Universiteit Leiden

Opleiding Informatica

DDoS Detection

using time-series analysis

Name:	Pavlos Platanias
Date:	August 26, 2016
1st supervisor:	Dr. Wojtek Kowalczyk
2nd supervisor:	Dr. Andre Deutz

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgments

I would like to thank my first thesis supervisor Dr. Wojtek Kowalczyk for his constant guidance during this undertaking. He made himself available for questions and assistance very frequently, and always provided good insight and perspective on my work. His patient and constructive style of feedback was invaluable in helping me complete the work.

My sincere thanks also goes to Dr. Andre Deutz for his willingness to co-supervise, advise and provide feedback on my work on short notice.

I also want to thank my initial internship supervisor Barry Weimes of MSS Department of Fox-IT for his enthusiastic encouragement and help with the starting phases of my work. He made sure to put my interests first and mediate between me and the company so that my work could proceed as planned.

Thanks are also due to my second internship supervisor Edwin van Vliet also of MSS Department of Fox-IT. His focused perspective on how to bring my thesis to completion was pivotal in helping me finishing my research.

Lastly, I would like to thank Christian Prickaerts of MSS Department of Fox-IT for being extremely helpful, offering advice and support during the course of my internship.

Abstract

As more and more people worldwide make use of services provided through the Internet, the damage caused by attacks aiming to disrupt these services keeps increasing. One such type of attack is the DoS (Denial of Service) attack, in which an attacker attempts to keep the victim's resources such as bandwidth, memory, CPU cycles or number of connections occupied and degrade the experience of legitimate users communicating with the victim. In this work, we develop a system to detect such attacks by creating forecast models of network traffic features using time-series algorithms, and detecting anomalies by tracking outlying values.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of traffic going through the Internet has been growing steadily over the last decade, as the capabilities of its infrastructure to generate and deliver large amounts of information expands. With the percentage of Internet users in the developed world reaching 78% [1], the amount of services provided through it has also grown. This growth however has made those services an attractive target for malicious attacks aiming to disrupt them in order to inflict financial damage, extortion, activism, or simply an outage of the service.

DDoS (Distributed Denial of Service) attacks in particular have been growing in popularity over the last years. This type of attack aims to disrupt an internet service by flooding it with traffic, taxing its capabilities to respond and preventing legitimate users from accessing it. Damage by such attacks [2] is caused by a pause in profit generating services, productivity loss, customer loss and disaster recovery costs. These damages are surveyed [3] to be on average over \$40.000 per hour, making an average DDoS that lasts 6-24 hours cost over \$500.000, without accounting for customer loss or secondary malware infections.

Detecting such attacks fast is critical in preventing them from doing extended damage to the targeted services. A DDoS attack will eventually make itself known by achieving its aim of taking a service down, but it is possible to detect and takes measures against it before this happens during the ramp up phase of the attack. It is also worth noting that a DDoS attack might not always aim to completely shut down a service but instead try to deteriorate it by occupying important system resources. These attacks won't announce themselves and if left undetected, can continue to cause damage for an indefinite amount of time. Also, DDoS attacks are often used as a diversion in order to either test a victim's defences, hide the delivery of other malicious attacks or conceal the theft of the victim's data. Such attacks have been surveyed to make up 26% of total DDoS attacks in 2016 [4].

Unfortunately, the amount of data that has to be monitored in order to detect a DDoS attack when it starts makes manual inspection an impossible task. Tools are necessary to help human analysts sift through the data, summarize trends and detect attacks automatically. This is not a trivial task, as neither attacks or normal network traffic always follow the same patterns. Traffic peaks, daily and weekly patterns, usage changes, all affect the volume and structure of traffic. In addition, attacks themselves do not remain static as new vulnerabilities and techniques are discovered to cause damage. An attack detection system should be able to differentiate between those legitimate changes and malicious attacks. Raw traffic data is unsuitable as input to such a system however. Data must be preprocessed, structured, metrics and features extracted from it, and the features must be suitable for classification of attacks. The amount of data also creates problems in storing historic data and processing new data fast enough for results to be relevant.

Our goal in this thesis is to create a platform for such a system. Our platform takes advantage of historic traffic data to create a seasonal model of network traffic and detects unusual patterns that don't conform to that model. We use network flow data collected by the Aguri traffic aggregator program [5] and construct relevant features such as bytes per second, traffic per protocol, IP address distribution entropy that can characterize and differentiate between normal and anomalous traffic. We then treat the values of those features as time-series and build forecast models for them with help of available algorithms, in order to identify DDoS attacks. As new traffic arrives, we use it as input to our pre-calculated weekly model and create new one-point-ahead forecasts, and identify anomalies according to how many standard deviations they fall away from the predicted mean, for every feature. We classify malicious traffic based on the anomaly vector across all features.

This thesis is organized as follows. We give an overview of DDoS attacks, their mechanics and characteristics in chapter 2. In chapter 3, we survey related work in the field of DDoS detection. In Section 4, we give an general overview of our approach followed by more details for each stage of detection. In chapter 5 we describe our datasets and experiments while chapter 6 contains our conclusion.

Chapter 2

DDoS

A DoS(Denial of Service) [6] attack is an attack that aims to stop or degrade any services that a victim provides over a network by consuming resources intended for legitimate users. It typically attacks network entities by saturating the link between the victim and the Internet with non-legitimate traffic, but it can also target different kinds of resources such as CPU time or amount of connections by overwhelming a particular network port or application service instead. A DDoS(Distributed Denial of Service) is a distributed, more frequently encountered version of the attack in which many attackers target the same victim simultaneously.

What makes these attacks particularly dangerous is the relative ease of execution, the difficulty of identifying the attacker and the difficulty of mitigating it. There are simple programs that can be used to execute a DoS attack against a target without special technical knowledge [7] and malicious entities offering such attacks as a service online for a typical cost of a few dollars [8]. The nature of the attack makes identifying the attacker a hard goal as well [9–11], since the IP addresses that malicious traffic originates from are either unwilling accomplices in the form of infected computers, or entirely fake. Lastly, mitigating a DoS attack is also a hard task, as it requires both the identification of malicious traffic and a scalable way of eliminating it. DDoS mitigation is typically done at the ISP (Internet Service Provider) or CDN (Content Delivery Network) level, since it requires resources and information the victim doesn't usually posses.

In the following sections we will give an overview of different types of DoS attacks and how they work.

2.1 Properties of DoS

2.1.1 Strategy

One of the main characteristics of a DoS attack is the strategy by which it intends to achieve its goals. We present some of the common strategies here, while a more thorough taxonomy can be found in [6].

Volumetric attacks are simple in that they rely on pure numbers to overwhelm the victim. They generate and send massive amounts of packets to the victim and using up the bandwidth of the network line between it and the Internet. The number of packets also stresses systems such as firewalls, end-routers, network cards and other parts of the victim infrastructure that need to process individual packets. Due to their simplicity and effectiveness, volumetric attacks are the most common form of DoS attack encountered.

Application-Layer attacks try to achieve the same result using less traffic. Instead of relying on brute force, they waste resources by exploiting weaknesses in protocols. An HTTP Flood attack for example sends a comparatively small amount of HTTP requests, but each request takes much more processing power to reply to than processing a network-layer packet of a volumetric attack. Such attacks that use particularly low volume of packets are often characterized as 'low-rate DoS'.

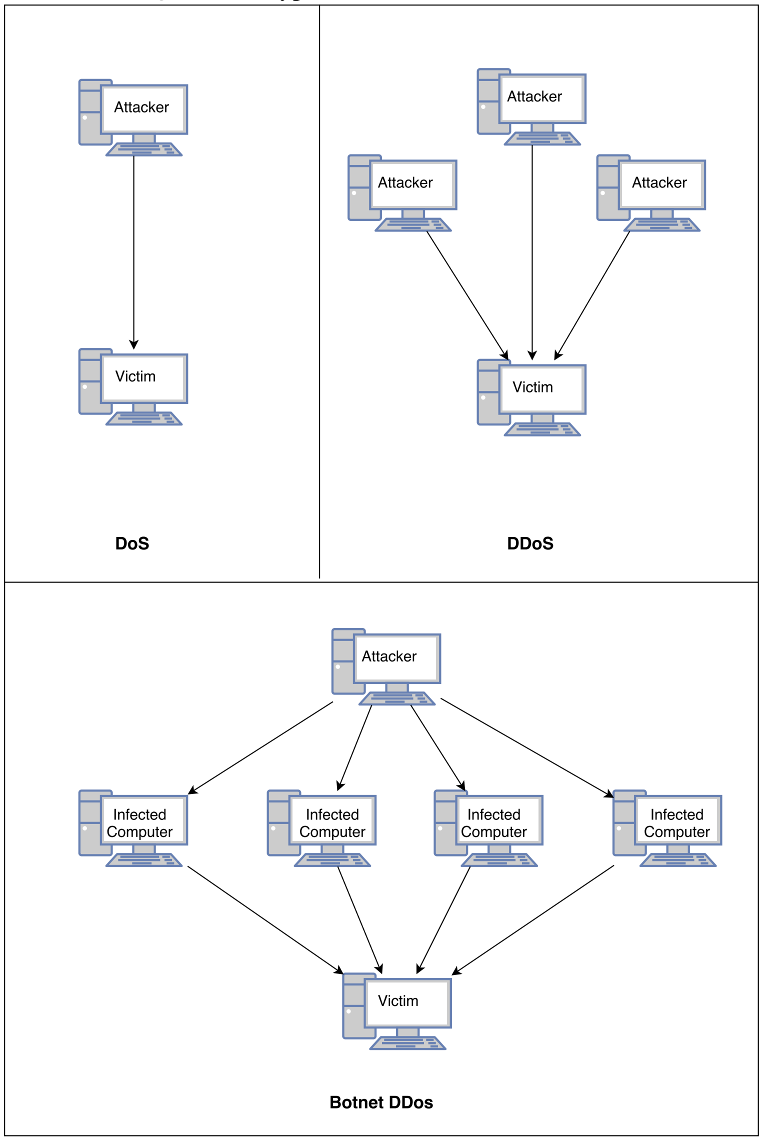
2.1.2 Distribution

A DDoS attack is a distributed version of a regular DoS attack. In the DoS variant, a single attacker uses a single device and Internet connection to transmit malicious traffic and the magnitude of the attack is limited by the attacker's bandwidth. In the DDoS variant, a single attacker employs many devices distributed across the Internet which amplifies the magnitude of the attack. Those devices usually belong to legitimate users but have been compromised by the attacker and are under his control. Collectively, these clusters of infected devices are called botnets.

2.1.3 Traceability

The traceability of an attack varies depending on its type. A DoS attack from a single attacker employing IP spoofing to hide his identity is very hard to trace at the victim level. Existing traceback methods require access to the network infrastructure between the victim and the attacker and are

Figure 2.1. Types of Denial of Service attacks



used at the ISP level. To describe the traceability of DDoS attacks, a distinction must be made between tracing the devices transmitting attacking traffic and tracing the perpetrator of the attack. Tracing the attacking devices works in the same way as with the DoS variant, but since the attacker does not own the attacking devices but is only controlling them through a malicious infection tracing his identity would require access to an infected machine. [12]

2.1.4 Reflectivity

An attack can be said to have reflective properties [13] if it employs legitimate third party devices to send attack traffic to the victim by sending the reflector device a request with the source set to the victim's IP address. Each reflector then replies back to the victim, potentially overwhelming it with responses to requests it didn't make. Note that reflector devices do not need to be compromised unlike the devices that make up DDoS's botnets, and can be used in conjunction with them to further amplify an attack.

2.1.5 Amplification

In reflective DoS attacks, the amplification ratio is defined as the ratio of the size of each packet the attacker sends for each packet the victim receives. A greater amplification ratio allows the attacker to use fewer resources of his own while taking up more of the victim's. Amplification is usually achieved by exploiting network protocols whose answers can be larger in size than their respective request. The amplification ratio of a DNS lookup request can be as big as 50x and 200-1000x for a CHARGEN request.

Chapter 3

Related work

There has been extensive research in using statistical classification, pattern recognition or learning techniques for the purpose of anomaly detection on network traffic data. In this section, we give a brief overview of the important literature in the subject.

In [14] Principal Component Analysis is used to separate network traffic into normal and anomalous components. Their method takes advantage of data from multiple network link by finding correlations in their respective data. In [15] sample entropy is used as a feature describing the distribution of unique source and destination IP addresses, and a clustering algorithm to detect anomalies such as DoS, DDoS and port-scan attacks. In [16] the benefits of entropy based features are further investigated, and it is shown that port and address distribution entropy features do not perform well in scan detection, but are limited to detecting volume anomalies such as DDoS attacks. In [17] network traffic is modelled as a time-series that displays LRD (Long-Range Dependence) characteristics and compare its autocorrelation and Fourier transform to detect DDoS flood attacks. [18] also rely on the LRD characteristic of network traffic and use a wavelet analysis to measure the dissimilarity between incoming traffic patterns to identify DDoS attacks. However, its sliding windows of 21 or 43 minutes for traffic pattern comparison make this method unsuitable for live detection. In [19] the DFT (Discrete Fourier Transform) of the number of incoming packets is found to differentiate between normal traffic and low-rate DDoS attacks with a response time fast enough to be used as live detection. In [20] the averaged Hurst parameter of normal traffic, measured as a series of the size of each incoming packet, is found to significantly differ from that of DDoS attack traffic. In [21], the joint entropy between the number of IP flows and the number of incoming packets

Table 3.1. Features and Methods used in DDoS Detection literature

Paper	Traffic data used to construct features	Methods used
[15]	source IP, destination IP, source Port, destination Port	Entropy, Subspace, Clustering
[16]	bytes in fixed interval	Wavelet Transform
[17]	bytes per packet	Autocorrelation, Fourier Transform
[14]	flow level information	Principal Component Analysis, Subspace
[19]	number of packets	Fourier Transform
[20]	bytes per packet	Hurst Parameter
[21]	number of packets, number of IP flows	Joint Entropy Analysis,
[24]	bytes per second, router CPU utilization, packet drop counts, packet header fields	Entropy, Time-series Forecasting, Rule Based Detection, Clustering
[22]	packets per flow	Entropy, Time-series Decomposition

for windows of incoming traffi is used to detect both high and low rate DDoS attacks. [22] define *flow connection entropy* as the entropy of the distribution of packet arrival probabilities and model it as a time-series. They then use a decomposition method to separate it into trend, seasonal and random components and detect anomalies such as DDoS attacks using a sliding window CUMSUM algorithm on each of those components.

