1pt

EE40150 - Final Year Project: Investigation into the Precision Time Protocol

James Cox
Department of Electrical and Electronic Engineering

University of Bath

May 5, 2014

Abstract

There is an ever increasing importance to synchronise clocks or processes in several industries including telecommunications and power distribution. Use is made of the high precision clocks on-board the Global Positioning System (GPS) for synchronisation but this method is susceptible to jamming. Therefore solely relying on this technology would not be wise and an alternative method should be used alongside GPS. One example is a distributed network for synchronisation called the Network Time Protocol (NTP) or the Precision Time Protocol (PTP). This project investigates the newer of the two technologies (PTP) and investigates how it performs over an Ethernet network. Some existing hardware and software clocks were used to try and quantify network delay and how it changes at different points along the network. Finally packet metrics were used to attempt to quantify this delay and to compare test results against each other.

Contents

Acronyms

BMC Best Master Clock

CSAC Chip Scale Atomic Clock

CSMA/CD Carrier Sense Multiple Access with Collision Detection

CSV Comma Separated Variable

E2E End-to-End

GM Grandmaster

GNU GNU's Not Unix

GPS Global Positioning System

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronic Engineers

ITU International Telecommunication Union

LAN Local Area Network

MAC Media Access Control

MAFE Maximum Average Frequency Error

MATIE Maximum Average Time Interval Error

NERC North American Electric Reliability Company

NTP Network Time Protocol

P2P Peer-to-Peer

PPS Pulse per Second

PTP Precision Time Protocol

PTPd PTP Daemon

SA Security Association

SNTP Simple Network Time Protocol

SRS Stanford Research Systems

TAI International Atomic Time

TDEV Time Deviation

UTC Coordinated Universal Time

List of Figures

List of Tables

List of Code Extracts

*List of Code Extracts

1 Introduction

This section has been taken from the Interim Report [1] and has been modified. Some sections however may be exact copies.

In a number of high profile applications there is an ever increasing requirement for high accurate clocks. These clocks may be used for generating an accurate timestamp (used in telecommunications) or a method to synchronise processes in the automotive industry. Multiple clocks will be used in these applications, and thus they all need to be synchronised with one another.

It may not be feasible to have a high accuracy atomic clock (or Chip Scale Atomic Clock (CSAC)) due to space or cost constraints. An alternative to this would be to use a GPS receiver and time can then be synchronised to the accurate atomic clocks on-board the GPS satellites. Using high accuracy clocks like the ones mentioned would help to reduce any clock inaccuracies.

There are two main types of clock inaccuracies. Firstly they may have started at a different time relative to the others. Adjusting for this error is called offset correction. The second effect is caused by a difference in clock speeds. Therefore they need to be continuously adjusted which is called drift correction. The amount of drift correction required depends on the quality of the clock. Appendix 1 shows a table of clocks and their corresponding accuracies.

The following industries are examples that require this level of timing accuracy.

Automation Industry
Processes will need to be synchronised in order for the overall system to function. This can only be realised if their clocks are in sync with one another. If clocks are in sync then processes can also be separated away from communication between each machine and the processing of the control commands. [2]

Power Transmission
Time synchronisation is very important in the power transmission industry. An example of a situation where timing would have mitigated a fault from occurring is the North American blackout in August 2003 [3]. It made it difficult for the investigation team to be able to sort through the data received when the timestamps were gathered from an inaccurate clock. From the events of this blackout a regulation was put in place to define a minimum absolute accuracy for timestamped data. The adoption of the North American Electric Reliability Company (NERC) Standard PRC018-1 in 2006 [4] made it a requirement for any substation in the USA to log data to a minimum accuracy. The timestamped data must be accurate to within 2ms relative to Coordinated Universal Time (UTC).

Telecommunications
In telecommunications, timing protocols are considered when networks need to be synchronised or if mobile base stations need synchronisation pulses. With the increase in GPS jamming, systems such as 4G mobile telephony must rely on other timing methods in case GPS is affected.

All of the industries mentioned above could feasibly use GPS for a highly accurate timing reference if there was an issue with the system, for example a jamming incident, then there will be some major consequences should timing drift. It is known that jamming GPS receivers is becoming more common and thus an alternative method of time synchronisation should be use [5]. There are multiple cases where jamming may occur (both accidental and intentional) and thus a way of synchronising time with this threat must be realised.

One way of realising this is by using a distributed timing system, such as Network Time Protocol (NTP) or Precision Time Protocol (PTP). This would allow for multiple nodes to synchronise their clocks with a more accurate time source.

1.1 How the Network Time Protocol (NTP) Works

This section has been taken from the Interim Report [1] and has been modified. Some sections however may be exact copies.

This is a technology originally designed in 1985 and used to synchronise clocks over a packet switched network. It is able to achieve synchronisation with UTC within a few milliseconds and can maintain sub-millisecond accuracy on a Local Area Network (LAN) if ideal conditions are met. Errors due to different packet routes or network congestion can reduce this accuracy by 100ms or more [6].

NTP uses a client-server hierarchy split into stratums. Figure 1 on the next page shows the typical stratums numbered from 0 to 3.

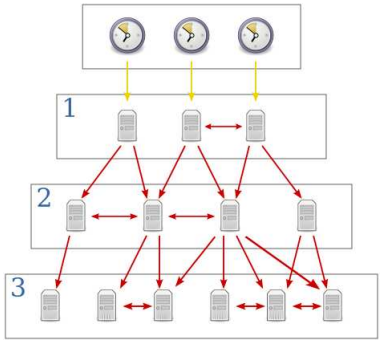


Figure 1: NTP Network Hierarchy

The reference clocks located in stratum 0 are high precision. They achieve this by either being atomic clocks or they utilise GPS synchronisation. The clocks in stratum 2 will base their time off of the clocks in stratum 1. A selection of the clocks in stratum 1 will be used for time synchronising each clock in stratum 2. This is done so that the time is more accurate and robust. Within a stratum, clocks may also synchronise each other for sanity checking to ensure that all clocks within a stratum are accurate between each other. Any layers below Stratum 2 will mirror the same algorithm, and there can be up to 15 layers. Stratum 16 is reserved for clocks that are not synchronised with NTP [7].

Simple Network Time Protocol (SNTP) is used in applications which do not require a high timing accuracy and achieves this by ignoring drift values. Therefore it is recommended that SNTP is only used in the higher stratums [8]. The SNTP specification is part of the NTP specification [9].

Issues arise when synchronising time over a packet switched network where sub millisecond accuracies are required. PTP was developed as a successor to the existing NTP standard which aims to reach higher levels of accuracy. Meeting this value of accuracy is very difficult however with a traditional Ethernet network.

When standard switches are used, the packet delay between two nodes is indeterminate. This may be because the packet route from A to B changes depending on network load, or a packet may be held in a switch for an unspecified amount of time whilst working with other data. Therefore this is undesired for time synchronisation as this packet delay must be taken into account when working out the clock offset. Specific timing switches can be used which will prioritise PTP packets, but these may not be available in existing networks or be too expensive to be suitable.

1.2 How the Precision Time Protocol (PTP) Works

There were few important elements to understand about how PTP works before the project could start. These were general PTP operation and the BMC algorithm. A comparison between NTP and PTP was made. In addition to this security aspects surrounding PTP were also analysed.

1.2.1 PTP Explanation

PTP uses a similar master-slave hierarchy of NTP, but it does not use the stratum method. Instead it uses domains which separate out PTP synchronisation networks. The master clock for the domain will broadcast out the current time to all of the other clocks on the network using a multicast message. In IEEE1588-2008 [10] this can occur up to one message every 32 $\frac{1}{4}$ ms. Figure 2 below outlines the time delays of messages sent between the grandmaster (Time Server) and the slave (User).

This diagram has been based off of the PTP diagram on the website cited here [11].

