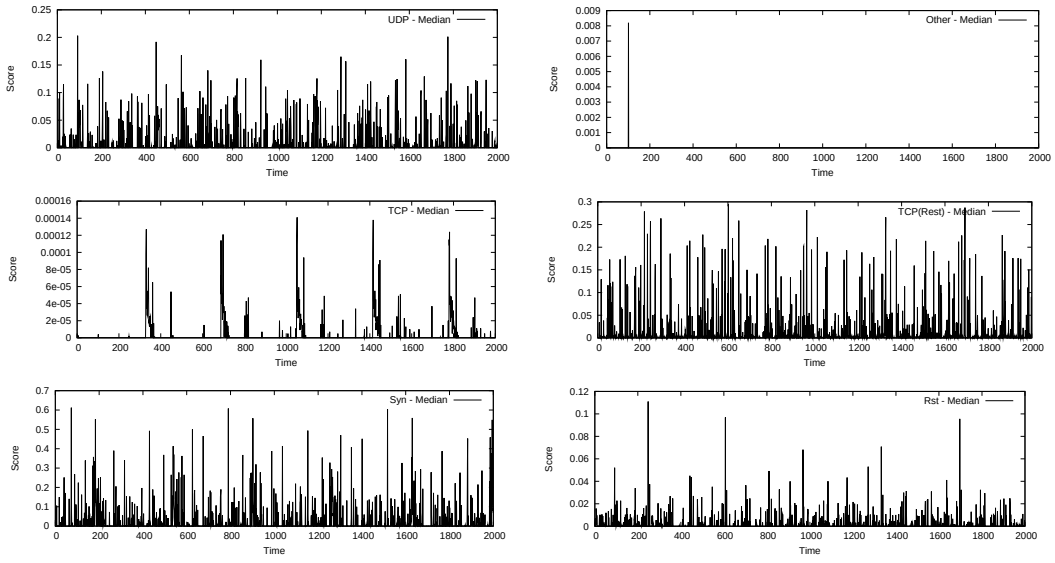

Figure 6.3. Trace 1: Median of the Algorithm outputs at various Anomaly tree nodes.

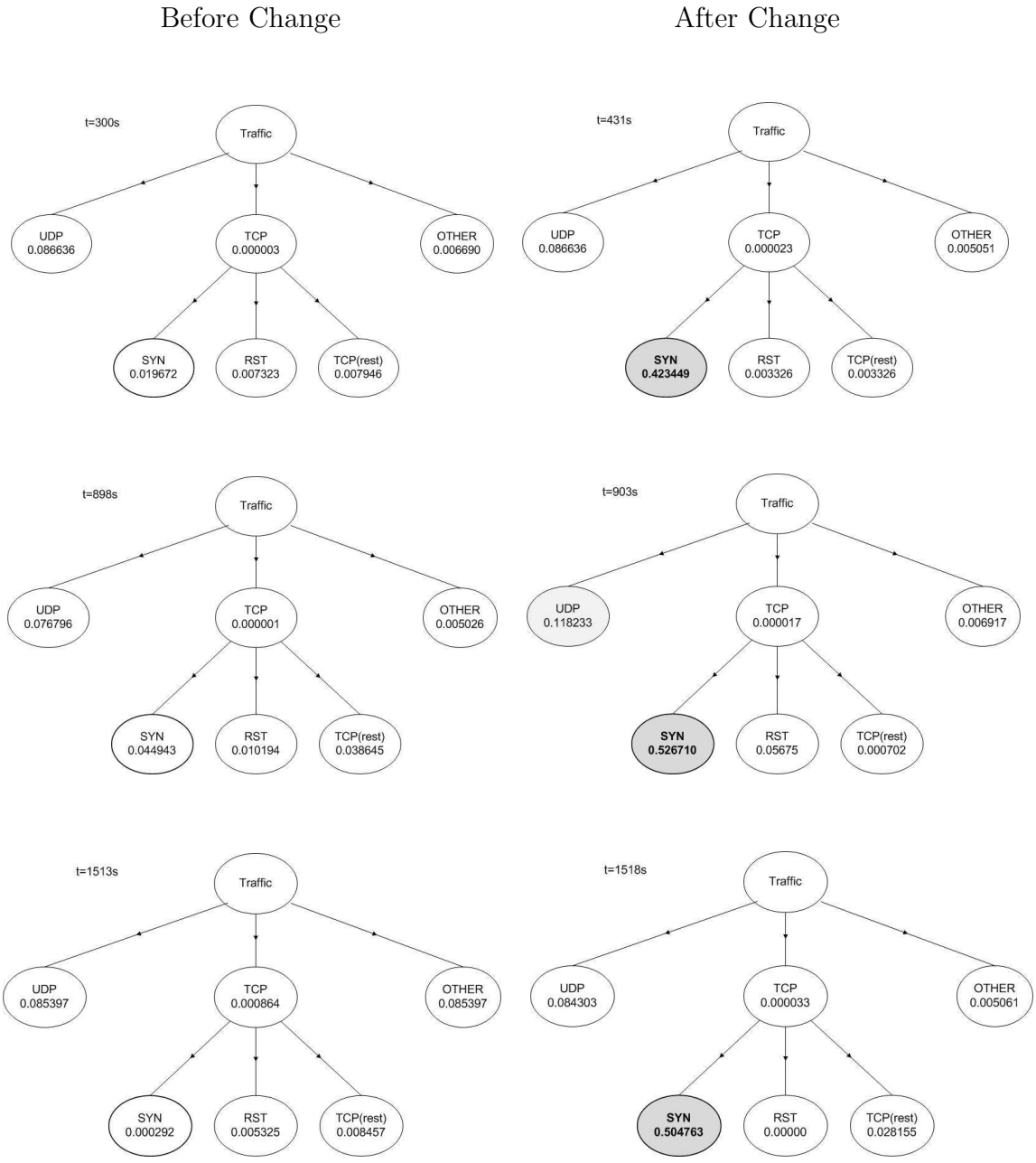

Figure 6.4. Trace 1: Anomaly Tree nodes at various instants of time

6.2 Trace 2

Trace 2 is actually a series of Code Red worm traces scripted to replay one after the other using tcpreplay. The total duration of the traces is 816.64 seconds including the background trace which does not contain the worm traffic.

6.2.1 Average of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the average of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.6. The CodeRed trace shows a large change in TCP, TCP-Syn and TCP-Rst traffic as seen in the Fig. 6.6. This was also observed in the manual reading of the trace where at instances after 220s into the trace a large amount of TCP traffic is seen. The behavior of the various algorithms at the anomalous nodes as seen in the Average Anomaly Tree is shown in Fig. 6.5.