There is also research in the more practical matter of how to handle alerts produced by anomaly detection systems described previously, or commercial ones already implemented. [23] take the approach of modelling the alerts themselves as a time-series and focusing on deviations from *normal* alert flows as important anomalies. [24] propose a tiered approach to network traffic anomaly detection and filtering. Their initial detection stage uses time-series forecasting to model normal traffic and catch outliers, then the focused detection stage relies a thresholding strategy for packet header information.

The problem of finding anomalies in time-series data is not constrained to network traffic. Algorithmic research has been done by [25] that introduces a new algorithm for finding most dissimilar subsections of time-series among time-series data. [26] use a kernel matrix alignment method on multivariate time series to detect and characterize pointwise and sub-series anomalies across all the data flows. [27] work on medical alert time-series data and create a model based on *ARIMA* (Auto Regressive Integrated Moving Average) forecasts to detect disease outbreaks.

Our own research uses many types of features of network traffic data discussed above, but our data is aggregated first and turned into summaries, missing some information in the process but preserving the overall structure. By combining entropy-based features for source and destination address distribution with volume-based features such as port and protocol traffic amount, we attempt to get a more general idea of the shape and structure of traffic and detect DDoS attacks that affect that structure. We use time-series forecasting techniques to create weekly forecasts of each of our features during normal traffic then track outliers that fall outside the expected values. By aggregating the outlier alerts across the forecast models of all features and over time, we attempt to detect DDoS attacks. While there exist many methods for modelling time series [28], in our paper we use only elementary methods.

Chapter 4

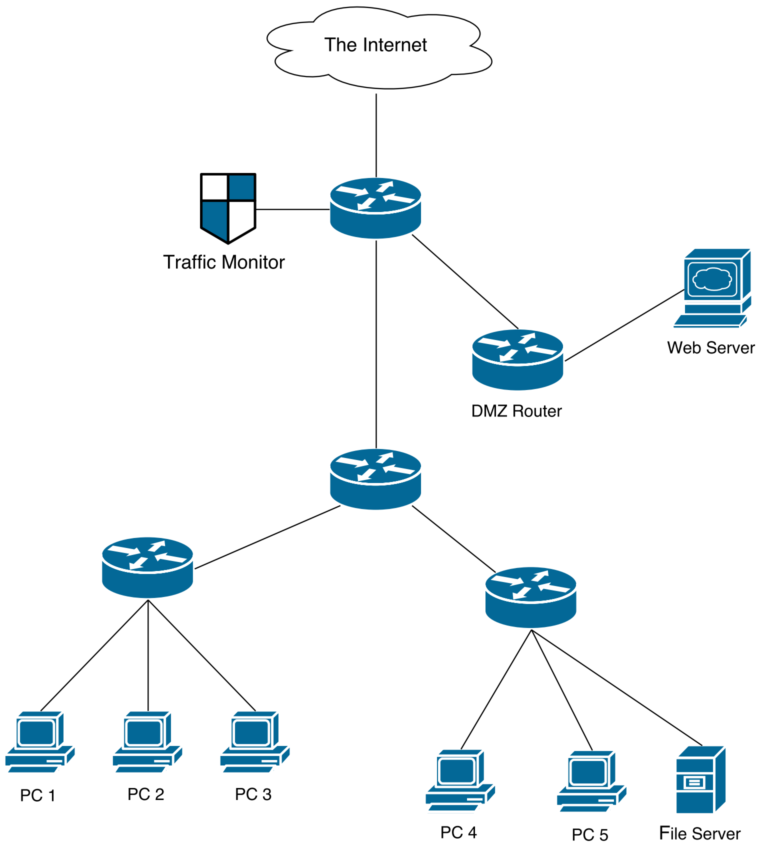
Our Approach

In this thesis, we develop a platform to detect DDoS attacks in network traffic. We briefly describe its workings in the following paragraphs, then elaborate on each subsystem in the following sections.

We assume traffic is monitored and logged at an appropriate location in the victim network. Selecting appropriate points is necessary in order for traffic to be descriptive. Traffic monitored on a subnet deep inside the victim's network will only describe the interaction of those particular hosts with themselves and outside hosts, but traffic monitored very close to the border of the victim network and the Internet will not be able to inspect encrypted VPN traffic. Choosing an appropriate location also depends on the structure of the network and the nature of the attack the system is attempting to detect. For DDoS attack detection, a listening point close to the connection of the victim network with the outside world, as seen in Figure Figure 4.1, is a good place to monitor traffic but individual network structure still needs to be taken into account. For this task of traffic monitoring we use the Aguri [5] software tool which captures incoming traffic and aggregates it into summaries, more details into which will be given in subsection 4.1.1. The hardware side of traffic capturing is outside the scope of this thesis.

Our platform initially attempts to create a seasonal time-series model of network traffic features such as bytes per second, sample entropy of IP addresses or specific protocol traffic. Distinguishing between normal or anomalous traffic variations is the key idea behind our system, and since normal traffic is affected by seasonal effects such as workday routine or weekends, those seasonal effects need to be captured in our forecast models. To create a model that can capture those seasonalities,

Figure 4.1. Traffic monitoring location in a typical network



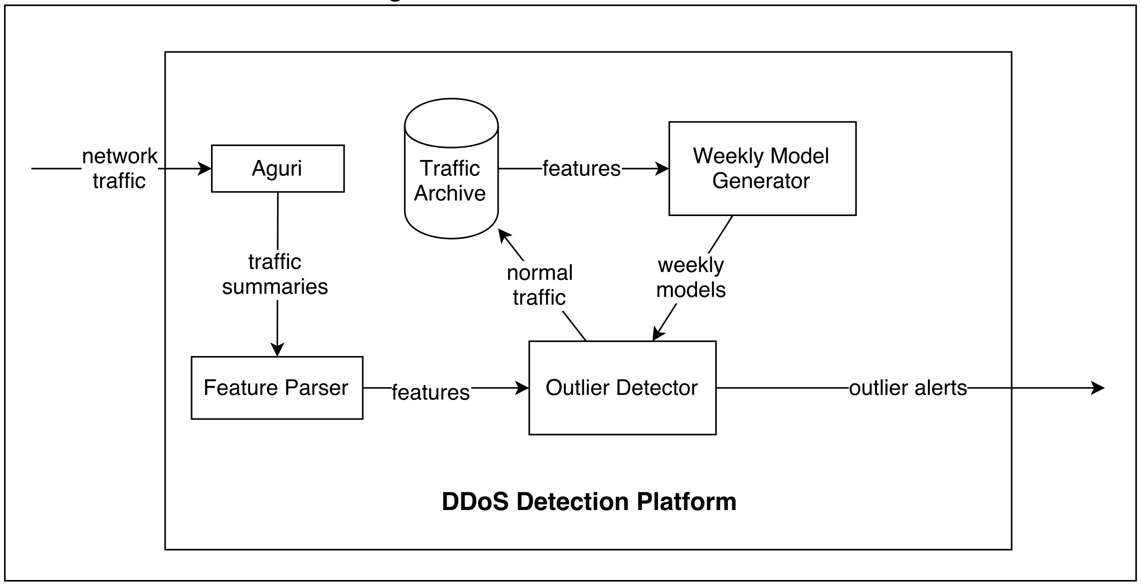
past data spanning multiple seasonal periods is required. In this case, our platform requires at least 2 to 4 weeks of Aguri traffic logs.

Once enough past data is acquired, a time-series model of traffic data can be created. We use time-series forecasting techniques explained in section 4.3 to create weekly models for all the traffic features we are monitoring. While the model creation is a fairly time-consuming process and the resulting forecast is many-steps-ahead, it only needs to be done weekly. Then, accuracy can be improved by passing new data to the existing models for one-step-ahead forecasting without the need to recalculate model parameters.

As new traffic is captured, its features are extracted and compared against the model forecast for that timeslot. Outliers falling outside the prediction intervals of the forecast are flagged and only traffic falling inside the expected values is used as input to the models for one-step-ahead prediction. Instead of the outlier values, we use pre-calculated weekly forecast values as input. This makes the model more accurately follow incoming traffic while avoiding having the model calibrate itself to expect outliers.

Outliers that span different features across multiple time periods are classified by the platform as anomalies.

Figure 4.2. DDoS Detection Platform



4.1 Data Preprocessing

The first consideration in attempting to use network traffic data to detect DDoS is selecting which parts of the data to actually use. Raw traffic data is comprised of a series of packets made up of protocol headers and payload, both of which are dynamic and change depending on the type of packet. While the headers of each protocol have consistent structure and information stored in set fields, the payload can be anything from a bitstream, a url or a compressed image file. While useful information for detecting attacks can be found in payloads, creating an algorithm that can combine payload and header analysis is beyond the scope of this thesis. Information contained in header fields such as TCP flags can be analysed much faster than payload, and given the primary characteristic of DDoS attacks to produce enormous amounts of traffic, any DDoS detection system has to be able to operate very fast on incoming packets otherwise the attack will incapacitate the detection system as well as the targeted services. For this reason, we focus only on a small number of packet attributes; its total length in bytes, protocol, source and destination IP addresses, source and destination ports.

Table 4.1. Typical IP Flow fields

Protocol	Source IP	Destination IP	Source Port	Destination Port	Size

This subset of traffic information seen in 4.1, aggregated over time, is usually referred to as a network or traffic "flow" [29] and has been used extensively as input in network anomaly detection schemes.

A second consideration is data aggregation. Network traffic produces an enormous amount of data that needs to be handled before humans or algorithms can use it. While the network flow format aggregates packets with matching fields and similar arrival times into the same flow, flow data still has a significant size. In this thesis, we use the Aguri traffic aggregator program to aggregate flows even further, into bigger flows from potentially many sources to many destinations; flows from IP's close to each other get aggregated into subnet flows, while flows of similar protocols get aggregated into larger protocol-group flows.

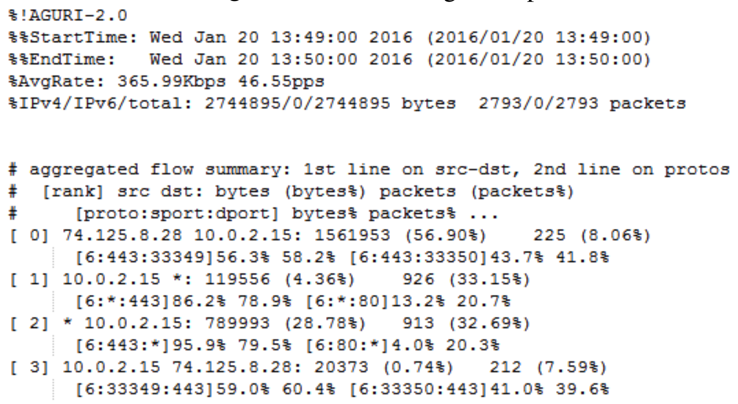
4.1.1 Aguri

Aguri [5] is an flow-based network traffic profiler software that aims to create concise summaries of network traffic. It reads packets and stores them into entries of 5-tuple format consisting of Source IP, Destination IP, Source port, Destination port, protocol and size in bytes. If a packet shares all its fields with an existing flow entry and it was transmitted before a certain timeout threshold, it is added to the flow and the packet/byte counts are incremented. Aguri aggregates flow entries according to each of the four identifier fields into larger flows until the collective aggregate traffic for a flow crosses a certain threshold, producing four separate traffic profiles.

4.1.1.1 Output types

The default output of Aguri consists of IP source — destination flow entries along with absolute and percentage amounts of bytes and packets corresponding to that flow. Each such flow is then broken down into IP protocol and Port source — destination subflows along with the percentage of the parent flow they make up. Flows that do not go over a traffic threshold (by default 1% of bytes or packets) get aggregated into bigger flows.