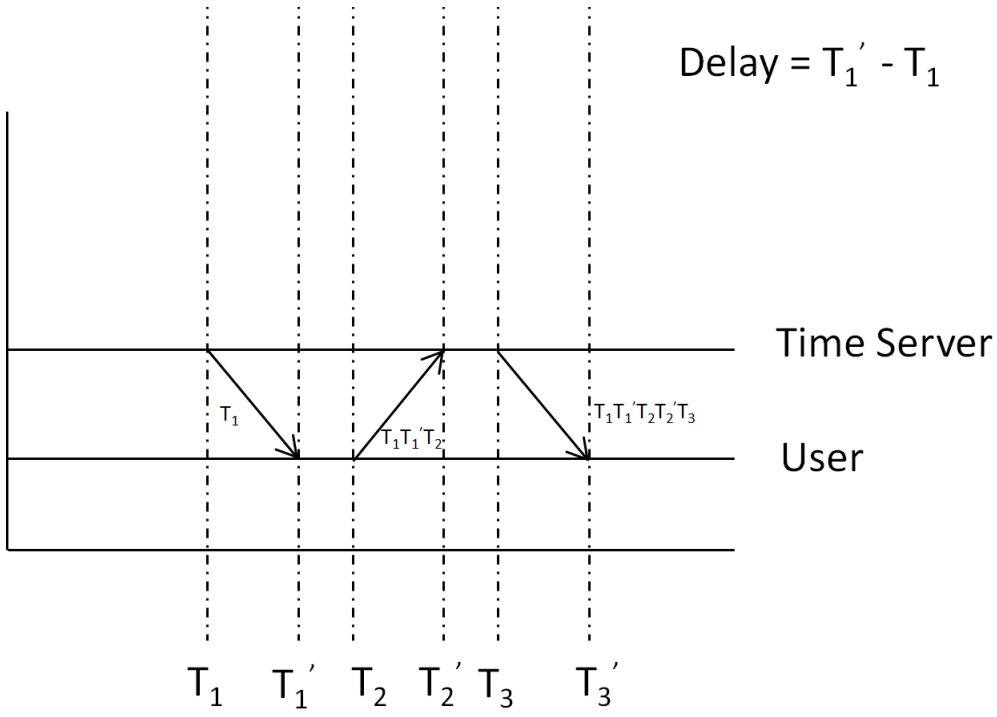


Figure 2: PTP timing diagram

The list below shows the basic steps that PTP follows [10]:

1. Broadcast begins at t_1 where the master sends a *sync* message to all clocks on the domain.
2. Each slave clock takes a note of when the message was received using their local clock. This timestamp is labelled $t_1^{'}$.
3. An optional *follow_up* message may be sent which includes an accurate timestamp of t_1. This step occurs if the master clock does not have the capability to create an accurate timestamp when sending the *sync* message.
4. In order for the slave to synchronise with the master, the round trip delay needs to be known. Therefore a *delay_req* message is sent by the slave clock at time t_2.
5. The master will respond to this message with a *delay_resp* message. This timestamp is called $t_2^{'}$.

At this point t_1, $t_1^{'}$, t_2 and $t_2^{'}$ are now known.

If we define d as the transit time, and $\widetilde{o}$ as the constant offset between the two clocks, the one one delay is:

$$t_1^{'} - t_2 = \widetilde{o} + d \tag{1}$$

$$t_2^{'} - t_2 = -\widetilde{o} + d \tag{2}$$

If we rearrange equation 2 for d :

$$d = t_2^{'} - t_2 + \widetilde{o} \tag{3}$$

Substituting Equation 3 into 1:

$$t_1^{'} - t_1 = \widetilde{o} + t_2^{'} - t_2 + \widetilde{o} \tag{4}$$

$$t_1^{'} - t_1 - t_2^{'} + t_2 = 2\widetilde{o} \tag{5}$$

$$\widetilde{o} = \frac{t_1^{'} - t_1 - t_2^{'} + t_2}{2} \tag{6}$$

The offset is now known and can be adjusted for. The following assumptions have been made to create the calculations above.

1. Message exchange occurs over such a significantly short period of time that the delay can be assumed to be constant.
2. Transit time is symmetrical (i.e. time from master to slave is the same as slave to master).
3. Both the slave and the master can measure the transmit and receive times of messages accurately (ignoring clock drift).

These assumptions are important to be able to then calculate the delay offset. PTP uses a few other methods so these assumptions are not required in practice.

1.2.2 Comparison with NTP

The Precision Time Protocol (PTP) was first developed in 2003 with the intention to build on the existing NTP standard. A new PTP standard was introduced in 2008 with additional functionality such as boundary clocks and using domains instead of subdomain fields. NTP is used by Microsoft to synchronise clocks over the internet, whereas PTP is used in industry where higher accuracy is needed. These changes and comparison with NTP have been tabulated below, Table 1.

Table 1: NTP vs PTP Version 1 vs PTP Version 2

Feature	NTP	IEEE1588-2002	IEEE1588-2008
Time System	UTC	International Atomic Time (TAI)	TAI - can choose epoch
Transparent Clocks	No	No	Yes
Unicast	No	No	Yes
Domains	Subdomain Name Fields	Subdomain Name Fields	Domain Numbers
Clock Quality	None	Data Field Stratum	Clock Accuracy
Selection Algorithm For Best Clock	Unknown	Election Based	Hierarchical
Unique Features	None	Noise Reduction	Alternate Time Scale Grandmaster Cluster Unicast Masters Alternate Master Path Trace

1.2.3 A Typical PTP Network

PTP networks can be configured in a number of different ways. The decisions on what hardware to use depends on the overall network complexity and the required accuracy of the time synchronisation.

Grandmaster Clock The grandmaster clock is the main source of time synchronisation within the same PTP domain. This clock will consist of a highly accurate timing source, such as a CSAC or be based off of a GPS time reference. This clock is picked using the BMC algorithm among all of the other possible master clocks.

Ordinary Clock An ordinary clock is a PTP clock with a single Ethernet port. They are also called nodes in a PTP network. These are the most common types of clocks on the PTP network as these are the end nodes that are then connected to devices that require synchronisation.

Boundary Clock A boundary clock is a replacement to a standalone switch. It usually has multiple PTP ports and thus provides a link between domains. Boundary clocks prioritise PTP packets in order for the one way delay to be minimised.

Transparent Clock A transparent clock main role is to account for switch delay by updating the time interval field of the PTP packet. There are two types of transparent clocks which might be used in a typical network which are End-to-End (E2E) and Peer-to-Peer (P2P) transparent clocks.

They both measure the event message transit time (also known as the resident time) for both *sync* and *delay_rq* messages. This information is then added to the correction field in the two messages. The slave clock can then use this information to work out a more accurate offset. Note however that E2E clocks in particular do not account for the propagation delay of the link.

A P2P clock also takes into account the upstream delay, which is the propagation delay between the two transparent clocks. This time is then added on to the offset mentioned previously.

The figure below (Figure 3 [12]) is an example of a PTP network.

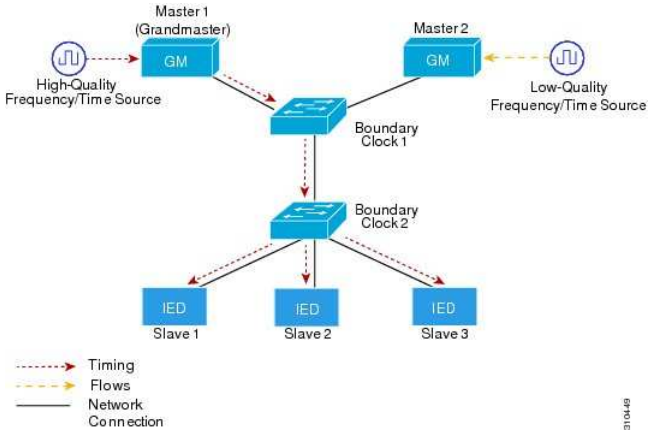


Figure 3: A Typical PTP Network

There are several different network configurations that could be used when using a PTP network. The most minimal network configuration would include a grandmaster clock and a single slave, but they can get more complicated if boundary and transparent clocks are used.

There are some areas of concern with PTP that were addressed in this report. It is unknown how well PTP operates on a busy network when standard switches are used instead of boundary clocks and there are some security concerns with PTP as all messages are transmitted in plain text.

1.2.4 Best Master Clock (BMC)

The BMC algorithm is used to determine the most suitable clock to be the grandmaster of a PTP network. The following criteria are used when determine which clock on the network is the best.

Identifier This is a unique identifier which is constructed from the device's Media Access Control (MAC) address.

Quality This refers to the technology used to implement the clock, and takes into account the expected time deviation.

Priority This is an administrator assigned precedence to select a particular grandmaster. In IEEE1588-2008 there are two 8bit priority fields.

Variance This refers to the stability of the clock based on its performance against the PTP reference.

The 2008 standard uses a hierarchical selection algorithm system, which is outlined below.

1. Priority 1
2. Class
3. Accuracy
4. Variance
5. Priority 2
6. Unique Identifier (tie break)

The BMC algorithm is used to determine each master, and ultimately the grandmaster of the entire PTP network.

1.3 Project Description

This project aims to investigate PTP performance on a heavily used Ethernet network, and to attempt to quantify PTP performance. This will be done using both raw delay data and calculating a number of different packet metrics. There are also some deliverables as part of the Final Year Project which are an interim report, a log book, a final year report and a poster. These deliverables, along with the sub tasks created to complete the project them have been detailed in the Gantt chart, as seen in Appendix 2. This has also been tabulated in the same appendix.

The following project objectives have been identified:

Learn about PTP and other work in relation to the protocol
This stage would occur at the beginning of the project to understand how PTP works. This is important so work can then be carried out to investigate PTP performance on a network. This section will also include any relevant literature reviews.

Collect PTP Data
In parallel with the above, PTP data can be collected. This will be monitoring the performance of PTP across the network as well as how using multiple types of grandmaster/slaves affect the performance. Different clock locations in the network will also be considered.