6.2.2 Median of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the median of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.7

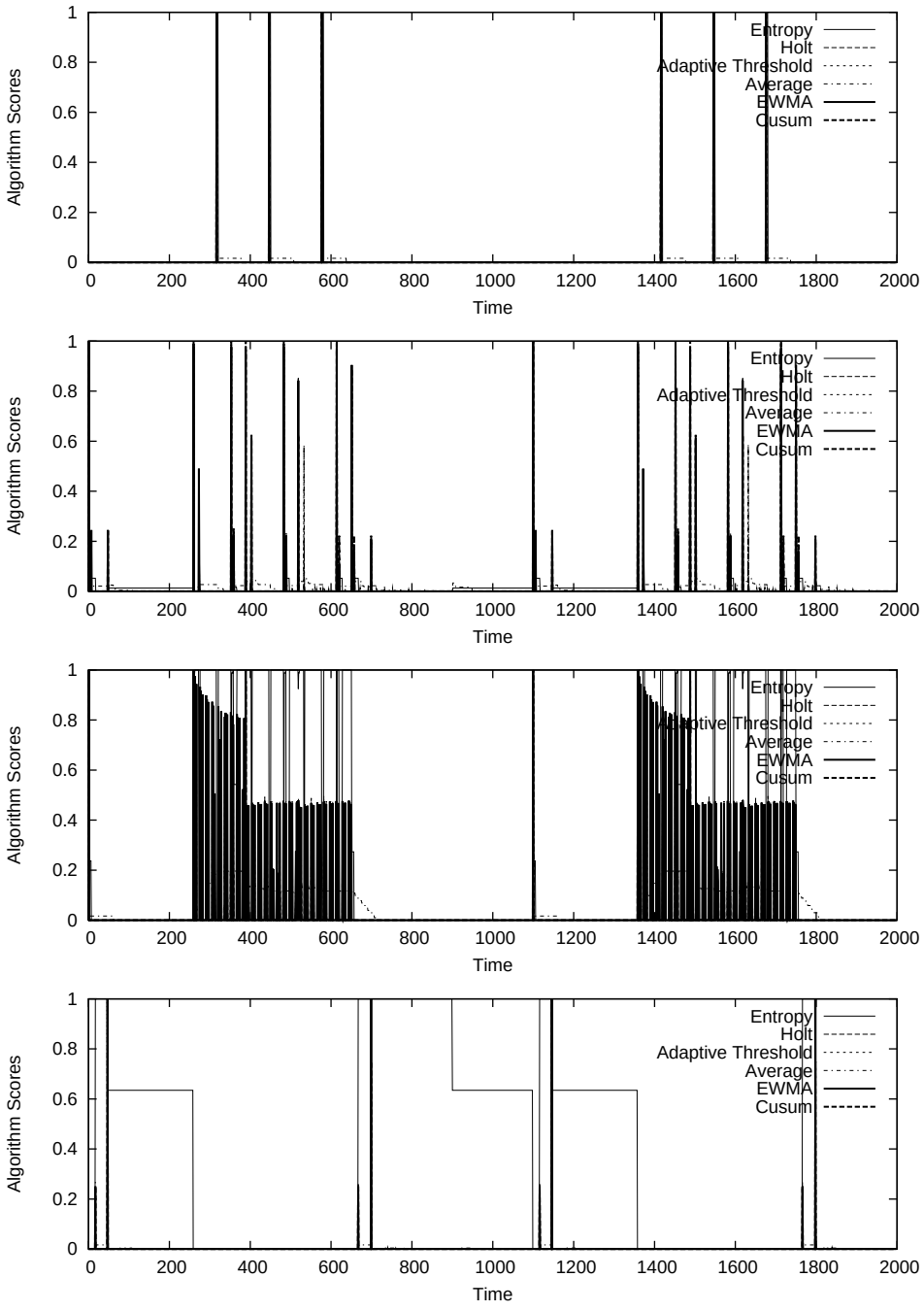

Figure 6.5. Trace 2: Output behavior of Algorithms as seen at the Syn, Rst, TCP and UDP nodes in the Anomaly Tree respectively.

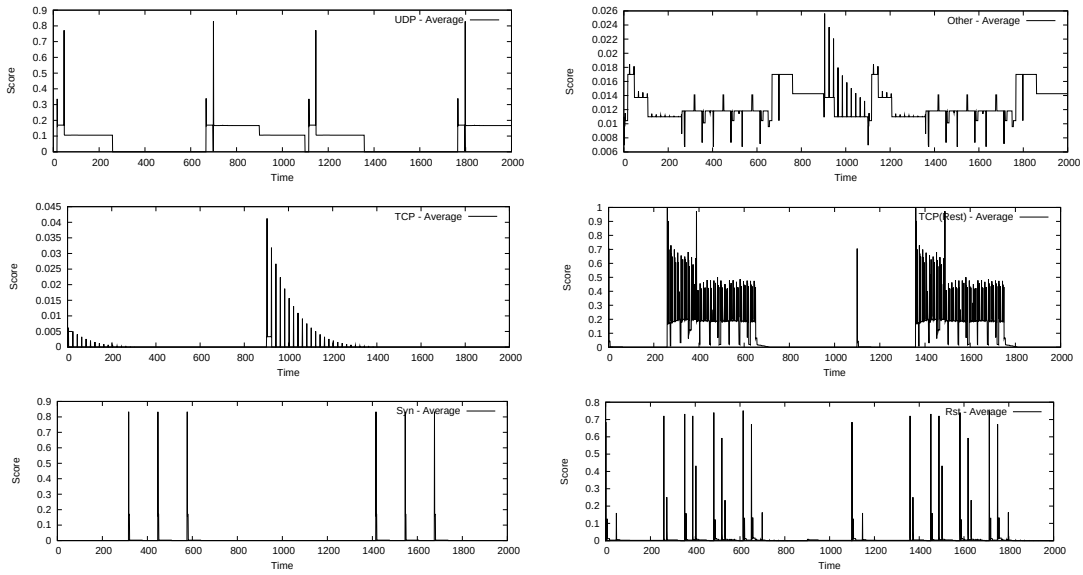

Figure 6.6. Trace 2: Average of the Algorithm outputs at various Anomaly tree nodes.