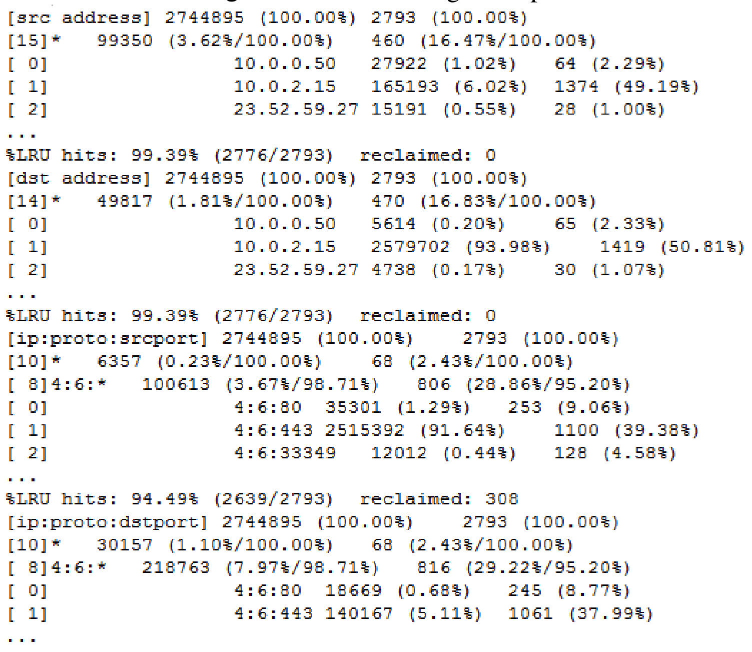
Figure 4.3. Default Aguri output



```
%!AGURI-2.0
%%StartTime: Wed Jan 20 13:49:00 2016 (2016/01/20 13:49:00)
%%EndTime:   Wed Jan 20 13:50:00 2016 (2016/01/20 13:50:00)
%AvgRate: 365.99Kbps 46.55pps
%IPv4/IPv6/total: 2744895/0/2744895 bytes  2793/0/2793 packets


# aggregated flow summary: 1st line on src-dst, 2nd line on protos
#  [rank] src dst: bytes (bytes%) packets (packets%)
#     [proto:sport:dport] bytes% packets% ...
[ 0] 74.125.8.28 10.0.2.15: 1561953 (56.90%)    225 (8.06%)
      [6:443:33349]56.3% 58.2% [6:443:33350]43.7% 41.8%
[ 1] 10.0.2.15 *: 119556 (4.36%)    926 (33.15%)
      [6:*:443]86.2% 78.9% [6:*:80]13.2% 20.7%
[ 2] * 10.0.2.15: 789993 (28.78%)   913 (32.69%)
      [6:443:*]95.9% 79.5% [6:80:*]4.0% 20.3%
[ 3] 10.0.2.15 74.125.8.28: 20373 (0.74%)   212 (7.59%)
      [6:33349:443]59.0% 60.4% [6:33350:443]41.0% 39.6%
```

Figure 4.4. Verbose Aguri output



```
[src address] 2744895 (100.00%) 2793 (100.00%)
[15]*   99350 (3.62%/100.00%)   460 (16.47%/100.00%)
[ 0]                10.0.0.50   27922 (1.02%)   64 (2.29%)
[ 1]                10.0.2.15   165193 (6.02%)  1374 (49.19%)
[ 2]                23.52.59.27 15191 (0.55%)   28 (1.00%)
...
%LRU hits: 99.39% (2776/2793)  reclaimed: 0
[dst address] 2744895 (100.00%) 2793 (100.00%)
[14]*   49817 (1.81%/100.00%)   470 (16.83%/100.00%)
[ 0]                10.0.0.50   5614 (0.20%)    65 (2.33%)
[ 1]                10.0.2.15   2579702 (93.98%)    1419 (50.81%)
[ 2]                23.52.59.27 4738 (0.17%)    30 (1.07%)
...
%LRU hits: 99.39% (2776/2793)  reclaimed: 0
[ip:proto:srcport] 2744895 (100.00%)    2793 (100.00%)
[10]*   6357 (0.23%/100.00%)    68 (2.43%/100.00%)
[ 8]4:6:*   100613 (3.67%/98.71%)   806 (28.86%/95.20%)
[ 0]                4:6:80  35301 (1.29%)   253 (9.06%)
[ 1]                4:6:443 2515392 (91.64%)    1100 (39.38%)
[ 2]                4:6:33349   12012 (0.44%)   128 (4.58%)
...
%LRU hits: 94.49% (2639/2793)  reclaimed: 308
[ip:proto:dstport] 2744895 (100.00%)    2793 (100.00%)
[10]*   30157 (1.10%/100.00%)   68 (2.43%/100.00%)
[ 8]4:6:*   218763 (7.97%/98.71%)   816 (29.22%/95.20%)
[ 0]                4:6:80  18669 (0.68%)   245 (8.77%)
[ 1]                4:6:443 140167 (5.11%)  1061 (37.99%)
...
```

The verbose output lists, for each of the four fields of Source IP, destination IP, Source Port, Destination Port, all the unique entries above the aggregation threshold and the traffic in bytes and packets they correspond to. Aggregated IP's form subnets while aggregated ports form port subgroups.

4.1.1.2 Aggregation

Aguri stores the counts for IP addresses and ports in a Patricia tree [30], a full binary radix tree. Internal nodes consist of a prefix key, which is the common bits of its two children, and a count. To update an entry count, the algorithm looks up the entry in the tree by starting from the root node and checking the prefix. If the prefix matches, the next bit of the entry decides which branch to follow. If the prefix does not match, a new node is created and inserted into the tree. The key is assigned to the new node together with the count. A new parent branch point is also created with the matching prefix of the new and one of the existing leaf nodes.

There are two aggregation mechanisms acting on the the tree at the same time. A performance optimizing one and a summarizing one. The summarizing aggregation goes through each leaf node in the tree, checks its entry count and aggregates it into its parent, adding its count to that of its parent if the entry is below a certain threshold. The performance optimization method has the aim of making the algorithm scalable for larger amounts of traffic by limiting memory usage and tree traversal time. It forces the tree to have a pre-set, limited number of nodes and replaces nodes according to an LRU replacement policy. In a high traffic scenario, this can cause nodes to get aggregated into their parent nodes despite meeting the threshold criteria. There is however a second much larger threshold that protects nodes that meet it, with the intent to preserve some critical structure.

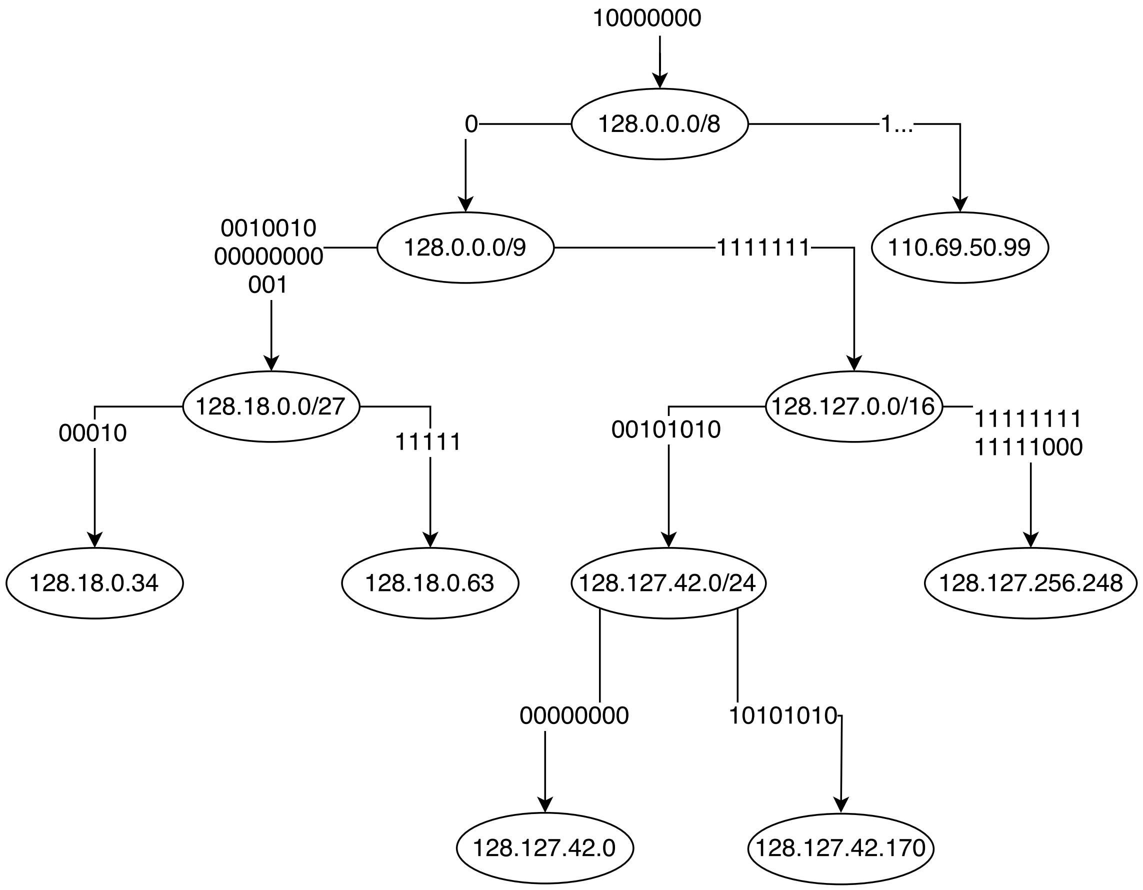
An issue with this algorithm is that as branch nodes with common prefixes are created as necessary, aggregation does not happen at uniform intervals. An IP address will get aggregated to the decision point above it, no matter how many bits it is away from that point. So, as seen in Figure 4.5, if IP address 110.69.50.99 were to fall under the traffic threshold and be aggregated, its traffic would go under subnet 128.0.0.0/8 directly, while unique IP address 128.127.256.248 would get aggregated to subnet 128.127.0.0/16.

4.2 Features

In this research, we select a mix of both volumetric and statistical features to describe network traffic. Features are extracted from aggregated summaries of traffic of a pre-set length. The length of these summaries in our research varies from 1 to 5 minutes. The features are the following:

- Bytes: Total number of bytes transferred during the summary time window.
- Packets: Total number of packets transferred during the summary time window.
- TCP Bytes: Percentage of TCP byte traffic to total byte traffic transmitted during the summary

Figure 4.5. Patricia Tree storing IP addresses



time window.

- TCP Packets: Percentage of TCP packet traffic to total packet traffic transmitted during the summary time window.

- UDP Bytes: Percentage of UDP byte traffic to total byte traffic transmitted during the summary time window.

- UDP Packets: Percentage of UDP packet traffic to total packet traffic transmitted during the summary time window.

- Ports: This is a collection of dynamically chosen features. We measure traffic originating or ending at well-known ports in the range of 0 to 1023, and create a feature for each such port crossing a certain threshold. We restrict ourselves to ports in the 0-1023 range because traffic in those ports is generated by specific applications and protocols, while ports outside that range are randomly assigned. Traffic on port 80 for example is generated by HTTP traffic. Since traffic is associated with two ports, the source and destination port, focusing on the 0-1023 range also prevents us from counting traffic twice as associated with both source and destination port. By choosing these ports dynamically, our system does not depend on manual identification of which ports had enough traffic to be important during each window. Heavy talkers are given the required attention, while spikes in ports we are not tracking will still be reflected as drops in ports we are tracking.

Percentage representations of port and protocol features were chosen in order to make our system less sensitive to uniform increases in traffic due to increased legitimate use of a network service. By using percentages, those features reflect only changes in the underlying structure of the traffic instead of its volume. We use a log it transform $logit(x) = \log(x/(1-x))$ to preprocess and change them from $[0-1]$ bounded to $[-\infty - +\infty]$ bounded for use in our forecasting algorithm. This transform used typically used in regression when the input is a percentage value [31] as the forecasting algorithms can give output over or under the $[0-1]$ bounds.

- Source and Destination IP Address Sample Entropy : Entropy was shown to be a descriptive feature for detecting attacks in network traffic in [15,16,21,22]. In the case of IP addresses, it gives information about the distribution of unique IP addresses producing or absorbing traffic. It is calculated as

$$H(X) = -\sum_{i=1}^{N} \left(\frac{n_i}{S}\right) \log_2 \left(\frac{n_i}{S}\right) \tag{4.1}$$

where S is the total number of unique IP addresses observed and n_i is the number of packets from each IP address i.

However, since our features are extracted from traffic summaries instead of complete traffic information, the calculated sample entropy is an approximation. For timeslots containing a number of unique IPs, the sample entropy is calculated as shown above. However when a timeslot contains a subnet, e.g. 192.168.1.0/24, we do not know how many actual unique IP addresses from that subnet contributed to traffic. The upper limit of unique

IP addresses is $2^{\text{address size}-\text{prefix size}}$ or 256 for the example, and the lower limit would be traffic of subnet/aggregation threshold. If the example subnet were responsible for 12% of traffic and the threshold for aggregation was 1%, there were a minimum of 12 unique IP addresses forming that subnet. If there were fewer, at least one of them would not have been aggregated.

We select the second strategy and use the least amount of unique IP addresses with traffic equally distributed among them when calculating entropy values in this thesis.