Implement packet metric scripts
To be able to understand the performance of the network, some packet metric scripts will be created. A suitable language will be chosen once this part of the project begins.

Determine packet performance using these scripts
In combination with the collected data and the metric script development, the scripts will be run with the data collected and PTP performance will be evaluated based on them. Multiple window sizes and types of metric will be used to quantify network performance.

Test Chronos' equipment and provide feedback
As Chronos has provided this project with some equipment, this equipment will also be thoroughly tested and any information gathered can be passed to them once the project is completed.

Documentation
It was noted that it would be beneficial to document as much as possible about the hardware used, which can then be fed back to Chronos if need be. This would also be useful internally.

Investigate into security measures
Some research into how to secure PTP will be made along with some recommendations where applicable. As PTP messages are sent in plain text there is room for exploitation or hacking of the protocol.

2 Literature Review

With the following objectives and tasks in mind, a literature review was performed on any relevant material. The material in the following section is directly relevant to the project. Any further reading will be discussed later in the report.

2.1 Definitions and Terminology for Synchronisation in Packet Networks

The first paper [13] is an International Telecommunication Union (ITU) recommendation paper titled "Definitions and Terminology for Synchronisation in Packet Networks". It defines a number of definitions and terms when dealing with packet synchronisation. The areas of interest in this report were packet metrics which are found in Appendix I3 and I4. These can be split into three sections: Packet Selection Methods, Packet Metrics without Pre-filtering, and Packet Metrics with Pre-filtering.

2.1.1 Packet Selection Methodology

There are two main methods of selecting packets when calculating a packet metric: either using a selection technique at the same time as the packet metric calculation, or as a pre-processing technique before the metric calculation is performed.

Packet selection when integrated with the calculation is very useful when the behaviour of a network is to be determined with respect to its packet delay variation. This is because it provides a generic method that is independent to a particular slave clock implementation [13]. This packet selection method is also known as a Class B metric.

The other method uses a pre-processing technique which preselects packets from a time window. By doing this the process will average out any inconsistencies in the delays, thus resembling a clock running in steady state. Therefore this method is more suitable when trying to specify network limits. This is known as a class A metric. Based on the definitions above, Class B metrics were focussed on throughout this project as they were decided to be more relevant.

There are four examples of packet selection methods that are mentioned in the recommendation report. These are: Minimum Packet Selection Method, Percentile Packet Selection Method, Band Packet Selection Method and Cluster Range Packet Selection Method. The normal packet selection method uses a mean value across the window, whereas the minimum packet selection method takes the lowest value. The band packet and percentile packet uses a band mean method, with the percentile method setting the lower band to 0.

It was decided that the cluster range packet selection method was not required for this project as there were enough methods already discussed that would be useful to compare them against. This packet selection method will therefore not be implemented with the other metrics.
The other method of packet selection is using pre-filtering before the metric is calculated. An averaging function is applied to the set of data. It is important that the data is not filtered too much or data will be lost. This filtered packet sequence can then applied to the metrics mentioned previously in the report. Pre-filtered metrics are useful as they can help specify network limits.

Note that these metrics were not initially planned to be implemented, so the full explanation will not be added here. Instead these can be found in the report [13].

2.1.2 Packet Metrics without Pre-filtering

The first packet method technique discussed is Time Deviation (TDEV). It is used to specify network wander limits for timing signals and can also be used for packet data. TDEV can be applied to both integrated and pre-processed packet selection methods.

The implementation equations are quoted in the reference. The approximation equations were used when implementing the functions.

2.1.3 Time Deviation (TDEV)

The basic TDEV equation has been given below, in Equation 7.

$$TDEV(\tau) = \sqrt{\frac{1}{6n^2}\langle[\sum_{i=1}^{n}(x_{i+2n} - 2x_{i+n} + x_i)]^2\rangle} \tag{7}$$

Because of the angular brackets denoting an ensemble average sum, this would not be possible to be implemented. Therefore there is an approximation, given below in Equation 8. From now on any equations quoted will be in this approximate form.

$$TDEV(n\tau_0) \equiv \sqrt{\frac{1}{6(N-3n+1)}\sum_{i=1}^{N-3n+1}[x(i+2n) - 2x(i+n) + x(i)]^2]} \tag{8}$$

for n = 1, 2, ... Integer part ($\frac{N}{3}$)

The value in brackets next to x denote the window that the particular selection method will be operating over. It has been assumed for this to mean that the window is weighted to 1, with the centre point in the window being the current point. Therefore care will need to be taken when implementing these correctly. The mean of each of these windows will be taken for the standard TDEV case. The minTDEV uses the same approximation formula as above, but the minimum value is taken of the window instead of the mean.

There are pros and cons for both of these metrics. Firstly the minTDEV calculation is suitable for packet delay noise processes but not for frequency offset calculation [13]. One downside however is that it is very sensitive to low lying outliers.

The next set of packet metrics discussed in the document was those based on the band mean. This is a mean that is calculated over a subset of the window size. An equation for this has been given below.

$$x_{band\_mean}(i) = \frac{1}{m}\sum_{j=a}^{b} x_j + 1 \tag{9}$$

The approximation equation for this metric has been provided below in Equation 10.

$$bandTDEV(n\tau 0) \equiv \sqrt{\frac{1}{6(N-3n+1)} \sum_{i=1}^{N-3n+1} [x_{band\_mean}(i+2n) - 2x_{band\_mean}(i+n) + x_{band\_mean}(i)]^2} \quad (10)$$

for n = 1, 2, 3, ... Integer part ($\frac{N}{3}$)

The last metric type used in this set was a percentile TDEV. This is very similar to the bandTDEV, but the value of a (the lower percentage) is set to 0.

2.1.4 MATIE

The next set of packet metrics performed were Maximum Average Time Interval Error (MATIE) and Maximum Average Frequency Error (MAFE). These are used to be able to analyse any time error of a set of packet delay . The estimation formula for MATIE has been provided below.

$$MATIE(n\tau_0) \equiv \max_{1 \le k \le N-2n+1} \frac{1}{n} | \sum_{i=k}^{n+k-1} (x_{i+n} - x_i)| \quad (11)$$

for n = 1, 2, ..., integer part ($\frac{N}{2}$)

To calculate the MAFE, the simple relationship below is used in Equation 12.

$$MAFE(n\tau_0) = MATIE(n\tau_0)/n\tau_0 \quad (12)$$

Same as before, these two metrics can also be used with the minimum packet selection type.

3 Project Methodology

Based on the objectives mentioned previously, the project was split into a few distinct sections:

Data Collection This part of the project involved collecting PTP timing data on the university network. It consisted of using a number of different clock types and locations on the network.

Packet Methods This section involved the implementations of the packet metric scripts based on the recommendation report. Focus on the implementation will be made in this section rather than the metrics themselves.

Calculating and Analysing Results Once the metrics were implemented completely and tested, there was a need of a few supplementary scripts to process this data.

There are some other topics that could be completed time permitting:

Securing PTP Considerations As PTP is inherently an insecure system, there will be some work into investigating how PTP could be secured from rogue users on the network.

Methods to reduce Packet Collisions Due to the way Ethernet networks work, there is a high probability of packet collisions. This is detrimental to PTP as it would increase the time between sync messages. Thus it would be useful to investigate, based on the results collected above, ways to reduce these packet collisions.

4 Data Collection

The first step to perform with this part of the project is to work out what hardware is available. The following hardware was identified:

- Hardware Grandmaster - Chronos TimePort [14]
- Hardware Slave - Chronos Syncwatch [15]
- Hardware Slave - Beaglebone Black [16]
- Software Grandmaster - PTPd
- Software Slave - PTP Daemon (PTPd)

The above were identified to be suitable enough to carry out this project. If any assistance is required with the two Chronos hardware devices it will be possible to contact Chronos directly. The software clocks can be run on any suitable Linux machine.

4.1 Hardware - Timeport

4.1.1 Description

The Chronos CTL4540 Timeport [14] is a low powered portable device that is able to maintain its time to a high accuracy when disconnected from a synchronisation source. It is able to maintain accuracy within a couple hundred nanoseconds on its own. It also has an internal LiPo battery. This enables the device to be used to transport and measure time without using an external power source.

With the above features in mind, it is thus suited for a number of different markets including the power industry and telecommunication network operators. It can also be used to correct for any time errors caused by any cabling or equipment.

Typical methods of doing this would originally have involved using a Caesium atomic clock or setting up a GPS attenna and connecting this to some other equipment. The TimePort is best suited over these two operations because it is much lower power and much more transportable than an atomic clock. It also removes the requirement of relying on GPS.

Figure 4: Chronos TimePort Outside

The difference between the release TimePort and the TimePort that will be used in this project is that the firmware on the TimePort is bleeding edge. With that in mind time needs to be allocated to allow for any issues that the clock may have or to update the firmware if necessary. The university has close links with Chronos thus it should be straightforward to either get any issues solved or to receive a new TimePort. This clock will mainly be kept in the same position on the network and will act as a Grandmaster.