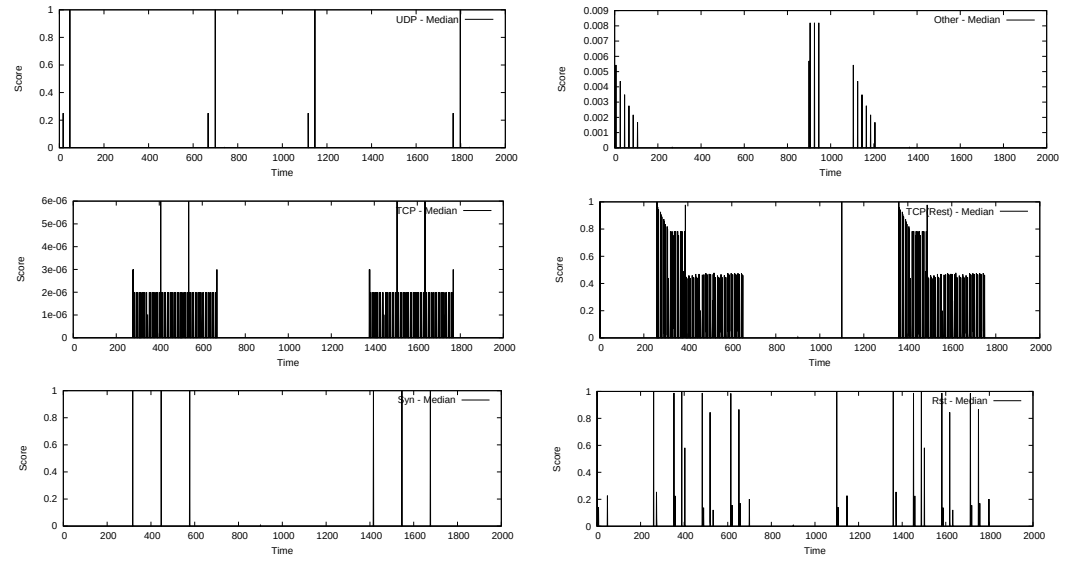

Figure 6.7. Trace 2 :Median of the Algorithm outputs at various Anomaly tree nodes.

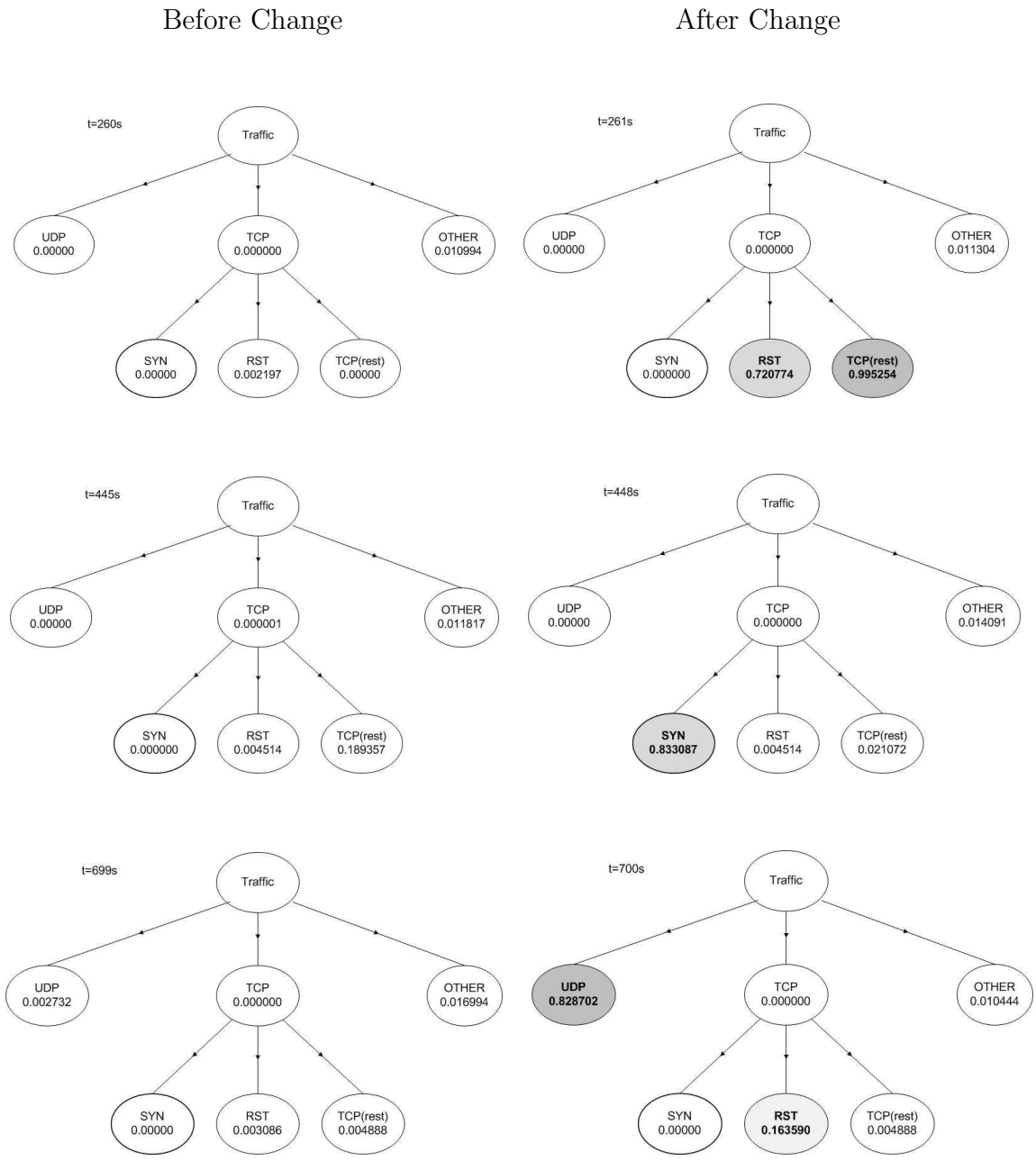

Figure 6.8. Trace 2: Anomaly Tree nodes at various instants of time

6.3 Trace 3

Trace 3 is a Distributed DoS attack trace from the DARPA 2000 Intrusion Detection evaluation carried out at MIT Lincoln labs. It includes a distributed denial of service attack run by an stealthy attacker. This attack scenario is carried out over multiple network and audit sessions. These sessions have been grouped into 5 attack phases, over the course of which the attacker probes the network, breaks in to a host by exploiting the Solaris sadmind vulnerability, installs trojan mstream DDoS software, and launches a DDoS attack at an off-site server from the compromised host.

6.3.1 Average of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the average of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.10. We observe large deviations in the Syn, TCP(rest), Rst and UDP plots. The algorithm behaviors for the affected nodes are shown in Figure. 6.9.