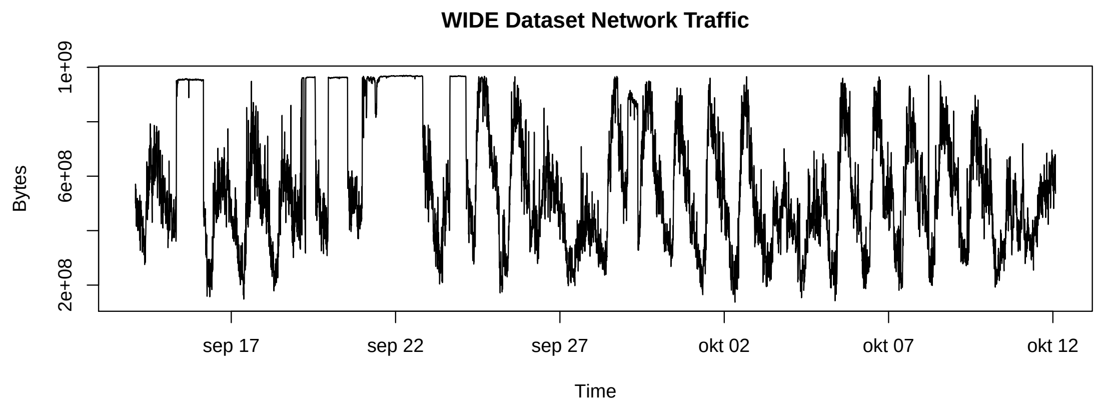
4.3 Time-series

Each feature, represented as a time-series, extracted from the Aguri traffic summaries is used by the DDoS Detection Framework as input to a univariate time-series forecast model, with the objective of creating an expected forecast of a week of future traffic. There are many considerations here both in regards to choosing the ideal time-series model as well as the granularity of the data. Most common time-series forecast models are essentially auto-regression models [28]; they forecast the next value of a variable of interest using a linear combination of past values of the same variable, making the assumption that past and future values of the variable are correlated. This assumption can empirically be observed to hold for normal network traffic in a sufficiently large network, an example of which is shown in Figure 4.6. While each individual network shows its own structural characteristics, daily usage patterns typically show an increase in traffic during work-hours, fall during nights and weekends, and periodic port and protocol usage patterns can also be observed. In the following section, we will look at a number of prevalent time-series models and their properties in the context of our research, and experiment with different configurations of those models to come up with one that is more meaningful for our purpose.

4.3.1 ARIMA

The Auto Regressive Integrated Moving Average(ARIMA) network of models is one of the most frequently used in time-series forecasting. An *ARIMA(p,d,q)* model, as the name suggests is a combination of autoregressive and moving average models with differencing included, described by the parameters p,d,q where:

Figure 4.6. Network Traffic from WIDE Dataset



- p = order of the autoregressive part.
- d = degree of first differencing.
- q = order of the moving average.

The autoregressive part of the model can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t \tag{4.2}$$

where y_t is the value of the variable at time t, c is a constant, e_t is white noise, $y_t - n$ are the lagged values of the variable y. The parameter p controls the number of past values of the variable taken into consideration.

The moving average part of the model can be written as

$$y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_n e_{t-q} \tag{4.3}$$

where c is a constant, e_t are the unobserved, past forecast error terms and q controls the number of past error terms to take into account.

Differencing is done in order to turn the time-series stationary, one whose properties do not depend on the time at which the series is observed. It is applied by replacing each value in a time-series with the difference of that value and the previous one.

$$y'_t = y_t - y_{t-1} \tag{4.4}$$

In an ARIMA model, the parameter d controls how many times this differencing is applied.

ARIMA models are useful for modelling forecasts for time-series that do not show any seasonal or trend patterns. Our traffic data however displays both multiple seasonal patterns and trend, making a simple ARIMA model unsuitable for modelling it.

4.3.2 Seasonal ARIMA

A seasonal ARIMA model can be used to model seasonal data by adding additional seasonal terms to the original model

$$\text{ARIMA}(p, d, q)(P, D, Q)_m \tag{4.5}$$

where P, D, Q are the autoregressive, differencing and moving average parameters of the seasonal part, and m is the number of periods per season, 12 for monthly data for example.

Seasonal ARIMA models can forecast a wider range of time-series data such as monthly, quarterly reports, daily usage patterns. However, they cannot accommodate multiple overlapping seasonal patterns in the data. Since our traffic data shows both daily and weekly seasonal patterns, we cannot use a seasonal ARIMA model to model it.

4.3.3 Exponential Smoothing

Exponential smoothing family models are forecasts that assign to future observations the weighted averages of past observations. Different models exist that accommodate simple time-series, series with trends, seasonal patterns or both. They can be classified [32] as shown in Tables 4.2, 4.3.

Simple exponential smoothing uses a smoothing parameter α to assign larger weights to newer observations making them more relevant to the future forecast. It can be written as:

$$y_{T+1} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + ... \tag{4.6}$$

Table 4.2. Taxonomy of ETS models

	Seasonal Component		
Trend Component	None	Additive	Multiplicative
None	(N,N)	(N,A)	(N,M)
Additive	(A,N)	(A,A)	(A,M)
Additive (damped)	(A_d,N)	(A_d,A)	(A_d,M)
Multiplicative	(M,N)	(M,A)	(M,M)
Multiplicative (damped)	(M_d,N)	(M_d,A)	(M_d,M)

Table 4.3. ETS models

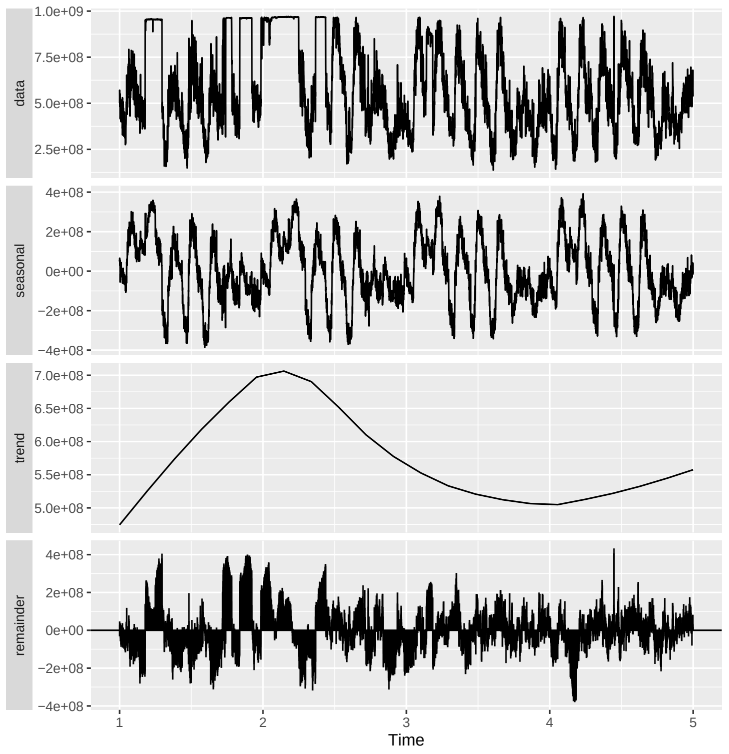
(N,N)	simple exponential smoothing
(A,N)	Holts linear method
(M,N)	Exponential trend method
(Ad,N)	additive damped trend method
(Md,N)	multiplicative damped trend method
(A,A)	additive Holt-Winters method
(A,M)	multiplicative Holt-Winters method
(Ad,M)	Holt-Winters damped method

Holt's linear trend method [33] adds a trend component, and the Holt-Winters method [34] adds a seasonal component as well.

4.3.4 STL Decomposition

STL Decomposition (Seasonal and Trend Decomposition using Loess) [35] is not a forecasting model, but can be used as an analysis tool or as a preprocessing technique. It splits a time-series into seasonal, trend and remainder components, uncovering seasonal patterns and trends from the data and showing to what extent they are responsible for the value of each data point. Then, the separate coefficients can be forecast individually, or the seasonal component can be assumed to remain stationary and only the remainder component be forecast.

Figure 4.7. STL Decomposition of 5 weeks of traffic from WIDE dataset



4.3.5 Prediction Intervals

When a forecast estimates future values of a random variable, it generates a range of possible values the variable can take with high probability, with the value in the middle of the range having the highest probability. That range of values is called the *prediction interval* of the forecast, and is associated with the probability it has of containing the actual future value. Typically, forecasts generate prediction intervals for 80% and 95% probabilities, but other values can be used.

Prediction intervals are generated using formula

$$y_i \pm k\sigma \tag{4.7}$$

where σ is the standard deviation of the residuals (the error of the forecast on the values used to generate it), and k is a multiplier that determines the percentage level of the prediction interval and represents the number of standard deviations away from the mean necessary to capture an area percentage of a normal curve equal to that prediction interval. For a 95% interval that number is 1.96, while other common values for the multiplier are given in Table 4.4.

Table 4.4. Common prediction interval values

Prediction Interval	Multiplier
70	1.04
80	1.28
95	1.96
99	2.58

However, prediction intervals widen the more a forecast extends into the future. Prediction intervals for a forecast 1 unit into the future are calculated as shown, but each subsequent future forecast that does not rely on new data inherits the errors of the previous forecast, widening the prediction intervals as well.

Moreover, in practice prediction intervals have been observed to be too narrow [36] because of the tendency of forecast models to ignore sources of uncertainty such as parameter estimates, choice of model and the future changes in the process that generated the historical data (DGP uncertainty).

Still, we use these values as they provide a useful metric of how uncertainty is associated with each forecasted value.

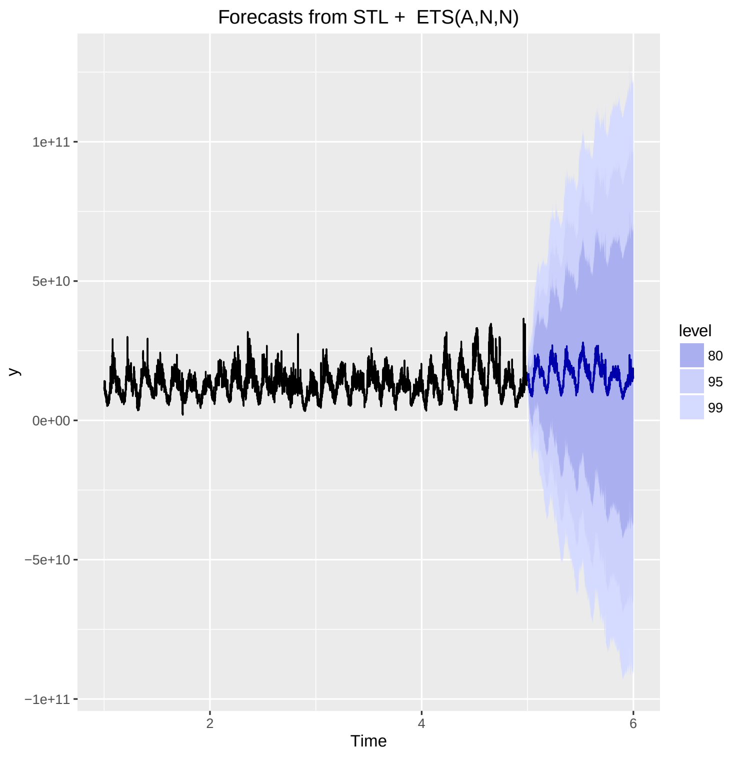
4.3.6 Our Approach

We tested the above forecast models and techniques on our data and arrived at the the conclusion that they were incapable of modelling both the daily and weekly seasonalities of our data and doing so fast enough to process series spanning many thousands of data points.

To cope with this issue, we chose a method that combines STL Decomposition with Exponential Smoothing forecasting. The data is decomposed into trend, seasonal and remainder components, an ETS model is then built for the remainder component and the forecast is re-seasonalized using the past data seasonal component. An example of the results produced by such forecast for one of our features on a part of our dataset can be seen in Figure 4.8 and Figure 4.9.

We create a weekly model of each traffic feature using this method. This model is considered a many-steps ahead model because its forecast horizon, the number of data-points it forecasts without access to new data, is one week. However, because the error associated with each predictions carries over to the next, the prediction intervals grow to an unreasonable width. This makes the forecast less useful for prediction because even if we get a good estimation for the mean value, wide prediction

Figure 4.8. Weekly forecast of bytes per second using 1 month of historic data.



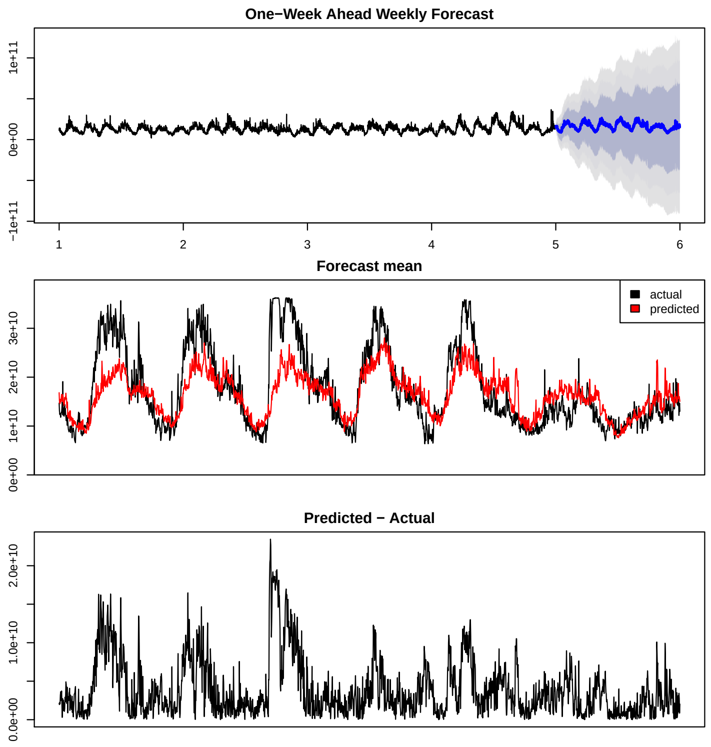
intervals mean we can expect big deviations from that mean.