4.1.2 Using the Chronos TimePort

The firmware version that the TimePort was running on did not implement the control port and therefore was not working. The only method to connect to the TimePort is via a USB connection.

To connect to the TimePort: connect the USB cable and run the following command.

```
screen /dev/ttyUSB0 115200, cs8, ixoff
```

Code Extract 1: Bash - Using Screen to connect to TimePort

This command sets the baud rate to 115200, and will connect to the first layer of the TimePort. The system is a restricted linux distribution. A full list of commands and the rest of the documentation can be found in Appendix **??**.

4.2 Hardware - Chronos Syncwatch

4.2.1 Description

The Chronos Syncwatch [15] is a hardware slave clock used to synchronise time in a number of different applications. It operates in all of the current synchronisation technologies such as SyncE, ESMC, PTPv2, 1PPS+TOD, 1PPS, on frequencies(64KHz-200MHz), T1 & E1 protocols and interfaces are supported. It can be used on both legacy and modern Ethernet/IP networks. It can simultaneously operate on a number of the protocols above. It can also operate in both local and remote modes. It is a small 1U rack sized device with a simple user interface. It also integrates with with Symmetricom's TimeMonitor software. The device markets include telecommunications, TV and radio broadcasting, and the power industry.

The table shown in Appendix **??** details the Syncwatch specifications. The figures below shows the outside panel of the device.

Figure 5: Chronos Syncwatch Outside

This product is similar to the TimePort in the fact that it is an engineering release version therefore all of the features are not fully documented.

4.2.2 Using the Chronos Syncwatch

The device can be accessed using either ssh, serial, or through the Syncwatch-Lab program. As the first two stages are similar, these will be discussed as one step.

Figure 6: USB to Serial Converter

To connect via ssh: log in using a terminal program using the following command:

```
ssh root@eepc-rjw-syncwatch.bath.ac.uk
```

Code Extract 2: SSH Command to Connect to the Syncwatch

using password: *syncwatch* .

This is the first Syncwatch layer. The alternative method to access this same terminal window is by using a USB to Serial converter as pictured earlier Figure 6.

Plug in the device to a computer, and connect to it using the following command:

```
screen /dev/ttyUSB0 115200, cs8, ixoff
```

Code Extract 3: Screen Command to Connect to the Device

As before, the command sets the baudrate to 115200, and connects to the device using screen.

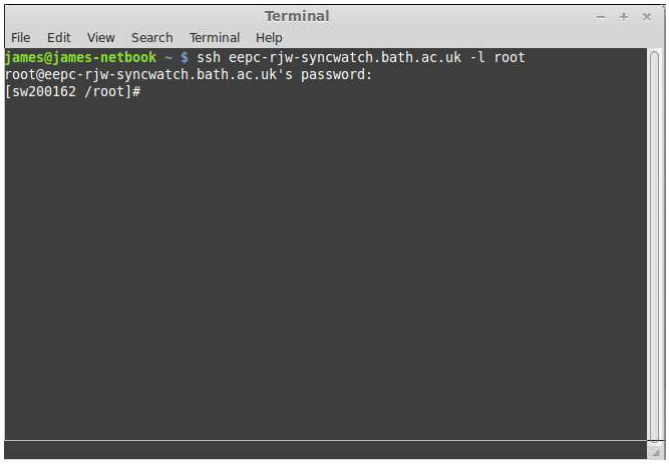


Figure 7: Syncwatch First Layer Screen

To access the PTP console, type the following:

```
minicom -S
```

Code Extract 4: Command to Access the PTP Console

This then brings up the second layer: the PTP console. The screenshot below (Figure 8 shows a list of commands available. The complete list of commands in both of the modes is shown in the documentation, shown in Appendix 4.

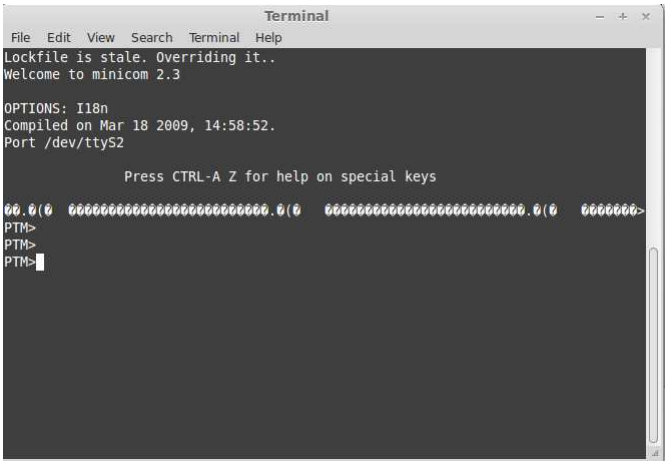


Figure 8: Syncwatch Second Layer Screen

Note that there is some serial corruption visible in this screenshot. This was addressed at the end of the report.

4.2.3 Hardware - Beaglebone Black

4.2.4 Description

The Beaglebone Black [16] is a hardware device which runs a restricted linux distribution. The PTP software daemon will run on top of this operating system. Throughout this report the Beaglebone Black will be called a hardware clock, but in reality it is running a PTP software implementation.

The Beaglebone Black is a cheap development platform that runs Linux. In terms of hardware capabilities it has an ARM Cortex A-8 processor with 512MB of DDR3 RAM. It runs a cut down version of Linux called Angstrom Linux. It has Ethernet connectivity and runs off of a 5V DC supply.

An SSH server has been set up on it with a static IP address. This made it easy to start the PTP daemon.

Below (Figure 9 [17]) is a labelled picture of the BeagleBone Black.

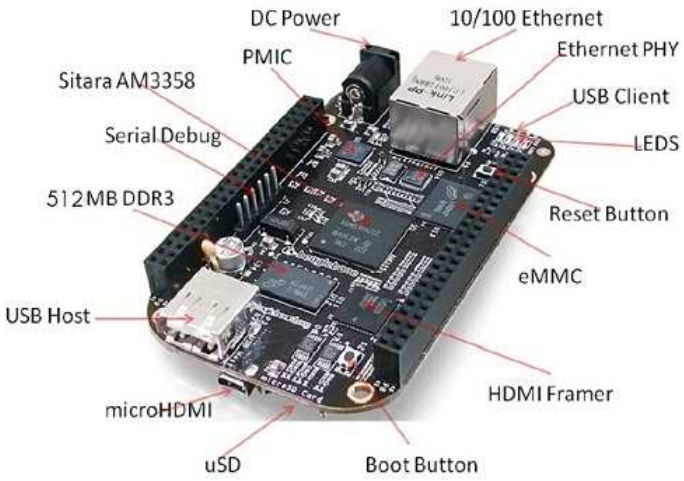


Figure 9: Labelled BeagleBone Black

The Beaglebone will be a useful device to use as a slave clock because of its portability. It would be able to be placed anywhere on the network without any disruption to that particular lecture room or lab space.

4.2.5 Using the Beaglebone Black

To set up the Beaglebone Black:

1. Plug in the Beaglebone Black to the 5V adapter.
2. Plug in the Ethernet cable
3. Once the Beaglebone boots you can then access the device over SSH.

Type:

```
ssh <username>@eepc-rjw-beaglebone.bath.ac.uk
```

Code Extract 5: SSH command to connect to the Beaglebone Black

To log in, replace < *username* > with the username on the device.

The screenshot below (Figure 10) demonstrates this. The ls command was typed to show that the connection was successful.

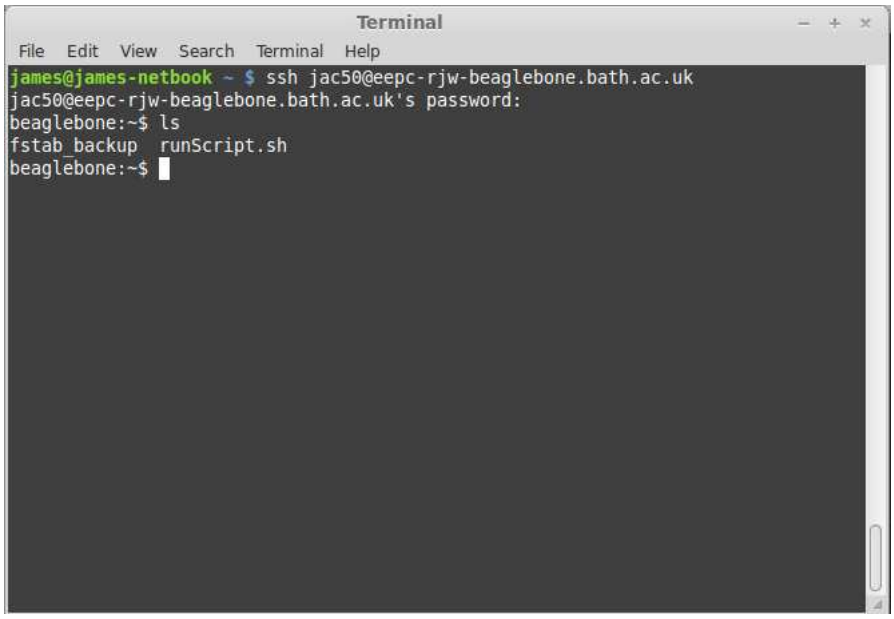


Figure 10: SSH to Beaglebone

Once SSH'd into the device, then the device can be accessed like any other linux machine. Note however that there is a restricted command set. When the BeagleBone boots, it was required that the SD card used to store the test data would be automounted and that the PTP daemon automatically runs. Several attempts in trying to automount the SD card using conventional means such as adding in an entry to fstab were attempted, but these did not work.