6.3.2 Median of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the median of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.11

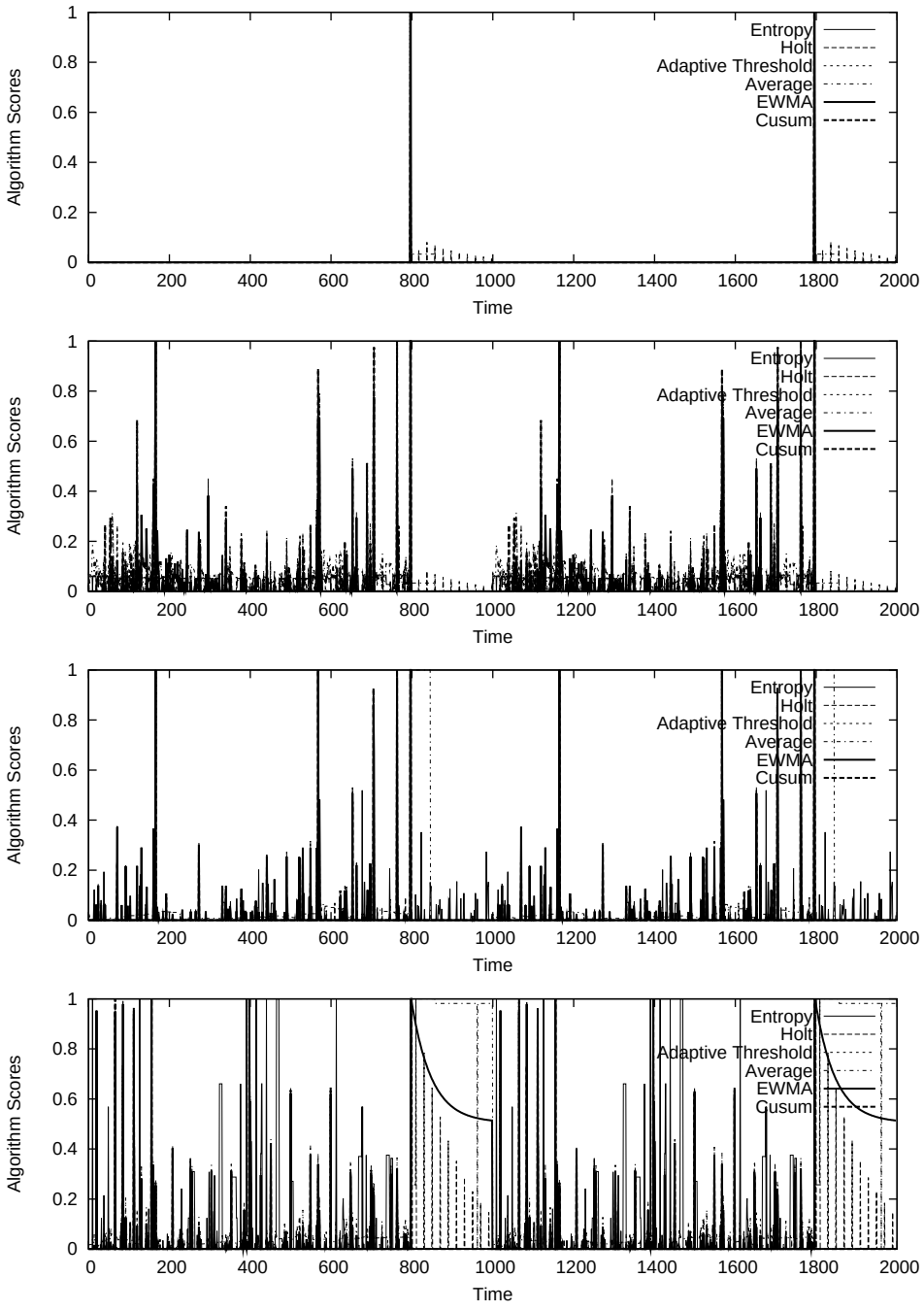

Figure 6.9. Trace 3: Output behavior of Algorithms as seen at the Syn, Rst, TCP and UDP nodes in the Anomaly Tree respectively.

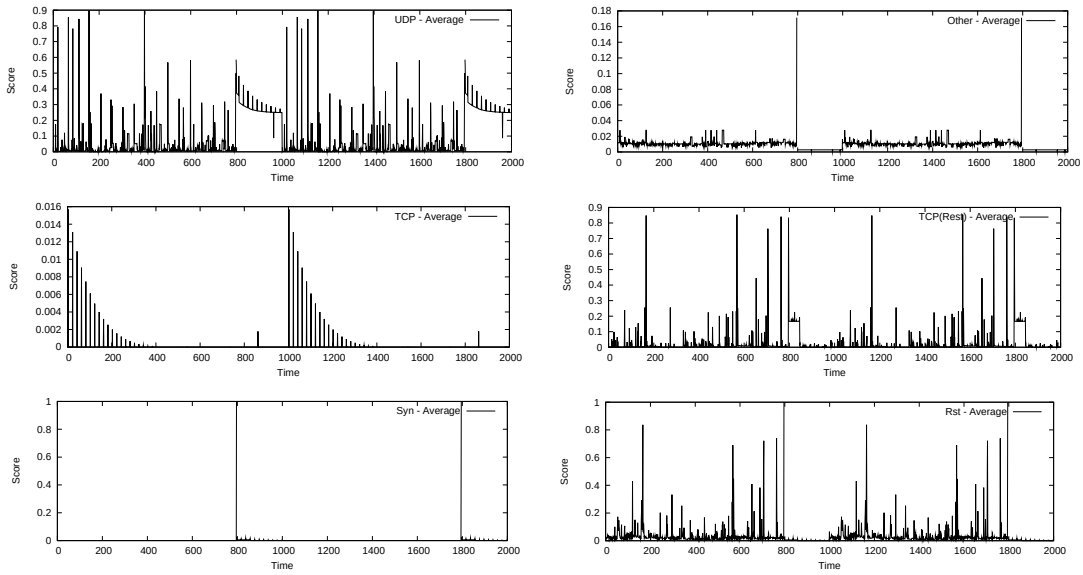

Figure 6.10. Trace 3: Average of the Algorithm outputs at various Anomaly tree nodes.