To rectify this issue, we augment our approach with sequential one-step ahead forecasts for each feature. After generating the weekly models, we use new traffic feature values arriving during the forecasted week as input to the model and forecast each individual data-point. This prevents the prediction intervals from widening and increases the accuracy of the forecast. We also introduce a filter that makes new values ineligible as input to the model, if they fall outside the 99% prediction interval of the forecast for that value. This is done in order to prevent the forecast from being affected by outliers. The results of this technique can be seen in Figure 4.10.

4.4 Anomaly Attack Classification

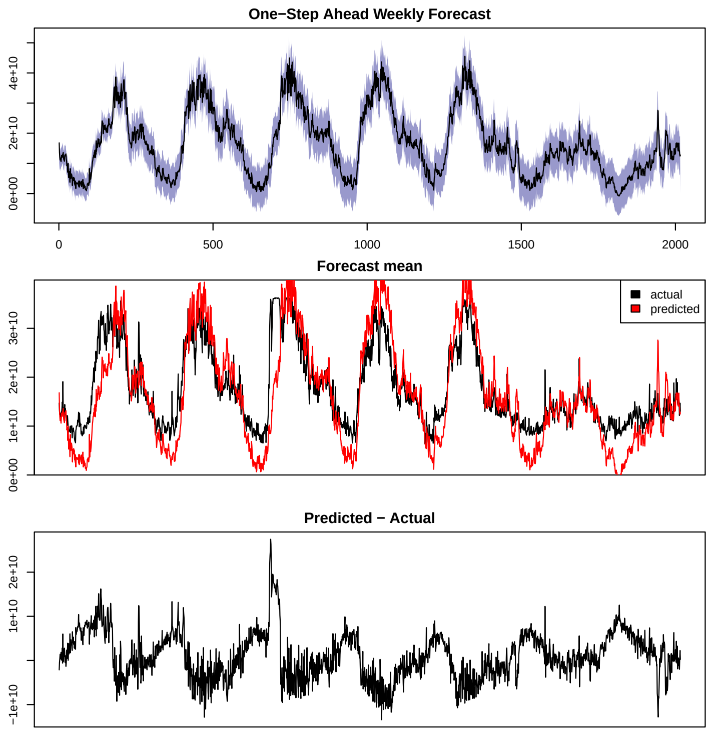
When incoming traffic features fall outside the expected ranges of values given by our forecast models, we flag an anomaly for that specific feature time-series and timeslot. These anomalies are found frequently even in normal traffic, so we only classify an attack if there are outliers across the majority of the feature time-series that persist for multiple timeslots. We use different thresholds

Figure 4.9. Accuracy of weekly forecast.



for the number of timeslots, number of features required for a series of anomalies to be recognized as an attack, and the number of standard deviations values need differ from forecasts in order to be characterized as anomalies. However, we are unable to test the accuracy of those thresholds due to our lack of labelled DDoS dataset of proper length to run detection on.

Figure 4.10. One-step ahead forecast using new values as input.



Chapter 5

Experiments

In this section, we perform experiments testing various aspects of our DDoS Detection Platform. We will first describe the datasets we use, then describe the experiments we undertook.

5.1 Datasets

A serious issue impeding research in network traffic anomaly detection is the lack of commonly adopted, good datasets to test algorithms on. The most common dataset in intrusion detection, KDD'99 [37], dates back to 1999 and the attacks it contains are less relevant today. It has also been criticized for having skewed distributions of attack and normal traffic [38] and proposed A corrected version, NSL-KDD [39] fixes some of these problems but still does not describe real world or modern attacks. This problem is made worse by the fact that traffic data often contains sensitive, private information, often belonging to 3rd parties. Commercial companies avoid releasing their data for research purposes for those reasons, and even research institutions need to first anonymize traffic prior to releasing it. Anonymization does not guarantee safety of sensitive data as deanonymization techniques can often reverse the process [40, 41], and there is no single commonly adopted anonymization scheme. Lastly, traffic datasets are rarely labelled. The amount of data involved makes a manual labelling by human analysts a very long process, and labels are not definitive in real world data, as an attack might escape notice or legitimate traffic might get misclassified. Artificial datasets and methods to generate them have been proposed [42, 43] that solve some of those problems at the cost of realism. In this thesis, we use a selection of both artificial and real datasets, labelled and unlabelled for different purposes. We test the accuracy of our forecast model on a real, unlabelled dataset since labels are not necessary to forecast mixed traffic while a labelled dataset is used to evaluate if our selected features and metric capture DDoS attack occurrences.

5.1.1 ICSX-UNB 2012

This dataset created by [42] consists of artificial network traffic created with the use of profiles containing descriptions of attacks and abstract distribution models of common applications or network protocols. Profiles were created by analysing real traffic and abstracting real user behaviour. Agents are then programmed to execute those profiles to create legitimate user activity. Attacks of various kinds were then executed in real time from physical devices. The benefits of this technique are that it allows complete capture of packets without any privacy issue, and that it offers complete labelling information for all traffic.

The traffic data spans a period of 7 days. There are 3 days containing only normal traffic and days containing the following attacks: Inside Infiltration, HTTP DoS, DDoS, Brute Force SSH. Of these, the DoS and DDOS attacks are of interest.

There were 12 timeslots of 1 minute each missing from the data. We interpolate their values using for each their two neighbouring timeslots.

5.1.2 GWA

The GWA dataset consists of anonymized traffic collected from Fox-IT's guest wireless network. It contains unlabelled, real world traffic for a period of 1 month, from February 29th to April 1st 2016 . As it is not known if there are any anomalies in the data, it was used to test the model forecasts of normal traffic.

Daylight Savings Time went into effect Sunday March 27, which made clocks turn forward 1 hour that day. Having a 23 hour day would skew our prediction model, so we correct this by repeating the previous hour a second time.

5.1.3 WIDE backbone

Collected by the MAWI Working Group of the WIDE project, this traffic data [44] is collected from the transit link of WIDE to their upstream ISP. It has been in operation since 2006. We collected a one year period of traffic data for the year 2015 in Aguri format.

There were 33 missing timeslots of 5 minutes each missing from the data. We interpolate their values using for each their two neighbouring timeslots.

Due to the way the dataset is being made available and its size, we were only able to get an Aguri parsed version of it, with 5 minute long timeslots and non-verbose output. As mentioned in subsubsection 4.1.1.1, the same features can be extracted from either output but the non-verbose data have more error than the verbose, when compared to unaggregated data.

5.2 Feature Visualization

Visualization is an important part of any detection system meant to help human analysts identify issues as well as automatically detect them itself. Human readable output of not just the results but inner parts of the process help detect errors in the system, calibrate it and discover new information that the system was not programmed to look for. For this reason, together with our models we produce visualizations showing both the evolution of raw features over time as well as their forecast models. Using these visualization we, as developers, were able to examine and evaluate aspects of our system and understand the responsiveness of our selected features. In the same way, users of our platform can use these visualizations in parallel with the detection features to get a better understanding of the underlying data and its structure.

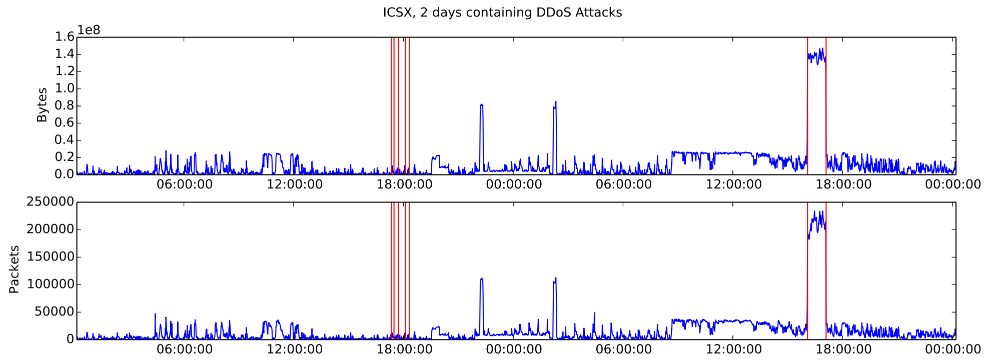
In this section, we will go through a number of visualizations of the evolution of various features on our datasets in order to showcase how information can be gained from it. We do not show a complete listing of visualization of all our features for all datasets, but focus instead on parts that contain interesting information.

5.2.1 Visualizations of ICSX dataset

First we examine packet and byte graphs for the ICSX dataset. We can see from Figure 5.1 that the DDoS attack on day 5 which was labelled on the dataset, denoted by vertical red lines, is clearly visible. However the smaller HTTP DoS attacks on day 4 do not cause any visible increase in traffic, and other traffic spikes are related to normal traffic instead of attacks.

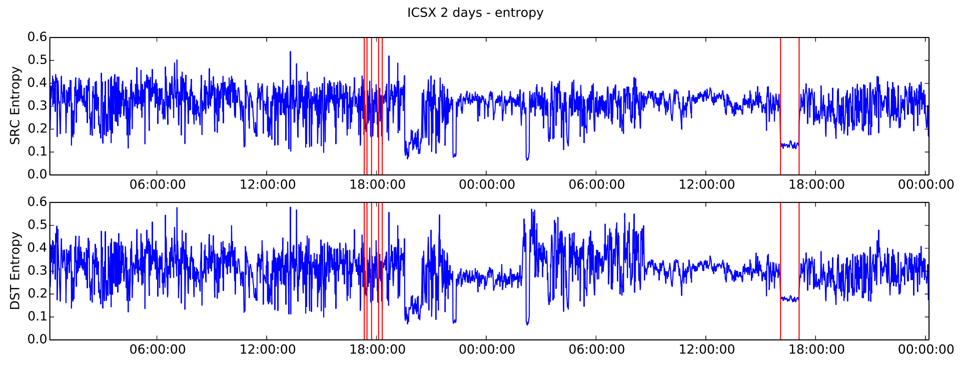
We then examine the source/destination IP address distribution entropy feature over time. We

Figure 5.1. ICSX - byte and packet traffic.



see from Figure 5.2 that the sample entropy measure is responsive to the DDoS attack, however its values during the attack are not larger in magnitude than during other times. What makes the attack noticeable is that the entropy remains visibly stable and unchanging during the attack, the distribution of source/destination IPs is much more static compared to normal periods. The HTTP DoS attacks however remain invisible.

Figure 5.2. ICSX - source, destination entropy



We also plot TCP and UDP protocols in Figure 5.3 as well as the highest traffic well-known ports Figure 5.4, found dynamically, but they do not show signs of the attacks.

5.2.2 Visualizations of GWA dataset

The GWA dataset does not contain any DDoS attacks to the knowledge of its providers, but it does contain some interesting information about protocol/port usage and daily/weekly seasonality. The strong weekly seasonality can be observed in Figure 5.5 while a less strong daily seasonality can be seen in Figure 5.6. We see that while daily traffic falls strongly inside a time window of

Figure 5.3. ICSX - TCP and UDP protocols

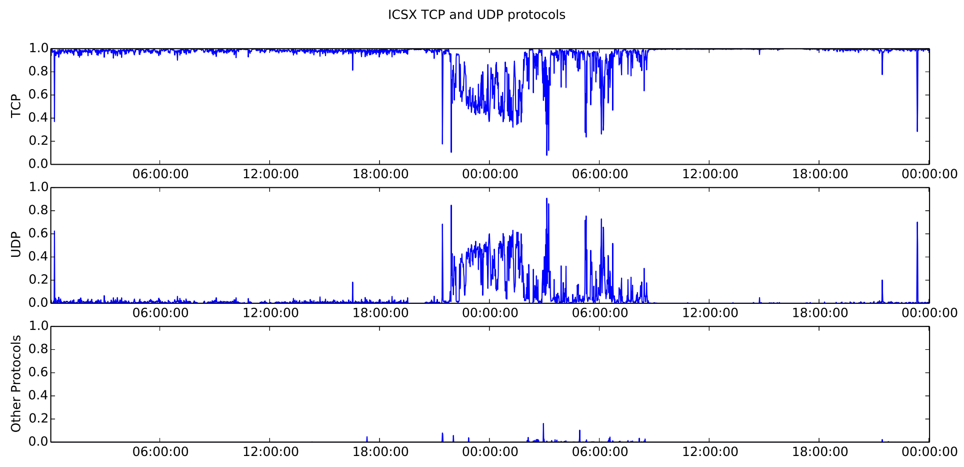
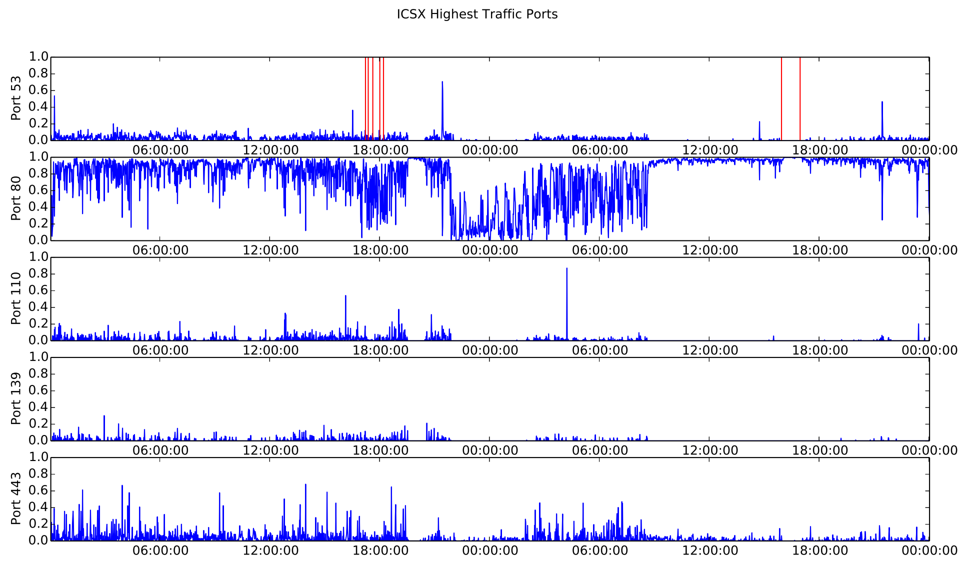