The method in getting round this was by creating a script in /etc/init.d. Any script located in that folder will automatically be loaded once the device boots. The PTP daemon was also run from this same script. The script would also need to automatically name the data file or the data would be overwritten every time the device was turned on. A convention of *timeport_YYYY_MM_DD.txt* was decided.

The full bash script is shown below, in Code Extract 6.

```
1 #!/bin/bash
2 mount /dev/mmcblk1p1 /mnt/sd
3 date=\$(date +"%Y_%m_%d")
4 sudo /home/eesrjw/ptpd2 -i eth0 -C -S -g -d 17 -V > /mnt/sd/eesrjw/timeport\_\$date.txt
```

Code Extract 6: Bash Script in init.d for Beaglebone Black

Line 1 is the bash shebang which lets the operating system know that the following script is written in bash. The second line mounts the SD card to the correct location. In this case the SD card is device *mmcblk*1*p*1, and the mount location is /mnt/sd/. The third line defines a date variable in Year_Month_Day format. The forth line runs the PTPd2 daemon. As the script is not saved in the PATH variable, the full path to the script is used. The flags will be discussed in more detail in the later section as similar flags will be used.

Once the script above is run (or if the PTPd2 script is run on its own from the terminal), the output is stored in the text file mentioned on Line 4.

Once the test is completed, the script can be killed by using *kill* − 9. The final step is to transfer the text file from the beaglebone to the local machine ready for packet metrics to be run on it.

4.2.6 Sending Data to the Local Machine

There are two ways to retrieve the data from the Beaglebone: either pulling out the SD card and using an SD card reader to transfer the text file, or remotely using a utility such as *rsync*.

It was decided that as all other commands are sent to the device remotely, that a short rsync script will be made. Code Extract 7 below is the rsync script used.

```
1 #!/bin/bash
2 echo "RSync List-only will run"
3 rsync --list-only jac50@eepc-rjw-beaglebone.bath.ac.uk:/mnt/sd/eesrjw/ ./
4 echo "Type filename here: "
5 read fileName
6 rsync -v --progress jac50@eepc-rjw-beaglebone.bath.ac.uk:/mnt/sd/eesrjw/i\$fileName ./
    NotSorted
```

Code Extract 7: Rsync Script

The script above prompts the user to type in the filename. It lists the files in the correct directory on the beaglebone in case the user does not know the correct file name. Line 6 then performs the sync operation, using the verbose and the progress flag.

The only issue with this script is that it prompts the user twice for the password. As this script was not run very often it was not an issue. If it was however more research would have been done to see if that could be fixed. A screenshot below (Figure 11 shows the rsync script transferring across a gzipped data file.

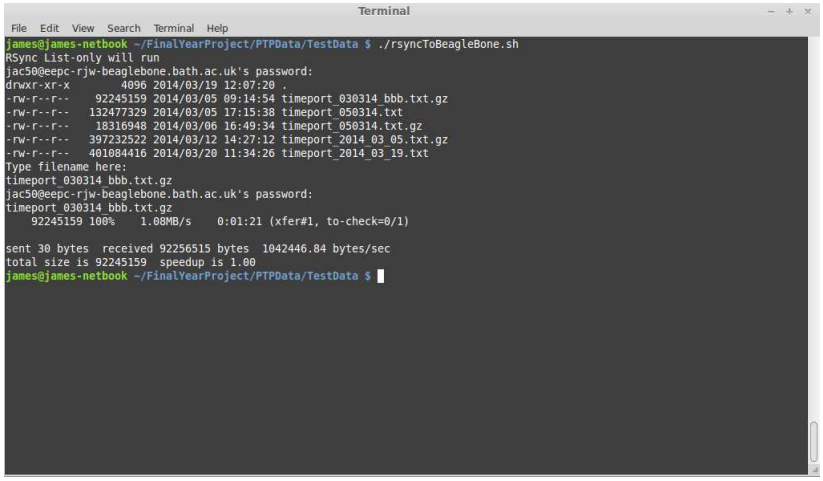


Figure 11: Rsync Example

It was important to gzip the file beforehand or the transfer would have taken a lot longer. The rate of data collection is around 10MB an hour.

Due to the expected size of some of the test files, rsync may not be a suitable method in some cases. Therefore an SD card reader was also sourced for use during the project.

4.3 Software - PTP Daemon (PTPd)

The final type of clock that can be used is a software daemon called PTPd (or sometimes PTPd2). It is a program written in C that meets most aspects of the IEEE1588 specification. PTPd2 meets the changes made in the 2008 standard.

The available flags that may be used for this project will be tabulated below (Table 2). The new and old flags correspond to after V 2.3.0 or before V 2.3.0 respectively.

Table 2: Flags used for PTPd

Flag Name	Description	New Flag	Old Flag
Interface	Network Interface to use	i	b
Domain	PTP Domain Number	d	i
Foreground	Run program in foreground	C	C
Verbose	Run in Verbose mode	V	None
No Clock Adjust	Do not adjust the local clock	n	t
Slave only mode	Set PTPd as Slave	s	g

PTPd can be used as both a slave and a grandmaster. As there is already a dedicated grandmaster, PTPd will be used mainly as a slave clock. The script call has already been given for the beaglebone. The code extract below has been used for the PTPd_Netbook.

```
1 ./ptpd2 -C -S -g -i 17 -t | tee /home/james/FinalYearProject/PTPData/TestData/TimePort-
    To-Soft-Test4/RawData.txt
```

Code Extract 8: Running the PTPd Script

The difference in the call to ptpd2 above is that the command *tee* has also been used so the data is displayed both on the screen and sent to the text file. An alternative method to this would be to redirect the standard output to the text file, then call $tail -f file\_name\_here$ to display the file in the terminal. This script can be

run on any linux computer with root access and the script saved in the current working directory. Note that the above script in Extract 8 is already run as a root user.

4.4 Data Collection Overview

As the majority of the devices above will be controlled remotely via SSH, it would be useful for all of them to be on static IPs assigned by the university computing services. All devices were able to get a static IP with a domain forwarding in the format eepc-rjw-¡nameofdevice¿.bath.ac.uk replacing ¡nameofdevice with the IP name in the table below (Table 3). This is a local address which was not forwarded outside of the university network. The summary table below shows what hardware is available, based on class, and IPs for the PTP port and the control port.

Table 3: Hardware Summary

Clock ID	Name	Type	PTP IP	Control IP	MAC Address
001	Chronos TimePort	Hardware GM	TimePort	USB over Serial	Unknown
002	Chronos SyncWatch	Hardware	syncwatchptp	syncwatch	00:16:C0:17:20:A1
003	BeagleBone Black	Hardware	beaglebone	beaglebone	Unknown
004	PTPd_Desktop	Software	Tesla	N/A	N/A
005	PTPd_Netbook	Software	N/A	N/A	e8:9a:8f:97:64:13

The clock ID was used with internal documentation to know which clock was used where. Note that the domains listed in the table are just part of the full domain name. To access one of them on the university network, add the prefix eepc-rjw- and the suffix .bath.ac.uk. For example, to access the Beaglebone the address is eepc-rjw-beaglebone.bath.ac.uk.

4.5 Locations for Clocks

To get a varied set of data points, it was decided to collect data at a number of locations throughout the network. The following locations were identified, along with some information on the surroundings.

Table 4: Clock Locations

Clock Location	Room number	Room Type	Left Unattended	Distance from Grandmaster
Watson's Office	2E 4.6	Office	Locked office	Same room
2nd floor lab	2E 2.10	Lab	Secure lab	Same subnet
Comms Lab	2E 3.14	Lab	Secure	Same subnet
Library	Library	Library	No. Busy 24/7	Different subnet
8W Rooms	Any Room	Lecture room	No	Different subnet
East building	Any Room	Lecture room	No	Different subnet

The locations above are only examples of possible locations. Due to getting access to rooms or the possibility of PTP not been transmitted out of the particular subnet it may not be possible to use some of the rooms listed above.

It was found to be difficult to collect data from some areas of the university. This was due to the unavailability of Ethernet docking ports.

A broad range of locations were attempted, within the limitations of the network. Connecting via a VPN was attempted but it was deemed that the PTP packets were not transmitted outside of the network and thus were not suited for the job.