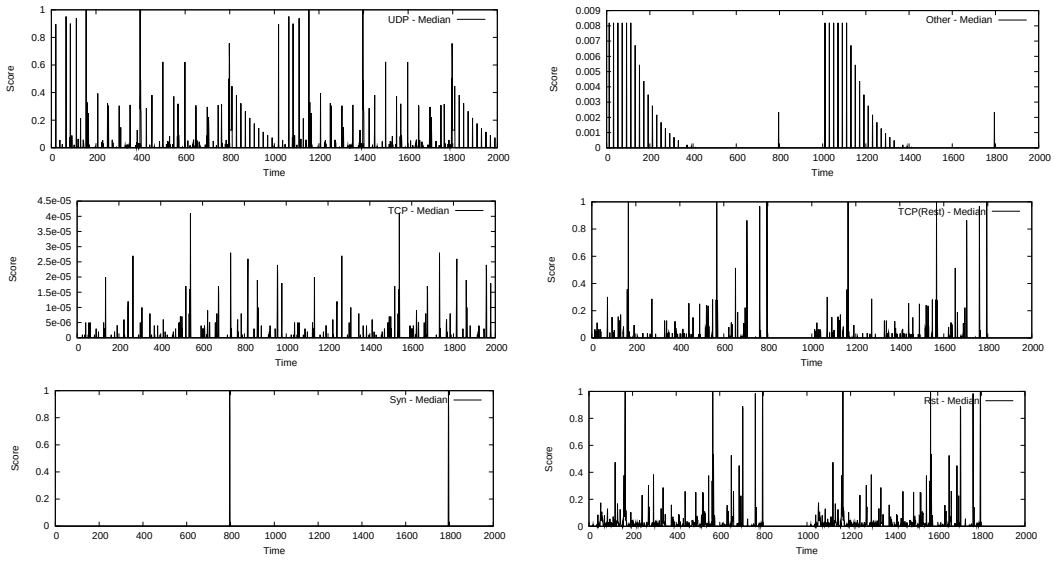

Figure 6.11. Trace 3: Median of the Algorithm outputs at various Anomaly tree nodes.

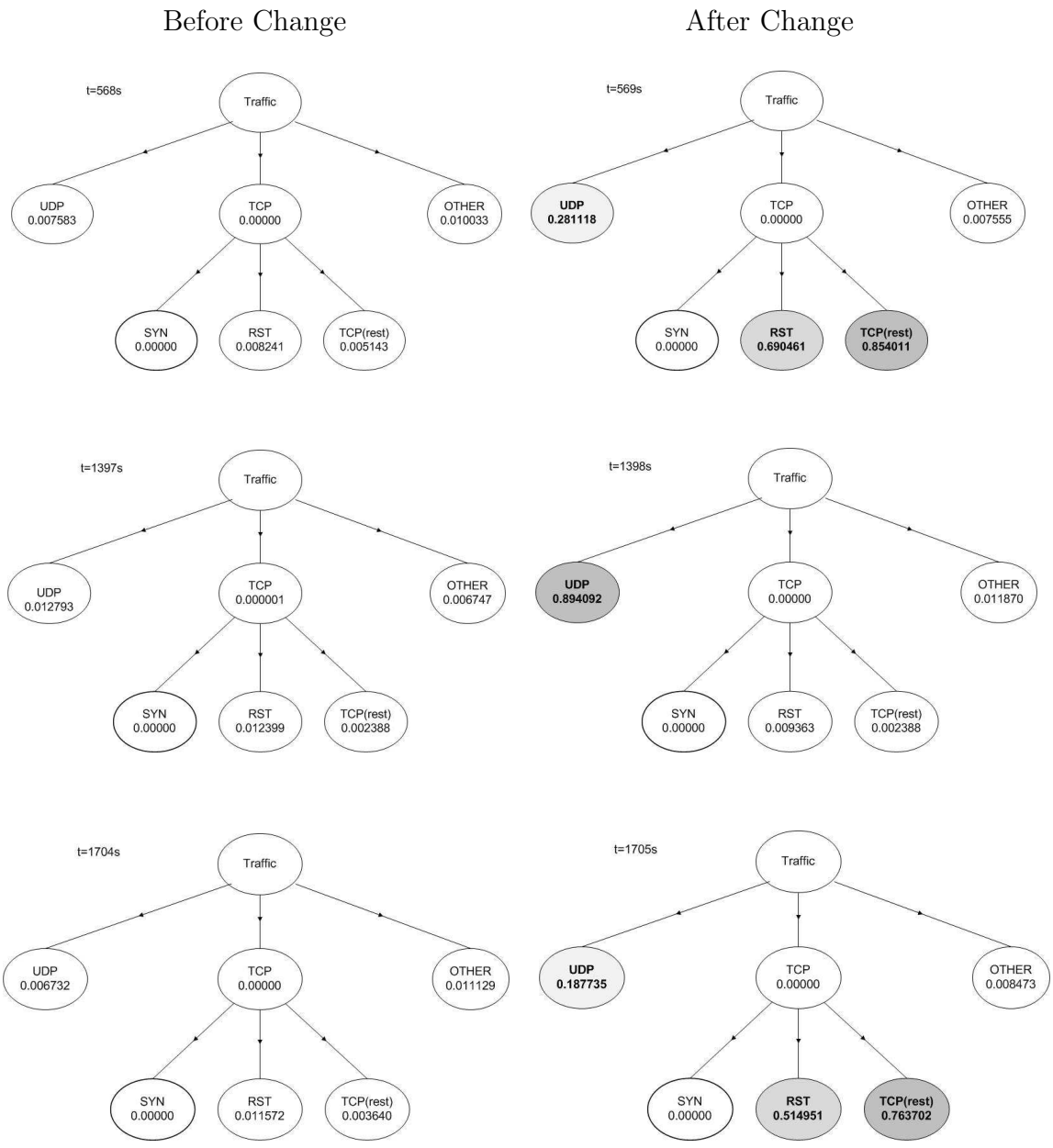

Figure 6.12. Trace 3: Anomaly Tree nodes at various instants of time

6.4 Trace 4

The trace was taken from the peering link at the Los Nettos network of the University of Southern California, LA, USA. The trace contains Syn portscan and portsweep traffic over a period of 900 seconds. A lot of zero-byte UDP packets are also observed in the trace. The trace was repetitively replayed a number of times using tcpreplay.

6.4.1 Average of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the average of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.14. The output behavior of the algorithms is shown in the Figures 6.13.