Figure 5.4. ICSX - Well known ports



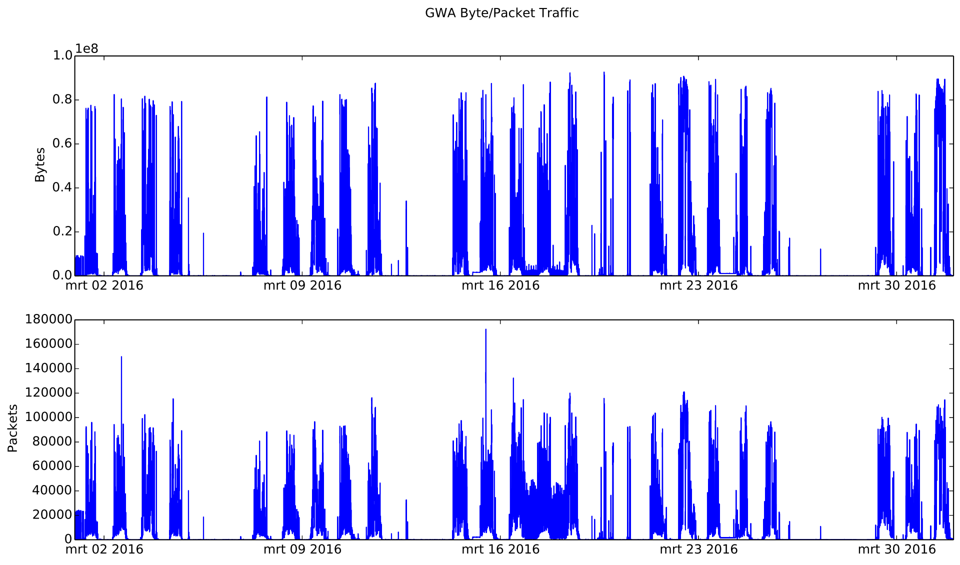
06:00 to 19:00, traffic inside that window does not display any visible patterns.

We can also see in Figure 5.7 that entropy shows a much stronger seasonality in this dataset than ICSX, with weekdays showing a very clear pattern.

5.3 Time-series Accuracy

In this experiment, we test the accuracy of our forecast models on our traffic features. While forecast accuracy is not enough to detect attacks, it is important for the models to accurately portray

Figure 5.5. GWA - Traffic in bytes and packets



normal traffic, so that incoming normal traffic won't deviate from the model and appear anomalous.

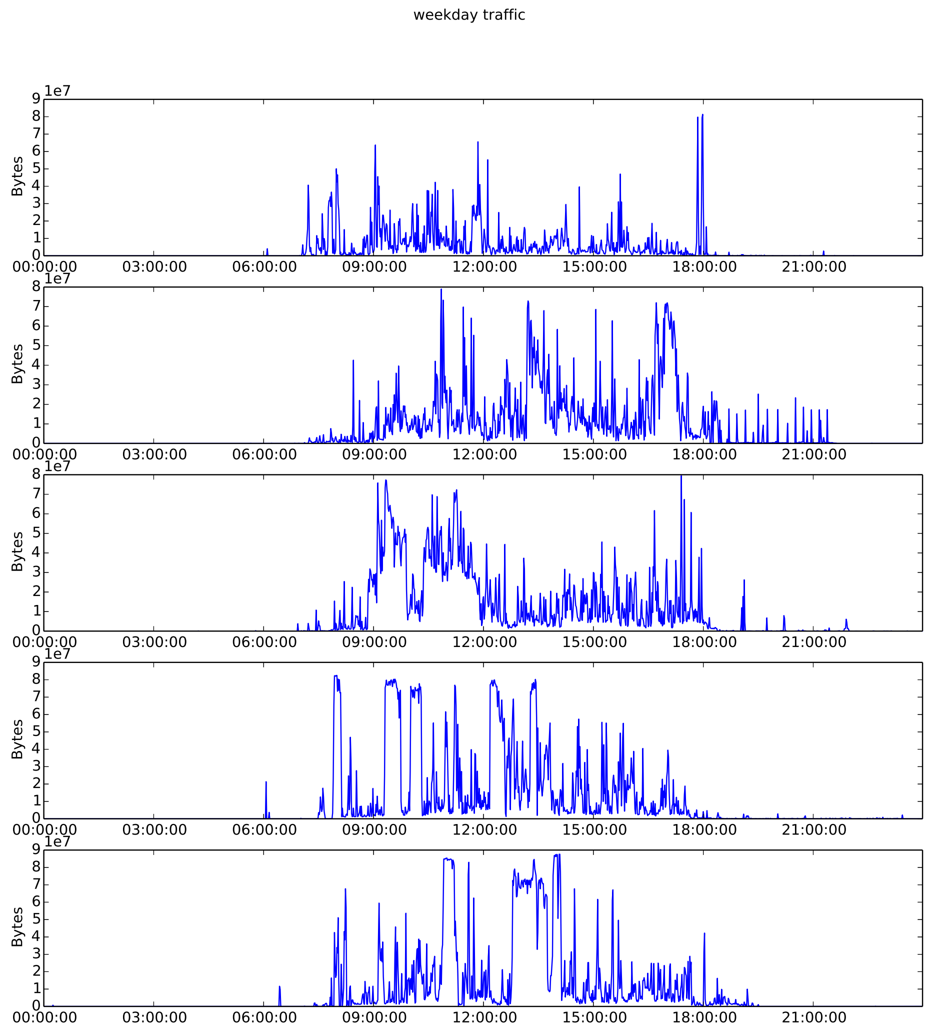
We first test the accuracy of the weekly forecast in a many-steps-ahead fashion, predicting the values of an entire week ahead without using any new values as input. Then, we test the same models with our one-step-ahead forecasting technique explained in subsection 4.3.6, using each newly arrived value as input to the weekly model for predicting the next one. However, we only accept values that fall inside the weekly models 99% prediction intervals as one-step-ahead input, otherwise we replace them with the weekly model's forecasted mean. Without this adjustment, the models would follow actual incoming traffic too closely for any outliers to occur.

5.3.1 Experiment setup

For this test we use the WIDE dataset, as its length allows us to do a rolling-origin window validation. We select 4 weeks of data, create forecast models for week 5 and calculate their accuracy by comparing it week 5's ground truth data. We then slide the window forward by 1 week and repeat the process. In this fashion, we test forecasts 40 times and calculate the averages of all accuracy metrics we track.

Due to the way we detect active ports and only use choose ports with heavy traffic as features, we note that not all sets of training and test data contain the same ports. Results for ports only include slices where those particular ports were active.

Figure 5.6. GWA - Traffic during weekdays



5.3.2 Accuracy Metrics

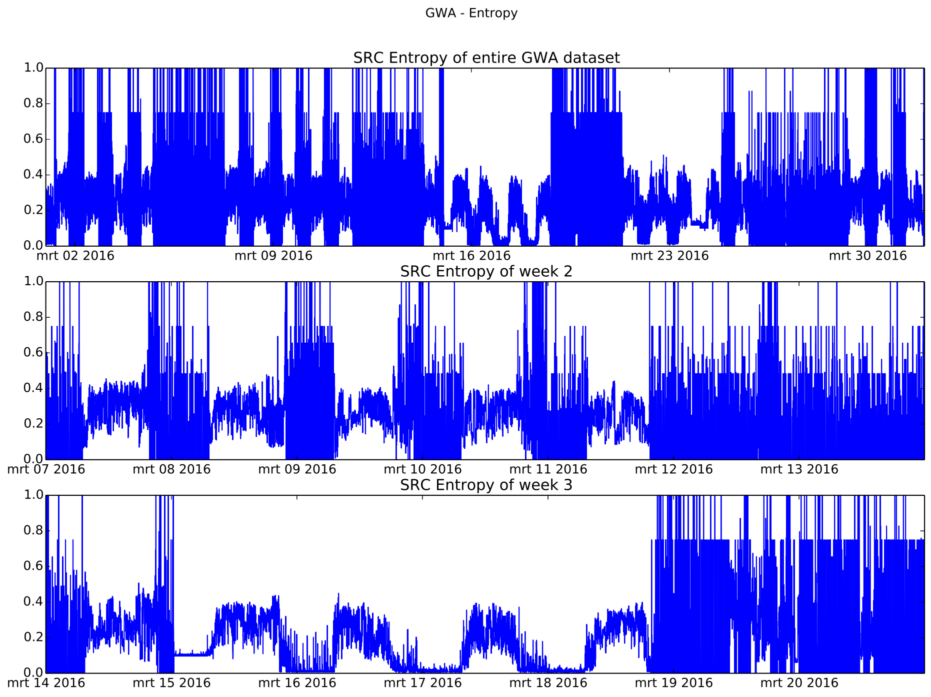
We first define our accuracy metrics.

- Mean Error(ME). Defined as

$$\mathrm{ME} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i) \tag{5.1}$$

It is a scale-dependent error metric. For features with high magnitude such as bytes and packets, it is hard to interpret.

Figure 5.7. GWA - Source entropy of various slices of GWA dataset



- Real Mean Squared Error(RMSE). Defined as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2} \tag{5.2}$$

  Another scale-dependent metric, it is useful for comparing multiple models of a single variable, but not suited for comparison across our different features.

- Mean Absolute Error(MAE). Defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{Y}_i - Y_i|^2 \tag{5.3}$$

  Same issues with RMSE apply here.

- Mean Percentage Error(MPE). Defined as

$$\text{MPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\hat{Y}_i - Y_i}{\hat{Y}_i} \tag{5.4}$$

A scale-free metric. It is useful for comparing errors across features. However, it tends to over-penalize negative errors, and is undefined for zero values.

- Mean Absolute Percentage Error(MAPE). Defined as

$$\text{MPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{Y}_i - Y_i}{\hat{Y}_i}\right| \tag{5.5}$$

Same issues as MPE.

5.3.3 Many-steps ahead

Our results for the many-step ahead forecasting of all our features can be seen in Table 5.1. We can see quite a large amount of error on all features. This is partly to be expected then a forecast predicts values so far in the future, but the results of the protocol and port features show that those features might not contain enough information or be stable enough for predictive models to be created with them.

Table 5.1. Time-series accuracy metrics for 1-week-ahead forecast of WIDE dataset features.

	ME	**RMSE**	**MAE**	**MPE**	**MAPE**
bytes	6.478204e+07	5.149509e+09	3.953088e+09	-5.65847	24.03471
packets	5.774606e+02	1.428433e+06	1.119224e+06	-6.01010	24.08023
srcEntropy	1.132000e-02	4.241000e-01	3.361000e-01	-3.48622	17.33201
dstEntropy	7.070000e-03	3.278200e-01	2.525200e-01	-3.95811	18.13831
TCPBytes	-1.100000e-03	6.092500e-01	4.853500e-01	-38.89398	188.08438
TCPPackets	-1.088930e+00	1.726330e+00	1.560470e+00	-46.89845	80.53729
UDPBytes	1.561900e-01	3.313560e+00	2.734310e+00	-41.14296	147.03575
UDPPackets	-2.800300e-01	8.171930e+00	6.761390e+00	-50.19298	859.61572
OtherBytes	-1.259000e-02	8.696300e-01	6.861900e-01	-4.12633	24.24322
OtherPackets	-9.856800e-01	3.260640e+00	2.663540e+00	-13.61616	74.52146
X80	-1.040550e+00	2.375950e+00	1.862620e+00	-167.12435	273.48359
X22	3.111100e-01	1.984730e+00	1.636350e+00	-26.28551	94.64175
X123	-1.650400e-01	7.882410e+00	6.418860e+00	79.89595	1182.22840
X443	-6.131000e-02	3.532310e+00	2.563950e+00	11667.08834	12782.62391
X873	4.272100e-01	4.072720e+00	3.188720e+00	-325.52906	716.24140

5.3.4 One-step ahead

Our results for one-step ahead forecasting can be found in Table 5.2. The percentage errors are much lower for bytes, packets and the entropy metrics. However the protocol and port forecasts have not improved.

Table 5.2. Time-series accuracy metrics for one-step ahead forecast of WIDE dataset features.