A network topology map was required for this project to be able to correlate how many switches the PTP data is likely to be travelling through based on the grandmaster and slave positions. Unfortunately they were unable to be sourced, so a rough estimate was calculated below.

When using the communications lab for both the grandmaster and the slave (2E3.14), it was estimated that the data would travel though 7 total switches.

4.6 Test Sheets

As there were several tests performed during the project, and it is important to note times and locations of each test, a test sheet has been created using LibreOffice Calc. An example of a test sheet is found in Appendix 5.

Each test sheet will have the following:

Test ID Each test gets a unique ID number. The number increments for every test performed.

Test Name A general name for the test. This usually consists of the GM clock type, the slave clock type, and the number associated with that type of test.

Test Date The date at which the test was performed in ISO 8601 format.

File Name The file name for the test file. If the test has to be stopped for any reason, a new file is made with a number at the end. The file name is typically RawData.txt.

Directory The directory where the test data is stored.

Clock Type Each clock will have its clock type listed. The clock types have been mentioned earlier in this report.

Clock Name The name of the clock. This is a unique identifier in case multiple clocks of the same type and model are used.

Clock Model The model of the clock.

Start Time The time that the test started. This is as accurate as possible so network data can be correlated with it.

End Time The time that the test finishes. If the test is stopped prematurely but started up again, the final end time is noted here, but the intermediate start and stop time is listed in the comments section.

Network Activity An average network activity for the day (low, medium, high). This is used to correlate delay spread with network activity.

Test Description Brief description of why the test was performed and what the expected outcome of the test is.

Comments Any comments can be noted here. Start/Stop times, or if any issues arise were noted.

All of the test sheets will not be shown in this report, but the information has been compiled into a summary test sheet. The summary sheet (Appendix 6) will include the Test name, date, directory, and start and end times of the tests.

4.7 Testing Schedule

This part of the project will run in parallel with the implementation stage of the packet metrics, as this does not rely on them being completed. The tests that are to be completed will include:

- Hardware to Hardware
- Hardware to Software
- Hardware to Beaglebone
- Different locations
- Different Times

An explicit testing schedule has not been produced but the full list of tests that would have been suitable to be completed have been listed below. Instead there are week blocks allocated in the gantt chart for data collection, and tests will be carried out throughout that time.

The following tests that have been identified as important tests to run have been included in the table below.

Table 5: Some Tests to Run

GM Clock Type	GM Clock Location	Slave Clock Type	Slave Clock Location	Duration
Hardware (TimePort)	Watson's Office	Hardware (Syncwatch)	2E 3.14 lab	24 hours
Hardware (TimePort)	Watson's Office	Software (PTPd)	Watson's Office	24 hours
Hardware (TimePort)	Watson's Office	Software (PTPd)	2E 2.10 Lab	24 hours
Hardware (TimePort)	Watson's Office	Beaglebone	Anywhere available	24 hours

Any other tests may also be performed, including but not limited to: different grandmaster clock types, different time of day, or different durations.

4.8 Data Processing

The final section of this part of the project consisted of initially processing the data that is gathered from the various clocks mentioned above. This is important because certain parts of the data is only needed for certain metrics and can thus be reduced to reduce the overall memory usage of the scripts.

4.8.1 Example Data File

An example file output for the PTPd implementation which is run on the majority of the clocks is shown below in Figure 12.

```
james@james-netbook ~/FinalYearProject/PTPData/TestData $ tail -n 50 TimePort-To-Soft-Test2/RawData.txt
2014-02-28 13:22:53.562901, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011554454,  0.000000000,  0.011558941, 512000, S
2014-02-28 13:22:53.594550, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011565167,  0.000000000,  0.011571394, 512000, S
2014-02-28 13:22:53.626307, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011598558,  0.000000000,  0.011625722, 512000, S
2014-02-28 13:22:53.658092, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011640084,  0.000000000,  0.011654446, 512000, S
2014-02-28 13:22:53.689762, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011638269,  0.000000000,  0.011622092, 512000, S
2014-02-28 13:22:53.721708, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011626357,  0.000000000,  0.011630622, 512000, S
2014-02-28 13:22:53.753292, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011639041,  0.000000000,  0.011647460, 512000, S
2014-02-28 13:22:53.784978, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011595722,  0.000000000,  0.011543984, 512000, S
2014-02-28 13:22:53.816714, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011546574,  0.000000000,  0.011549164, 512000, S
2014-02-28 13:22:53.848550, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011600674,  0.000000000,  0.011652185, 512000, S
2014-02-28 13:22:53.880333, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011605194,  0.000000000,  0.011558204, 512000, S
2014-02-28 13:22:53.912177, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011597211,  0.000000000,  0.011636218, 512000, S
2014-02-28 13:22:53.943757, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011632345,  0.000000000,  0.011628473, 512000, S
2014-02-28 13:22:53.975501, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011639313,  0.000000000,  0.011650155, 512000, S
2014-02-28 13:22:54.007245, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011651499,  0.000000000,  0.011652845, 512000, S
2014-02-28 13:22:54.039153, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011658933,  0.000000000,  0.011665022, 512000, S
2014-02-28 13:22:54.070672, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011606767,  0.000000000,  0.011548513, 512000, S
2014-02-28 13:22:54.102527, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011609060,  0.000000000,  0.011669608, 512000, S
2014-02-28 13:22:54.134214, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011661959,  0.000000000,  0.011654310, 512000, S
2014-02-28 13:22:54.165891, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011600727,  0.000000000,  0.011547145, 512000, S
2014-02-28 13:22:54.197867, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011598488,  0.000000000,  0.011649832, 512000, S
2014-02-28 13:22:54.229375, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011600384,  0.000000000,  0.011550936, 512000, S
2014-02-28 13:22:54.261190, slv fcaf6afffe00122b(unknown)/01,  0.000000000,  0.011591978,  0.000000000,  0.011633021, 512000, S
```

Figure 12: Example of the File Output

This data is formatted as a Comma Separated Variable (CSV) filetype of the following format:

Timestamp The timestamp includes the big-endian date format specified under ISO 8601 as well as the time. The time is displayed in an HH:MM:SS.ssssss type format.

State The state refers to what clock type is being run.For example a Grandmaster or a Slave.

Clock ID The clock ID is a hardware ID which is specified under IEEE1588.

One Way Delay The one way delay is the average of the Master to Slave and Slave to Master delays. [18].

Offset from Master This is the difference between the master clock timestamp and the current slave clock timestamp.

Slave to Master This is the propagation delay from Slave to Master.

Master to Slave This is the propagation delay from Master to Slave.

Drift This refers to the drift of the slave clock relative to the grandmaster. It is unclear exactly what this number is though since it is an integer.

Last Packet Received This is a single character represents the received packet type. For example, S is a sync packet, and D is a delay packet.

4.8.2 Script to Parse the Data File

It can be seen that depending on what calculation is performed, there is more data gathered than what was required. Assuming that there will be 32 of these messages every second, the memory requirement will be high if a long test was performed. Therefore it was decided to create a script that would parse this data into a correct form for use later on. Even with the data parsed, there may be too many data points. Thus a method to average the data by an integer value will also be added into this script.

As this was a preprocessing step, it was important to try and keep the execution time of this script as low as possible. Due to the script having to operate on a line by line basis, it was decided to use the scripting language Awk [19].

Awk has several advantages over other similar tools, but its main strength is that it is a very useful and efficient tool in parsing rows and columns of log files citeawkAdvantages. It can perform operations on a line by line basis, or when certain conditions are met (for example every N lines, or at the start and end of the file). As it is used for file parsing, it has a built in regular expression engine which is handy for string manipulations. Its simple structure makes it a suitable tool for the job. It was also identified to be a useful tool to know in the future, and thus it was decided to use awk for this part of the project.

Flow Chart
The script needs to be able to read in the text file and save the correct columns to a new file. If appropriate the script should also average N number of data points and save these to the new input file instead. The time field must also be parsed correctly and the time delta added to the new file.

The general script layout has been converted to a flow chart, see below Figure 13.

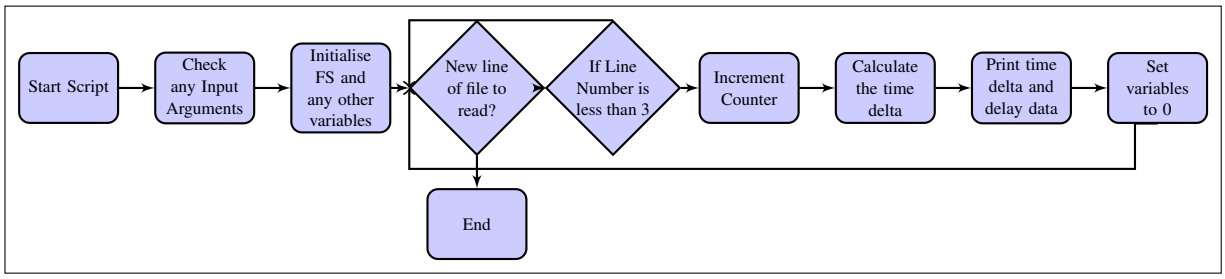


Figure 13: Awk Script Flow Chart

This flowchart was then used to create the awk script. Most of the code is self documenting, but the *add_time* will be explained as that was part of the code that required some extra work to get working correctly. The script can be found in Appendix 7.

add_time function
The role of this function was to amalgamate the times from the first sample to the Nth sample, with N being the size of the window. The main issue with this function was the formatting of the time itself as it was not either an ISO 8601 float nor in the correct format for an inbuilt function to be called. Therefore the following steps were performed to convert the time to the correct format.