6.4.2 Median of the Normalized Algorithm outputs

The experiment was carried out with the nodes updating the median of the normalized algorithm outputs at every time instance. The graphs of the node values versus time are shown in the Figure. 6.15

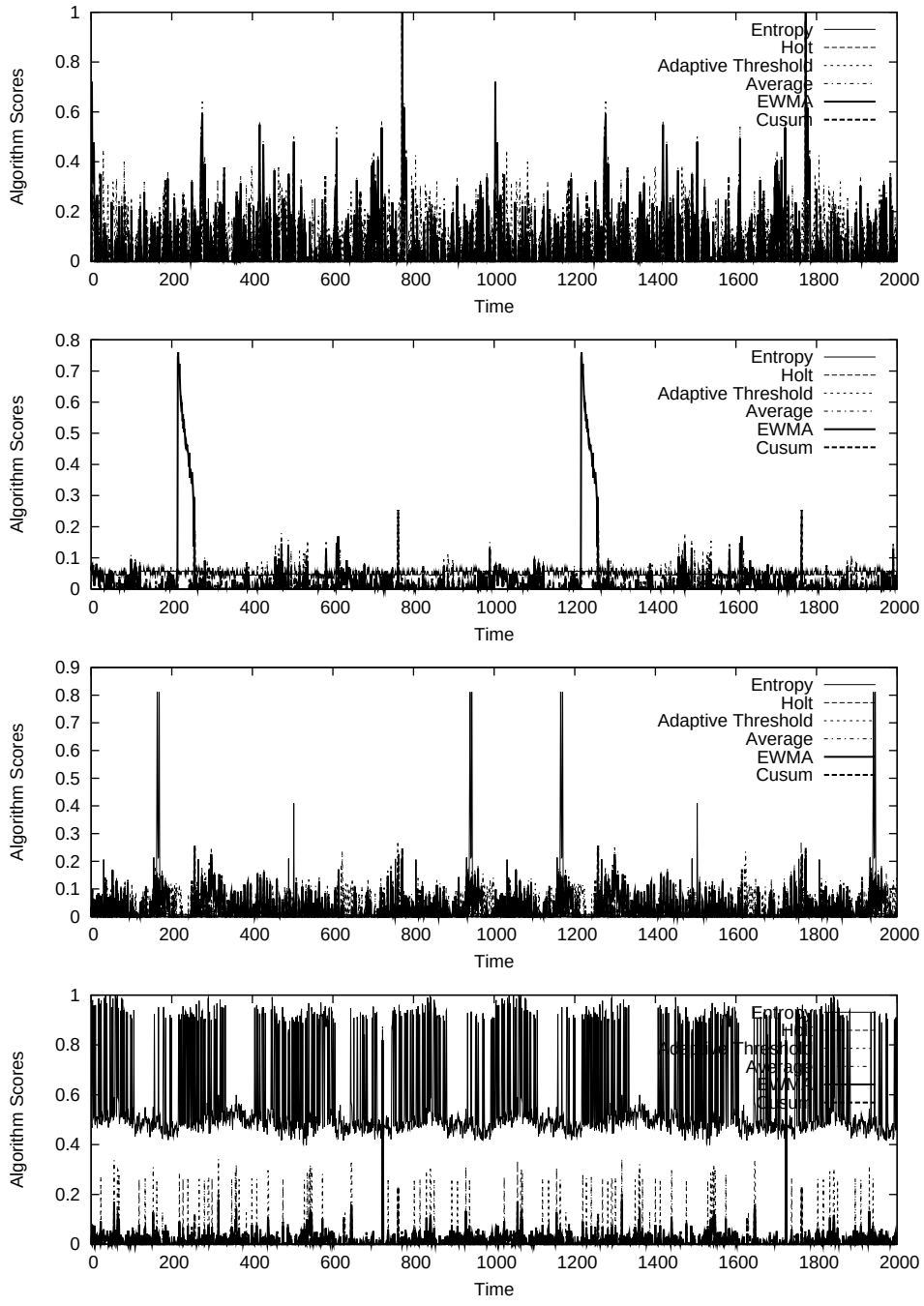

Figure 6.13. Trace 4: Output behavior of Algorithms as seen at the Syn, Rst, TCP and UDP nodes in the Anomaly Tree respectively.

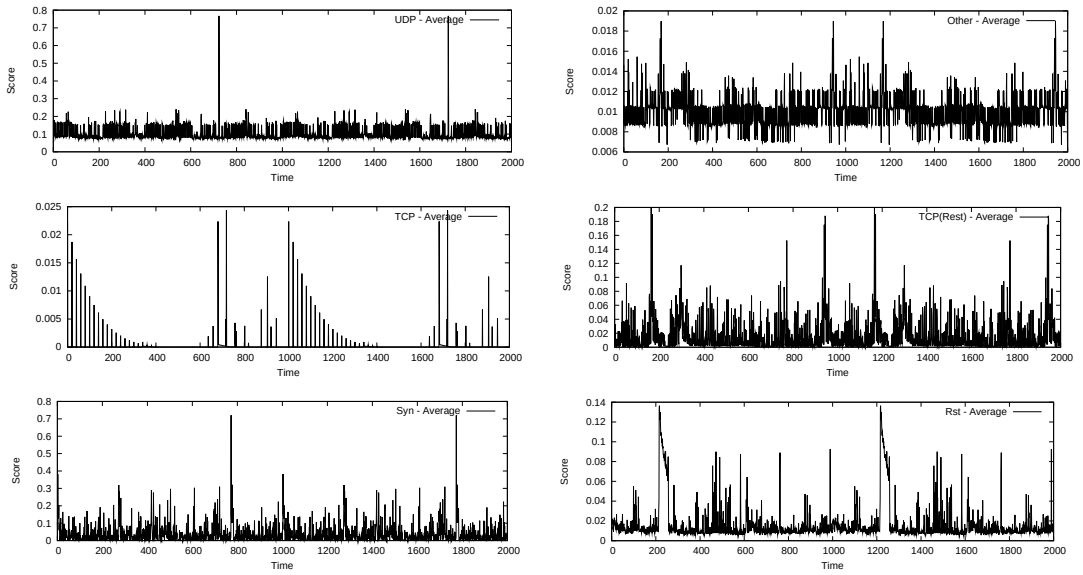

Figure 6.14. Trace 4: Average of the Algorithm outputs at various Anomaly tree nodes.

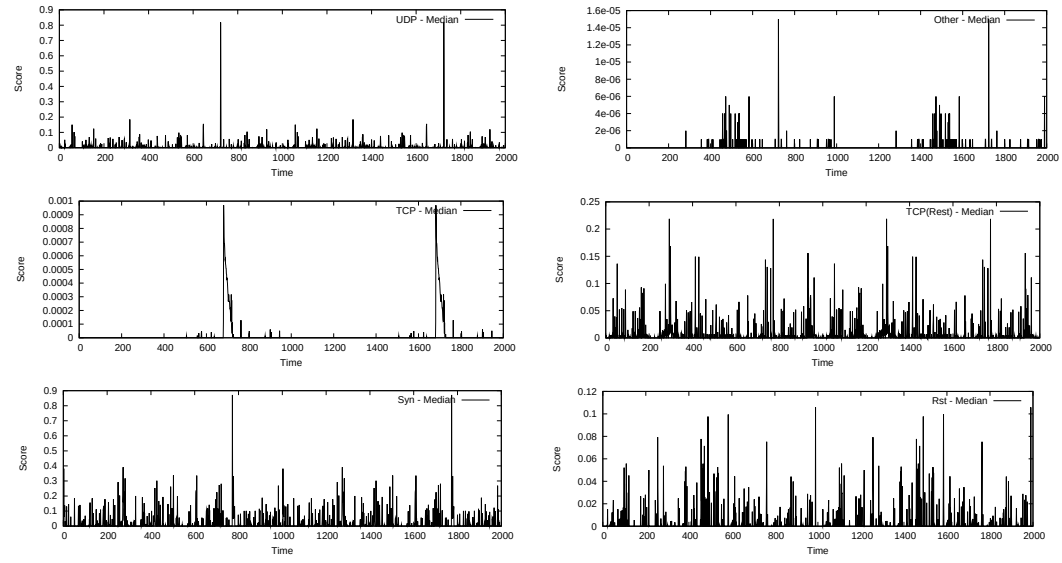

Figure 6.15. Trace 4: Median of the Algorithm outputs at various Anomaly tree nodes.