	ME	**RMSE**	**MAE**	**MPE**	**MAPE**
bytes	1.541472e+08	3.810846e+09	2.454939e+09	-2.22783	13.96944
packets	1.186215e+05	1.262325e+06	8.711053e+05	-2.46941	18.03999
srcEntropy	-3.778000e-02	2.802500e-01	1.882700e-01	0.12692	9.40822
dstEntropy	-1.960000e-02	2.343200e-01	1.460000e-01	-1.03414	10.20197
TCPBytes	-2.243000e-02	4.199000e-01	2.755900e-01	-41.17326	121.70299
TCPPackets	-6.548300e-01	1.213400e+00	9.492000e-01	-40.43020	67.29986
UDPBytes	-9.441000e-02	2.621400e+00	1.731770e+00	-14.40646	66.72570
UDPPackets	-3.929400e-01	6.884240e+00	4.702110e+00	30.86631	527.28850
OtherBytes	3.820000e-02	6.262800e-01	4.286000e-01	-3.24164	14.21264
OtherPackets	-3.935800e-01	2.303650e+00	1.488090e+00	-4.31687	56.68770
X80	-6.138300e-01	1.777320e+00	9.166700e-01	-144.20281	247.43745
X22	3.466800e-01	1.556820e+00	9.289400e-01	-21.55513	63.78604
X123	-3.395200e-01	6.659390e+00	4.406540e+00	-6.86401	899.26725
X443	2.093900e-01	2.817260e+00	1.551610e+00	918.70793	1304.70170
X873	-3.848300e-01	2.964210e+00	1.741770e+00	-73.35555	205.75969

5.4 Implementation and performance

5.4.1 Implementation

In this section, we describe the implementation details of our platform.

Packet inspection and log assembly are handled by the Aguri software as detailed in a previous section. Log parsing, data preprocessing and sanitization are done by scripts written in the Python 2.7 programming language [45] with the help of the Pandas library [46]. Time-series forecasts are created using the R programming language [47] and the Forecast package [48,49]. Communication between the 2 languages is handled by the RPY2 python interface [50].

5.4.2 Parameters and thresholds

In this section, we list all values we chose for parameters and thresholds on all levels of our platform.

5.4.2.1 Aguri

We use Aguri version 2.0. The aggregation threshold was set to the default value, 0.1% of traffic, while the span of the summary windows was 1 minute for the ICSX and GWA datasets and 5 minutes for the WIDE dataset. We use verbose output when available and fall back to concise output for the ICSX dataset. Those outputs offer the same data in different format, and our features can be reconstructed from either.

5.4.2.2 Preprocessing

During the preprocessing step, we detect frequently appearing ports in the 0-1023 range and use them as traffic features. Our threshold for considering those ports is if it appears, unaggregated by Aguri on 30% of the traffic summaries we build our model on.

5.4.2.3 Model Creation

Parameters for our forecasting model are estimated automatically during model creation by the R forecast package. For the STL Decomposition, the time window parameter is set to 7, as this number offered better AIC (Akaike information criterion) scores . For the ETS forecast, different model types are tested such as simple exponential smoothing, Holt's linear method, Exponential trend method or multiplicative damped trend method, then the best performing model according to AIC is used.

5.4.2.4 Attack Detection

Our thresholds for attack detection are at a testing stage since we were not able to test our platform on real DDoS attack datasets. We consider as an attack traffic that was outside the 99% forecast prediction intervals for all features simultaneously, for at least 3 consecutive time-windows.

5.4.3 Performance

An important characteristic of a detection system is how fast it is able to perform its intended purpose. Choice of algorithm, specific workflow, implementation tradeoffs, speed of necessary 3rd party applications and the amount of items to be analysed all affect how fast a system can detect attacks. For a detection system to be able to work *online*, act on incoming data as it arrives instead of batch data, it needs to be able to analyse each unit of data before the next one arrives. In this section, we will give some performance metrics for our DDoS detection platform. Those metrics were measured on a workstation with an Intel Core i7-3630QM CPU and 8GB RAM, running an Ubuntu 12.04LTS Virtual Machine on a Windows 7 64-bit operating system.

For benchmarking purposes, we use two small DDoS traces contain DDoS traffic exclusively, provided by [51], in addition to our regular datasets. Trace 1 has an average bitrate of 325 Mbps and average packet rate of 43 kpps over 295 seconds. Trace 2 has an average of 225 Mbps and average packet rate of 32 kpps over 281 seconds.

5.4.3.1 Aguri batch parsing of PCAP file

Aguri can either run in live mode intercepting and analysing traffic as it arrives, or it can process existing data in PCAP format [52]. We measure the following times for extracting features from datasets in PCAP format.

- Trace 1 took a total of 374 seconds of real time. This is a longer timespan than the duration of the pcap file.
- Trace 2 took a total of 244 seconds of real time.
- We also test a single day of the ICSX dataset. Aguri required a total of 570 seconds of real time to process this 86.400 second long dataset.

This shows that Aguri is capable of handling traffic in real time during normal network behaviour. However, its performance on the DDoS traces indicates that during heavy DDoS attacks, Aguri running on our current hardware implementation will be incapable of processing traffic in real time. It is possible that a dedicated hardware platform running Aguri will have better performance, but we did not test such a configuration.

5.4.3.2 Aguri live parsing

We were unable to play back the pcap files at the same rate they were recorded at, so we were not able to measure if Aguri could handle that specific rate of bits and packets. The highest recorded playback rate we could produce was 149.14 Mbps and 21 Kpps, which Aguri was able to process as it arrived.

5.4.3.3 parsing of Aguri log files into features

One of the benefits of Aguri's aggregation is that the generated logs are all roughly the same size, so we do not expect a performance difference between high and low traffic, other than the one Aguri itself introduces. What matters is the number of timeslots to be parsed, or the size of the log file in bytes.

- It took 29 seconds for our platform to extract and preprocess features from the entire ICSX dataset in Aguri format, spanning 10078 timeslots of 1 minute each or 7 days of traffic and 83.3 Mb in size.
- We also test feature extraction on the WIDE dataset, spanning 104828 timeslots of 5 minutes each or roughly a year of traffic and 273.3 Mb in size. Our platform took 219 seconds to extract and preprocess features.

These results show that our platform can easily convert aguri logs into our chosen traffic features in real time. These times include preprocessing tasks such as handling missing data and fixing timezone DST errors.

5.4.3.4 building a weekly model for all features

Building a weekly model of feature traffic is one of the main tasks of our DDoS Detection Platform. Once created, the model is used for a week then replaced with the next one, so this process does not need to be repeated for every timeslot of incoming traffic and does not have a strict performance constraint to meet. Despite that, we measure the time necessary to create those models.

We use the GWA and WIDE datasets for this measurement since ICSX spans less than a week of traffic, not enough for a weekly model to be built on. We initially tested model creation on

Table 5.3. Weekly model generation time

Dataset	Real-time Length	Timeslots	Model Build Time
WIDE	4 weeks	8064	4.19 seconds
WIDE	5 weeks	10055	5 seconds
WIDE	6 weeks	12071	5.74 seconds
WIDE	7 weeks	14087	6.4 seconds
WIDE	8 weeks	16103	7.15 seconds
GWA	4 weeks	44704	18 seconds
WIDE	$\sim$52 weeks	140790	43 seconds

a manufactured repeating version of ICSX as well, one that spanned the required weeks to build models on, but the similarity of the measurements led to a model with extremely narrow prediction intervals that couldn't be used in practice.

We test model generation on parts of the WIDE dataset and the GWA dataset, our results can be seen in Table 5.3. We observe that model generation time, once the required data is loaded in memory and already preprocessed, is actually quite low and linearly related to the number of timeslots in the data. WIDE uses less timeslots per week because its timeslot resolution is 5 minutes instead of 1 minute for GWA.

The short build time for the weekly model, while not needed for our current configuration, leaves open the option of creating these models for each new data point available in real time, as opposed to a weekly process.

5.4.3.5 outlier detection on incoming data

The outlier detection part of our platform is responsible for a number of tasks. For every timeslot, it compares its incoming features against the prediction intervals generated by that feature's weekly model, and detects how many intervals away from the expected mean they are. Once it does so, and if the the feature value was not an outlier, it also feeds those features as input to the weekly model and gets new prediction intervals for the next timeslot.

We test 1 week worth of data on the WIDE and GWA datasets to determine how demanding this process is. However, we note our tests were run in batch mode with the incoming features already loaded into memory, which eliminates possible inefficiencies and overhead from our timings and does not include time necessary for extracting those features (which was measured in a previous

Table 5.4. Outlier detection performance

Dataset	Timeslots	Real Time	Time Taken
WIDE	2016	1 week	90 seconds
GWA	4354	3 days	172 seconds

test). Our findings can be seen in Table 5.4. The time required for a single timeslot is less than 0.05 seconds, making this process able to run on real time data.

Chapter 6

Conclusions

In this thesis, we have implemented a DDoS Detection Platform using Aguri aggregate traffic summaries as input, extracting various time-series features of the traffic and creating forecast models of the features then detecting anomalies when incoming traffic falls outside model prediction intervals for a number of features, over time. The system also has mechanisms for visualizing data, helping users get a better understanding of the data, and how our traffic features behave during normal traffic operations. Unfortunately, we were not able to validate the system on a realistic DDoS dataset, therefore we suggest this system be in a real environment as an aid to users in the task of DDoS detection who would then calibrate the system on DDoS attacks as they happen.

In the following section, we will present our conclusions as well as issues we identified in all stages of our approach.

6.1 Issues

6.1.1 Aggregation

One of the main decisions of this thesis was to use lossy traffic summaries instead of full traffic information as raw data, with the goal of exploring if summaries can be descriptive enough to use as DDoS detection input. However, the specific mechanics of Aguri's aggregation proved to have a damaging effect on our features. Since IPs and ports were aggregated based on hard thresholds, Aguri's output would show drastic changes at areas near the threshold, making distribution-related features such as entropy very volatile. The number of IPs and subnets can drastically re-arrange itself with very small changes in traffic. In addition to that, our performance tests in subsection 5.4.3 showed that one of the expected benefits of aggregation, increased robustness of the listener software during DDoS attacks, failed to materialize. Aguri, using our hardware, could not handle a high-rate 300 Mbps attack in real time. This forces us to re-examine the usefulness of Aguri's traffic aggregation for DDoS detection. Offline detection however is feasible using Aguri, as the necessary processing time of even high-rate DDoS attacks remained at a level above, but close to the actual duration of the DDoS attack.

6.1.2 Feature Selection

Our features were chosen with the goal of describing the structure of traffic as well as the amount of it. The simple byte and packet features proved to be informative as well as able to generate accurate models of future traffic. The sample entropy feature however suffered from a significant loss of accuracy without the knowledge of how many IPs comprised each subnet appearing in the data. While attacks could still be identified using it, this was due to the volume of the attack dominating traffic and stabilizing the IP distributions to exactly the parameters of the DDoS attack. Low-rate attacks that altered the distributions in a less obvious way were not captured.

The protocol and port features were more informative than entropy and retained the seasonal characteristics of traffic, which made their forecast models more accurate.

6.1.3 Dataset Selection

Our method of choice for detecting attacks, traffic forecast models and structural features, proved to be restrictive in the type of dataset require to test it on. Our attempt to exploit the daily and weekly seasonalities of traffic required a significant amount of past data to calibrate the forecast models on and publicly available DDoS datasets such as ICSX were significantly smaller. Also, injecting or artificially creating DDoS attacks on normal traffic of appropriate length like the WIDE or GWA datasets would not capture the expected interactions of DDoS traffic with the victim network's normal traffic. The stifling effect of DDoS attacks would not be shown in the dataset. As such, we were unable to conduct accuracy testing on our attack detection method.

6.2 Future Work

In the process of creating this thesis, we had the opportunity to research many aspects of network traffic attack detection.

In the area of time-series analysis of network traffic features, we believe that the accuracy of our forecasts can be increased by using an ARIMA model supplied with Fourier terms as external regressors. Such a model will preserve the dual seasonality forecasting capabilities of our current model while also able to model holiday effects such as Easter or Christmas. Also, another possible improvement we wish to explore is generating weekly models more often during our procedure, from weekly to hourly or even real time, as our performance tests showed that this is feasible.

As for the general subject of DDoS detection, we believe that while automatic anomaly detection systems are attractive in theory, there is a significant progress to be made in the area of support tooling helping human analysts conduct detection. Traffic data is extremely varied, large in size, hard to label and usually confidential, which makes automated detection methods hard to calibrate. However, supportive tasks such as interfaces for dynamic aggregation and visualization were shown to be useful in identifying possible anomalies and understanding the underlying structure of traffic.

In the future, we will examine ways to create varied summaries and visualizations of more complete traffic data, including header information and payload. Another goal of visualization is to provide a ’teaching material’ for security experts to learn how network traffic features behave under ’normal’ conditions, and use that knowledge to further calibrate attack detection systems.

Bibliography

[1] B. Sanou, “The world in 2015: Ict facts and figures,” *International Telecommunications Union*, 2015.

[2] T. Dubendorfer, A. Wagner, and B. Plattner, “An economic damage model for large-scale internet attacks,” in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on*. IEEE, 2004, pp. 223–228.

[3] T. Matthews, “Incapsula survey: what ddos attacks really cost businesses,” 2014.

[4] D. McPherson, R. Dobbins, M. Hollyman, C. Labovitzh, and J. Nazario, “Worldwide infrastructure security report, volume v, arbor networks,” 2016.

[5] K. Cho, R. Kaizaki, and A. Kato, “Aguri: An aggregation-based traffic profiler,” in *International Workshop on Quality of Future Internet Services*. Springer, 2001, pp. 222–242.

[6] J. Mirkovic and P. Reiher, “A taxonomy of ddos attack and ddos defense mechanisms,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[7] S. M. Specht and R. B. Lee, “Distributed denial of service: Taxonomies of attacks, tools, and countermeasures.” in *ISCA PDCS*, 2004, pp. 543–550.