1. **Regular Expressions (Regex):** Before the function is called, a regular expression substitution is used to replace all colons with spaces and all dashes to spaces. This was important because the end goal is to convert the current time format into one seperate by spaces.
2. **Split Seconds portion by full stop:** This step would seperate the milliseconds away from the seconds.
3. **Compare the delta times to see if one was bigger than the other:** This was important so the correct sign of the delay was used.
4. **Calculate delta:** This is used to determine the size of the time deltas.

This script was tested thoroughly to make sure all of the cases were covered, but this was not fully documented.

Conclusion to the Script The script runs very quickly, primarily due to the correct choice in language early on. Any testing for this script was carried out before it was integrated into any other source file. It was important to try and keep this script standalone, so any data file can be parsed independently without needing to call an extra program.

The following table shows some execution times for this part of the script given the number of lines run on a netbook. The above results were taking with a 10point average.

Table 6: Execution Times for Awk Script

Number of Lines	Time (ms)
1000	7
10000	28
100000	1384
1000000	14145

Based on the table above, the script is suitable enough considering that this will only be run once then the file is saved.

5 Packet Methodology

This section of the project is the bulk of the programming development. Packet metrics will be created in a suitable language and will be run against the data collected in the previous section.

A range of packet metrics were chosen to be implemented from the report detailed in the literature review section of this report. The following packet metrics will be implemented: TDEV, minTDEV, percentileTDEV, bandTDEV, MATIE, and MAFE. Note that both MATIE and MAFE can have different packet selection methods (min, percentile, or band), so there may be more than the above implemented in the final script.

5.1 Choosing a Suitable Language

The first decision to make for this section of the project was to choose a suitable programming language. Based on the languages that would be suitable for a task such as this, the following languages were identified: R, C, Matlab, or Python.

The requirements that the language must meet in order to be suitable for the project are listed below.

REQ1- Familiarity with the Language

Spec: Used for a sufficient length of time

If the language was very familiar the development time of the scripts would be quicker. This extra time taken to learn the language may be acceptable however if there is a much better language suited for the task.

REQ2 - Well Documented

Spec: Not Applicable

The majority of modern high level languages are well documented, with some online resources better than others. The language must be well documented. This will also make it easier if **[REQ1]** has not been met fully as it would be easier to learn the language with good documentation.

REQ3 - Plotting Functionality

Spec: Sufficient plotting functionality available to meet the task

The language must have enough functionality to be able to plot the data gathered in several different forms. The plots must be easily exportable to a suitable format to then be attached to this report.

REQ4 - External libraries already available

Spec: All required libraries available

Some external libraries may be needed in case some specific functionality is required. Examples of external libraries that were required are a command line argument tool and logging functionality.

REQ5 - Speed

Spec: Performs the metrics in a reasonable length of time

The metrics should be able to run on a relatively large dataset in a reasonable length of time. As it is unknown how long the scripts will take, this reasonable length of time will be decided later. If need be optimisation can be made to make the scripts faster.

REQ6 - Linux Compatibility

Spec: Can be developed under Linux

As the rest of the development will be using a Linux Mint netbook, it is a preference for the language to be suitable for a Linux development environment.

To decide on the best solution, a set of ranking criteria was created as well as a ranking table. The table below shows the criteria that the above languages were compared against.

Table 7: List of Criteria for the Language Options

Criterion	Description	Requirement	Weight	Highest - 5	Lowest - 0
Familiarity with the Language	Is the engineer familiar with the language syntax and style?	**[REQ1]**	9	Developed a few large projects.	No familiarity
Plotting functionality	What plotting functionality is available for the language?	**[REQ3]**	8	Lots of plotting functionality	All plotting functions would need to be written from scratch
External Libraries available	Are all of the libraries available to complete the project?	**[REQ4]**	7	All of the required libraries are available.	Minimal library support.
Speed	Is the chosen language going to be fast enough for the application?	**[REQ5]**	7	Fast enough.	Not fast at all. Needs careful programming to make as efficient as possible.
Linux Compatibility	Is the language compatible in a linux development environment?	**[REQ6]**	7	Yes, it is compatible.	No. Windows Only
Development Time	How long would it take to develop a working example?	**None**	7	Less than a month	Longer than 3 months
Documentation	Is the language mature enough to have a full set of documentation?	**[REQ2]**	6	Yes. The language has clear and concise for all of the documentation.	Very limited.

Once the requirements were decided upon and the criteria set, a ranking table was produced. With the assigned weights, each language choice was ranked from one to five. In the table below the weight was multiplied by the rank to get the correct value. These were then added up to get a total figure of merit.

Table 8: Language Options Ranking Tables

Programming Language Decision					
Ranking Criteria	Weight	Language			
		R	Matlab	C	Python
Familiarity with the Language	9	9	36	45	27
Plotting Functionality	8	40	32	0	32
External Libraries Available	7	35	21	14	21
Speed	7	14	14	35	7
Linux Compatibility	7	35	21	35	35
Development Time	7	21	28	14	21
Documentation	6	30	24	12	24
Total Figure of Merit		**184**	176	155	167

Therefore based on the ranking table above it was decided that R will be the most suitable language for the task. In addition to this there are some R scripts as part of the PTPd which may also be used if suitable.

R is a programming language that used primarily for statistical computing. It is similar to the S programming language but under the GNU's Not Unix (GNU) umbrella. R has many strengths in statistics due to the wide range of libraries available to it. In addition to this it has extensive plotting functionality, both as standard and those written by third parties. For computationally difficult tasks functions can also be ported into C, C++, or Fortran. If this needs to be considered then C will be chosen as it is the more familiar language compared with the others.

The development environment used for this project will be between a Linux Mint netbook and a virtual machine running Debian. A standard text editor (vim) and R will be used to create the scripts, rather than using a full Integrated Development Environment (IDE) such as R-Studio. Note that the code will be written so it can run cross platform, provided that the necessary libraries have been installed.

5.2 General Packet Metric Implementation

The packet metric literature review section of this report (Section 2.1) outlines the equations used to calculate the metrics. It can be seen from these different metrics that there is a common format for all of them, and this will be exploited where possible.

Each of the packet types will be performed in the smallest number of for loops as possible to cut down on execution time. The general format for the packet metric script in a flowchart is the following (Figure **??**).

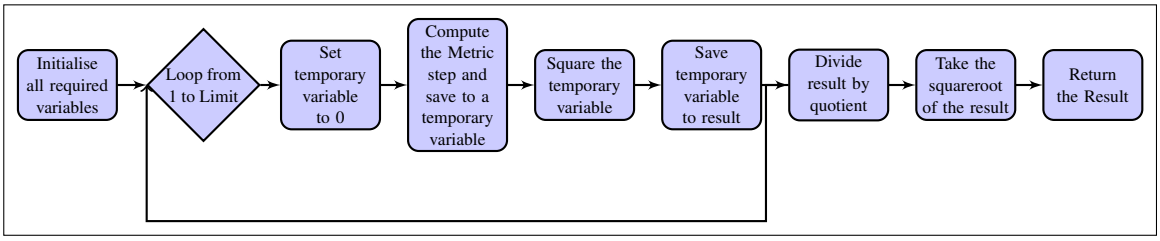


Figure 14: General Packet Metric Flowchart

The flow chart above has been converted into psuedo code which is also based on Equation 8.

```
TDEVAllMethods(To , n , N , x ,a ,b)
Data: window /* Sets the windowSize                                    */
    InterimStep
    X
Result: result
Initialise all required variables to 0;
for 1 to end of summation do Loop from 1 to the end of the summation
    Set inner step value to 0
    Calculate Metric;
    interimStep = interimStep * interimStep
    result += interimStep
end
result = result / (6 * N - (3*n) + 1)
result = sqrt(result)
return(result)
```

Algorithm 1: Psuedo-Code for General Packet Metric Script

The for loop length is determined by the value of N (the number of samples) and n (the current iteration in the set). Note that the algorithm designed above only calculates one result for the particular packet metric. This

function would have to be called continuously with an incrementing value of n up to the limit. This limit is determined by the particular metric. The limit is $\frac{N}{3}$ for TDEV based metrics and $\frac{N}{2}$ for MATIE or MAFE metrics.

Because the TDEV and MATIE packet metrics have different looping conditions, they will be created in two seperate scripts.