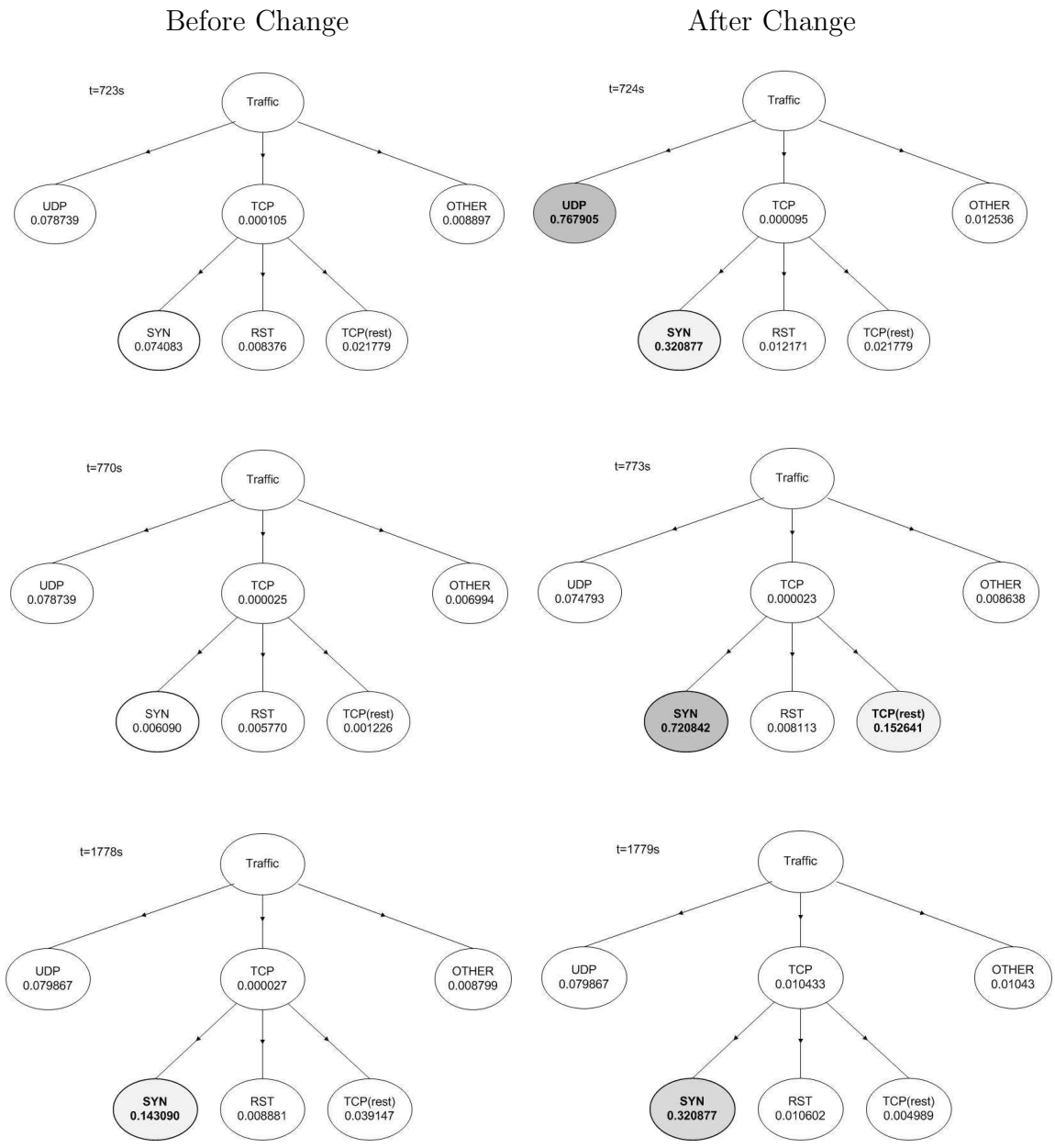

Figure 6.16. Trace 4: Anomaly Tree nodes at various instants of time

6.5 Anomaly Tree - Relationship between Anomalous and Non-Anomalous Nodes

The Anomaly tree represents a hierarchical relationship between the various subsets of traffic as observed in the network. At the highest level is the traffic as seen in the network. A level below are the TCP, UDP and Other (Non-TCP and Non-UDP) traffic. TCP is further divided into TCP-Syn, TCP-Rst and TCP(Rest)[1]. UDP, TCP-Syn, TCP-Rst and TCP(Rest) are further divided into 506 classes each according to the Entropy algorithm [20] explained earlier.

6.5.0.1 Detecting Anomalies

This hierarchical structure enables us to pinpoint the exact type of traffic causing the anomaly. At every time instance, the Anomaly tree updates the scores of the nodes as output by the alogithms. These scores are nothing but the Average *or* the Median of the normalized deviations as output by the algorithms. Normalization is used so as to enable us to compare and operate on the outputs of the algorithms on a similar timescale For example, in the case where there is a TCP-Syn portscan on port 80 occurring in the network, the Anomaly tree will register a high value at the TCP-Syn node. Further, Class 128 under Syn will also register a high value. However, one thing to note is that the parent node of TCP-Syn, TCP, might or might not register a high value depending on whether the total volume of traffic as seen among all its children increases or not. Thus, if there is an increase in the number of Syn packets but there is a proportionate decrease in the number of Rst packets, the TCP parent node will not register an anomaly. Thus, it becomes important to consider the parent-child relationship between nodes when characterizing the anomaly.

The Anomaly tree is also successful in registering multiple anomalies occurring in the system .For example, in Figure 6.16 , for the instances of 793s and 794s, the

[1]TCP(Rest) is the Non-Syn and Non-Rst TCP traffic

Anomaly tree is able to successfully register a TCP-Syn Portscan on Class 128 [2] and a UDP Flood attack occurring on various ports [3].