[8] S. Kandula, D. Katabi, M. Jacob, and A. Berger, “Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds,” in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 287–300.

[9] I. Hamadeh and G. Kesidis, “Performance of ip address fragmentation strategies for ddos traceback,” in *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*. IEEE, 2003, pp. 1–7.

[10] T. K. Law, J. C. Lui, and D. K. Yau, “You can run, but you can’t hide: an effective statistical methodology to trace back ddos attackers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 9, pp. 799–813, 2005.

[11] S. Yu, W. Zhou, R. Doss, and W. Jia, “Traceback of ddos attacks using entropy variations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 412–425, 2011.

[12] L. Spitzner, *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003, vol. 1.

[13] V. Paxson, “An analysis of using reflectors for distributed denial-of-service attacks,” *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 3, pp. 38–47, 2001.

[14] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing network-wide traffic anomalies,” in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[15] A. Lakhina, M. Crovela, and C. Diot, “Mining anomalies using traffic feature distributions,” in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 217–228.

[16] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, “An empirical evaluation of entropy-based traffic anomaly detection,” in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 151–156.

[17] M. Li, “An approach to reliably identifying signs of ddos flood attacks based on lrd traffic pattern recognition,” *Computers & Security*, vol. 23, no. 7, pp. 549–558, 2004.

[18] L. Li and G. Lee, “Ddos attack detection and wavelets,” *Telecommunication Systems*, vol. 28, no. 3-4, pp. 435–451, 2005.

[19] Y. Chen, K. Hwang, and Y.-K. Kwok, “Filtering of shrew ddos attacks in frequency domain,” in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN’05) l*. IEEE, 2005, pp. 8–pp.

[20] M. Li, “Change trend of averaged hurst parameter of traffic under ddos flood attacks,” *Computers & security*, vol. 25, no. 3, pp. 213–220, 2006.

[21] H. Rahmani, N. Sahli, and F. Kammoun, “Joint entropy analysis model for ddos attack detection,” in *Information Assurance and Security, 2009. IAS’09. Fifth International Conference on*, vol. 2. IEEE, 2009, pp. 267–271.

[22] H. Liu and M. S. Kim, “Real-time detection of stealthy ddos attacks using time-series decomposition,” in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–6.

[23] J. Viinikka, H. Debar, L. Mé, and R. Séguier, “Time series modeling for ids alert management,” in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 102–113.

[24] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, “Lads: Large-scale automated ddos detection system.” in *USENIX Annual Technical Conference, General Track*, 2006, pp. 171–184.

[25] E. Keogh, J. Lin, and A. Fu, “Hot sax: Efficiently finding the most unusual time series subsequence,” in *Fifth IEEE International Conference on Data Mining (ICDM’05)*. IEEE, 2005, pp. 8–pp.

[26] H. Cheng, P.-N. Tan, C. Potter, and S. A. Klooster, “Detection and characterization of anomalies in multivariate time series.” in *SDM*, vol. 9. SIAM, 2009, pp. 413–424.

[27] B. Y. Reis and K. D. Mandl, “Time series modeling for syndromic surveillance,” *BMC Medical Informatics and Decision Making*, vol. 3, no. 1, p. 1, 2003.

[28] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2014.

[29] B. Claise, B. Trammell, and P. Aitken, “Specification of the ip flow information export (ipfix) protocol for the exchange of flow information,” 2013.

[30] D. R. Morrison, “Patricia: practical algorithm to retrieve information coded in alphanumeric,” *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[31] R. Kieschnick and B. D. McCullough, "Regression analysis of variates observed on (0, 1): percentages, proportions and fractions," *Statistical modelling*, vol. 3, no. 3, pp. 193–213, 2003.

[32] J. W. Taylor, "Exponential smoothing with a damped multiplicative trend," *International journal of Forecasting*, vol. 19, no. 4, pp. 715–725, 2003.

[33] C. Holt, "Forecasting trends and seasonals by exponential weighted averages," *ONR Memorandum*, vol. 52, p. 1957, 1957.

[34] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.

[35] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990.

[36] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting*, vol. 18, no. 3, pp. 439–454, 2002.

[37] C. Elkan, "Results of the kdd'99 classifier learning," *ACM SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 63–64, 2000.

[38] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[39] "Nsl-kdd data set for network-based intrusion detection systems," http://nsl.cs.unb.ca/KDD/NSL-KDD.html, accessed: 2016-08-15.

[40] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, M. K. Reiter *et al.*, "Playing devil's advocate: Inferring sensitive information from anonymized network traces." in *NDSS*, vol. 7, 2007, pp. 35–47.

[41] D. Koukis, S. Antonatos, and K. G. Anagnostakis, "On the privacy risks of publishing anonymized ip network traces," in *IFIP International Conference on Communications and Multimedia Security*. Springer, 2006, pp. 22–32.

[42] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[43] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Towards generating real-life datasets for network intrusion detection,” *Int. J. Netw. Secur*, vol. 17, no. 6, pp. 675–693, 2015.

[44] A. Kato, J. Murai, S. Katsuno, and T. Asami, “An internet traffic data repository: The architecture and the design policy,” in *INETâĂŹ99 Proceedings*, 1999.

[45] G. Van Rossum *et al.*, “Python programming language.” in *USENIX Annual Technical Conference*, vol. 41, 2007.

[46] W. McKinney *et al.*, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, vol. 445, 2010, pp. 51–56.

[47] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/

[48] R. J. Hyndman, *forecast: Forecasting functions for time series and linear models*, 2016, r package version 7.1. [Online]. Available: http://github.com/robjhyndman/forecast

[49] R. J. Hyndman and Y. Khandakar, “Automatic time series forecasting: the forecast package for R,” *Journal of Statistical Software*, vol. 26, no. 3, pp. 1–22, 2008. [Online]. Available: http://www.jstatsoft.org/article/view/v027i03

[50] “rpy2 - r in python.” http://rpy2.bitbucket.org/, accessed: 2016-08-24.

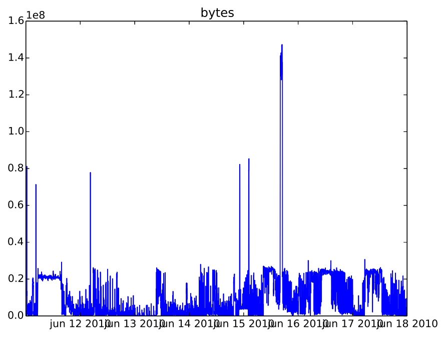
[51] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, “Booters - an analysis of ddos-as-a-service attacks,” in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 243–251.

[52] V. Jacobson, C. Leres, and S. McCanne, “pcap-packet capture library,” *UNIX man page*, 2001.

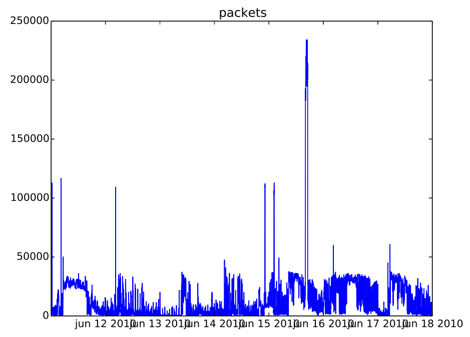
Appendix A

Appendix A

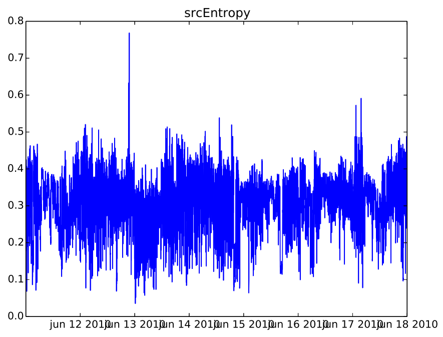
Graphic visualizations of all features of the ICSX database can be found here.



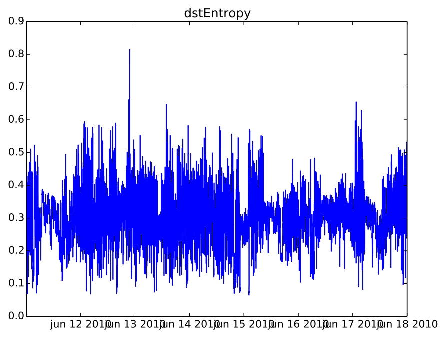
(a) bytes



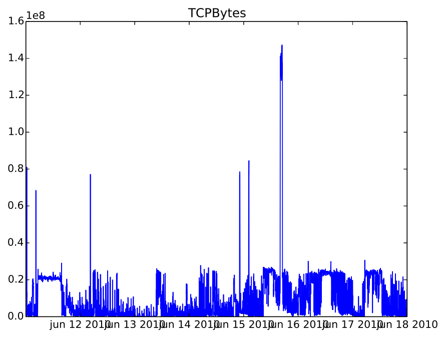
(b) packets



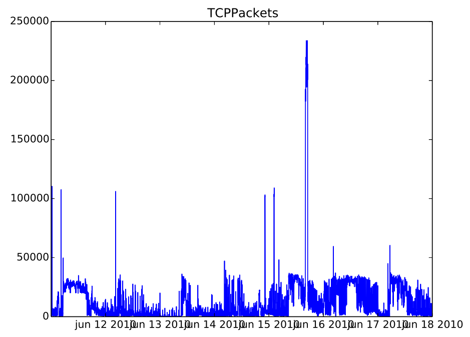
(c) source entropy



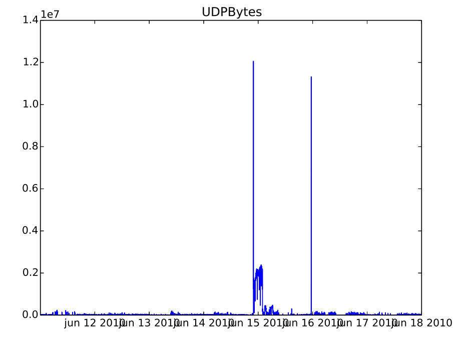
(d) destination entropy



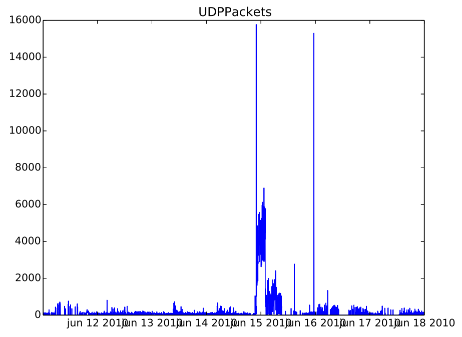
(e) TCP bytes



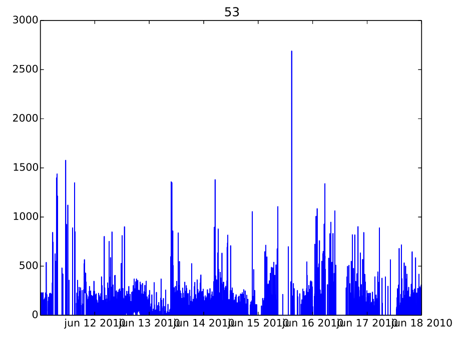
(f) TCP packets



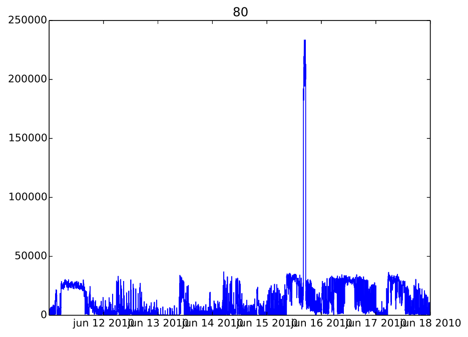
(a) UDP bytes



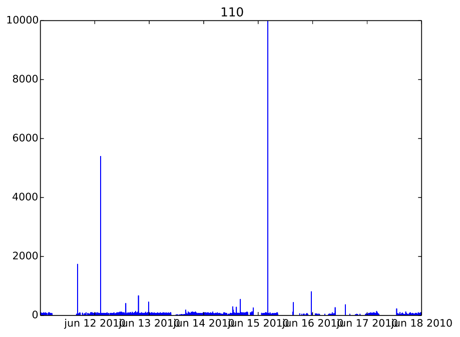
(b) UDP packets



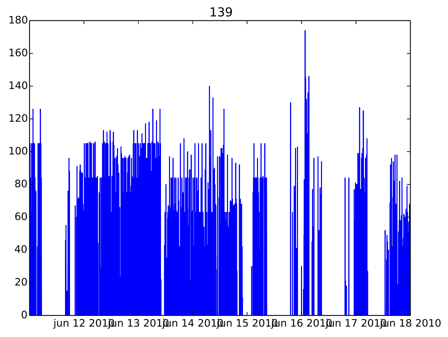
(c) Port 53



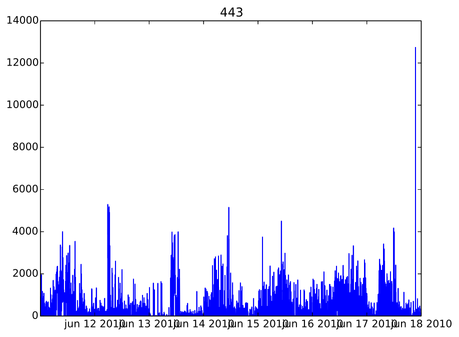
(d) Port 80



(e) Port 110



(f) Port 139



(e) Port 443