[2]Port 80 [20]

[3]The TCP-Syn and UDP children are not shown because of the large number of children, 506 in each case

CHAPTER 7

CONCLUSIONS

!

In this work, I have implemented six Anomaly Detection algorithms in parallerl on a Network Processor based passive measurement node. The algorithms are:

1. Holt Winter based Forecasting model proposed by Brutlag J.D. [4] which captures the history of the network traffic variations and predicts the future traffic rate in the form of a confidence band.

2. Behavior-based anomaly detection method proposed by Yu Gu [20] et al. that detects network anomalies by comparing the current network traffic against a baseline distribution.

3. Adaptive Threshold Algorithm [17], a straightforward and simple algorithm that detects anomalies based on violations of a threshold that is adaptively set based on recent traffic measurements.

4. Cumulative Sum Algorithm, [11], a proven statistical algorithm that maintains the cumulative sum of the deviations from a reference value, which is usually the mean of the time process.

5. Averaging Algorithm [16], another straightforward and simple algorithm that mainly averages the past 60 values. This is taken to be the predicted value which is compared to the present observed value.

6. Exponentially Weighted Moving Average, which applies weighting factors that decrease exponentially. Thus the most recent observations are given a greater weightage than the older observations.

The outputs of these algorithms were normalized and used to derive a common score of the state of the traffic each of those algorithms were monitoring. The state was constantly updated using an Anomaly tree, which maintains and updates the scores at various nodes which represent differenct subsets of traffic. Experiments were carried out on four different publicly available traces containing attack as well as background traffic. The Anomaly trees successfully registered an anomaly for the differnet cases at particular time instances. This was verified by physically inspecting the trace.

CHAPTER 8

FUTURE WORK

8.1 Collaborative Decisions and Feedback

The objective of parallel implementation is to successfully characterize anomalies and possibly reduce number of false positives that Anomaly Detection Systems are so susceptible to. We have six algorithms looking at different parts of the traffic. A more sophisticated method would be to design a correlation engine that takes into account the characteristics of not only the traffic, but also of each of these algorithms. The correlation engine can be designed to take into account the Total volume of traffic that is causing the anomaly, the Specific type of traffic, and the Intensity of the attack. Thus, we can use three new parameters:

1. *Intensity*: Defined by the number of packets involved in the attack during the monitoring cycle. E.g. Increasing number of Syn packets as found in Syn flooding attacks.

2. *Volume*: The parameter that is concerned with the total traffic viz. rate, total number of packets etcetera.

3. *Specificity*: Defined by the target of attack viz. port, protocol used, type of packets etcetera

Once we have the necessary information on the offending traffic, it can be selectively blocked and/or traces can be collected for later inspection, without affecting the performance as perceived by legitimate users. This constitutes the feedback to the

Anomaly Detection System, wherein the system is made autonomous by including a feedback to the passive measurement/capture node, thus enabling it to make intelligent decisions about whether to block the offending traffic and/or capture traces for further inspection.

Another problem is of a distributed measurement system which incorporate independent anomaly detection systems. These systems can have an ability to collaborate amongst each other to make intelligent decisions dynamically about the network traffic policies. Besides, the different systems can be divided in an independent manner thus enabling us to selectively block zombies (infected machines as in DoS attacks) which are a part of that particular subnet being monitored.

BIBLIOGRAPHY

[1] Snort: The open source network intrusion detection system.

[2] Change-point monitoring for the detection of dos attacks. *IEEE Trans. Dependable Secur. Comput. 1*, 4 (2004), 193–208. Member-Haining Wang and Member-Danlu Zhang and Fellow-Kang G. Shin.

[3] Barford, P, Kline J Plonka D, and A, Ron. A signal analysis of network traffc anomalies. In *In Proceedings of ACM SIGCOMM Internet Measurement Workshop* (Marseilles, France, Nov. 2002).

[4] Brutlag, J. D. Aberrant behavior detection in time series for network service monitoring. In *Proceeding of the 14th Systems Administration Conference* (2000), p. 139146.

[5] Bunga, Siddhartha. Online passive network measurement. Master's thesis, University of Massachusetts, Amherst, 2006.

[6] Cliff C. Zou, Member, IEEE Weibo Gong Fellow IEEE Don Towsley Fellow IEEE, and Lixin Gao, Member, IEEE. The monitoring and early detection of internet worms. *IEEE/ACM Transactions on Networking 13*, 5 (Oct. 2005).

[7] Corporation, Intel. Intel second generation network processor, 2005.

[8] Corporation, Radisys. Enp-2611 product data sheet, 2004.

[9] Deri, L., Suin, S., and Maselli, G. Design and implementation of an anomaly detection system: An empirical approach. In *In Proceedings of Terena TNC, 2003* (2003).

[10] Girardin, Luc. An eye on network intruder-administrator shootouts.

[11] Moore. A, Wagner. M, Ron M. Aryel. *Handbook of Biosurveillance.*

[12] Paxson, V.Bro. A system for detecting network intruders in real-time. In *In Proceedings of the 7th USENIX Security Symposium* (Jan. 1998).

[13] Ramaswamy, Ramaswamy. *An Embedded High Performance Network Measurement Architecture.* PhD thesis, University of Massachusetts, Amherst, 2006.

[14] Ramaswamy, Ramaswamy, Weng, Ning, and Wolf, Tilman. An IXA-based network measurement node. In *Proc. of Intel IXA University Summit* (Hudson, MA, Sept. 2004).

[15] Ramaswamy Ramaswamy, Weng Ning, and Wolf, Tilman. A network processor based passive measurement node. In *Passive and Active Measurement Workshop (PAM)* ((Boston, MA), Mar. 2005), pp. 337 – 340.

[16] Schwarzer, Christian. Prediction and adaptation in a traffc-aware packet filtering method. Master's thesis, Ecole Polytechnique Federale de Lausanne, 2006.

[17] Siris, V.A. Papagalou, F. Application of anomaly detection algorithms for detecting syn flooding attacks. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE* (2004), vol. 4, pp. 2050– 2054.

[18] Thottan, M, and Ji, C. Anomaly detection in ip networks. In *In IEEE Trans. Signal Processing* (Aug. 2003), pp. 2191 – 2204.

[19] Wang. H., Zhang.D., and Shin.K.G. Detecting syn flooding attacks. In *Proceedings of IEEE INFOCOM* (2002).

[20] Y.Gu, McCallum, A., and Towsley, D. Detecting anomalies in network traffic using maximum entropy, 2005.