5.3 TDEV and TDEV derivatives

Using the above flow chart as a base for the script, the TDEV and derivative functions were created. All of the TDEV scripts will be created in one source file in R.

The four metrics that were implemented were: TDEV, MinTDEV, BandMeanTDEV and PercentileTDEV.

TDEV The TDEV script requires that mean values are taken of each individual window. As there is a built in mean function to R, this will be used. Optimisation of this mean algorithm will not be looked into.

minTDEV Same as above, as a minimum function exists in R, this function will not be written directly.

bandMeanTDEV A band mean is similar to a mean but is taken over a particular set of values of the window. For example, if the band mean was taken between 20 and 80%, then the mean will use the middle 60% of values. As this function does not exist in R, this would have to be created.

The bandMean implementation can be found below, in Code Listing 12. This can also be found in Appendix 8.

```
#!/usr/bin/env Rscript
#-------------------------------------------------------
#-------------------------------------------------------
#--                 Function Name: bandMean           --
#--                 Name: Band Mean                   --
#--           Input: window - the samples             --
#--                        a   - lower band           --
#--                        b   - upper band           --
#--           Output : mean of window                 --
#-------------------------------------------------------
#-------------------------------------------------------
bandMean <- function(window,a,b){
  sum <- 0
  a <- (a / 100) * length(window)
  b <- (b / 100) * length(window)
  a = round(a) + 1 # 1 indexed
  b = round(b)

  for (i in a:b){
    sum <- sum + window[i] # sum window from a to b
  }
  average <- sum / (b - a + 1)
  return (average)
}
```

Code Extract 9: Band Mean Implementation

The script converts the bands a and b into an integer. If the window size was 5, and a was 0 and b was 100 (i.e the total window size), then a would be 1 and b would be 5. The sum is the computed by looping from a to b. The average is then taken by divided the sum by $b - a + 1$.

percentileTDEV The percentileTDEV is very similar to the bandTDEV, except that the lower band is forced to 0.

With these four packet types, and referring to the general form of the script in the previous section, the complete TDEV script can be produced. See Appendix 9 for the full implementation of the script in R.

5.4 MATIE and MAFE derivatives

The MATIE and MAFE derivatives were created in a similar method to the TDEV scripts above, but the main difference is both for loops have different upper bounds, hence why a different script needed to be created. The following MATIE and MAFE packet metrics were used: MATIE, MinMATIE, MAFE, and MinMAFE.

MATIE This is implemented in a similar fashion to TDEV, using the built in mean function.

minMATIE The built in minimum function was used for this metric, in a similar method to the minTDEV script.

MAFE As described previously, MAFE is very similar to MATIE but there is a scaling value. Therefore only 2 values will be produced in this function, then MAFE will be calculated at the end before the values are returned.

minMAFE This will be calculated based off of the minMATIE result.

The MATIE/MAFE script was then created, which can be found in Appendix 10.

5.5 Overall Packet Script

Once the packet metric functions were created, these will need to be both called from a separate file which will handle file IO and plotting. The following functionality that the overall script should perform:

Input Arguments For the script to be fully functional, the script must be able to accept input arguments.

Logging The script may end up taking a substantial length of time to compute the packet metrics. Therefore some functionality of logging must be built in to this script. The logs may also be saved to a text file if needed.

Writing to a File To save from calculating the same packet metric data for a particular data set, the results can be saved to a text file.

Plotting The script must have some plotting functionality, as that is how the data will be displayed.

5.5.1 Flow Chart

The general layout of the script will be discussed, then each relevant function will then be discussed in turn. The flow chart for this script can be seen below, in Figure 15. With the following flowchart, there are some decisions to be made on what libraries to use and how to go about implementing certain functions.

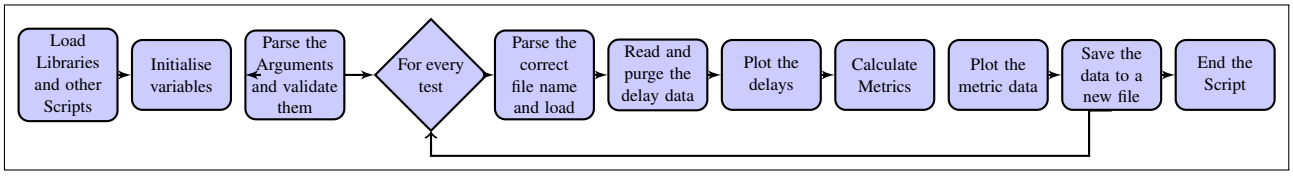


Figure 15: Overall Packet Script Flow Chart

5.5.2 Input Arguments

It was intended for this packet metric script to be fairly comprehensive. Thus it required input arguments so the user can choose what sort of metric to perform or on which set of data. The input argument functionality for R is similar in some respects with C: it only consists of positional arguments and uses an argc / argv type structure. It was more suitable to use a more substantial input argument library such that optional flags can also be used.

Argparse [20], a library originally written for Python but ported to R, was the most suitable choice. It has numerous useful features such as it generates a help list automatically and allows for the use of optional flags.

The next decision would be to see what arguments would be required. The following table outlines what arguments were used.

Table 9: List of Arguments

Argument Name	Description	Argument Flag	Type	Default Value	Destination
Test Number	Which test number should be used?	-nTest	Integer	- 1	nTest
Number of Lines	How many samples should be used?	-nLines	Integer	0	nLines
Directory	What is the test directory?	-directory	String	None	directory
Metric	What metric type do you want to run?	-metric	String	TDEV	metrics
Direction of Delay	Do you want Master to Slave or Slave to Master?	-delayDir	String	Master2Slave	direction
CSV File	Do you want to save the data in a CSV file?	-CSV	Boolean	None	CSV
Verbose Mode	Do you want the logging to be verbose?	-v –verbose	Boolean	None	verbose
Quiet Mode	Do you want the logging to be quiet?	-q –quiet	Boolean	None	quiet
Histogram	Do you want a delay histogram saved?	–hist	Boolean	None	hist
Colour Histogram	Do you want a colour delay histogram saved?	–cHist	Boolean	None	cHist
Delay Plot	Do you want the delay plotted?	–plotDelay	Boolean	None	pdelay
Statistics	Do you want to generate a table of Stats?	–stats	Boolean	None	stats
Starting Point	Set the starting point of the test data	–start	Integer	0	start
Interactive Mode	Enable Interactive Mode	-i	Boolean	None	interactiveMode

All of the arguments above have been implemented as optional, but there is some logic to work out which ones are required.
It was handled by both the main script and a few helper functions.

5.5.3 Logging

Due to the large number of functions and steps that the script has to perform (and possibly a long execution time), it was paramount that a suitable logging system was set up such that any issues can be tracked. This can also be used so the user can know which section the script is currently working on. A simple custom made

system could have been made which just had different print messages depending on the logging level, but due to range of 3^{rd} party libraries available for R, a logging library was used.

Similar with ArgParse, the logging library [21] was based off of the logging library for Python. The library is thread safe and has a number of different logging levels. One useful feature is that it can have multiple loggers which will direct output to the console and/or a text file depending on the level of the logging entry.

The code extract below (Code Extract 10) details the steps to initialise the logger. Once initialised a logger entry can be written using one of the functions detailed in the documentation.

```
if (args$verbose == TRUE) {
  basicConfig(level=10)
  loginfo("Verbose mode activated")
} else if (args$quiet == TRUE) {
  basicConfig(level=30)
  loginfo("Quiet mode activated")
} else {
  basicConfig(level=20)
}
addHandler(writeToFile, file="LogFiles/Test.Log", level=10)
loginfo("-------------------------------------------------")
loginfo(" ---------- PacketMetric.R Log ------------------")
loginfo("-------------------------------------------------")
```

Code Extract 10: Setting up Logger

Based on the input arguments the logging level is set to either 10 or 30, and a filehandler is added. The following logging messages were used throughout the Packet Metric Script.

Table 10: Logging Entries

Logging Entry Name	Level	Type	Description
Header	Info	General	Header for the Log File
Script Loading	Info	Libraries	Notifies when all libraries have loaded successfully
Init Logger	Info	Logger	Initialises the Logger
Interactive Menu	Info	Interactive Menu	Notifies when the interactive menu has been activated.
Directory / Test No.	Error	Parse Args	Error thrown when no directory or test number is found.
Sample Size Changed	Warning	Parse Args	Warning when the sample size has been changed to 50.
Test Notification	Info	Script	Notification on which test is running
Read File Error	Info	Reading File	Error when reading the correct file. File not exists.
Data Conversion	Info	Reading File	Data File is being converted
Head Running	Info	Reading File	Head was successfully run
File Type Info	Info	Data Parsing	Displays the file version type
Plot Notification	Info	Plotting	Displays the type of plot that was created
Iteration Notification	Info	Metrics	Notifies the user iteration number and time
Run time and Memory	Info	Closing	Notifies the user what the run time and memory usage is

Other logging entries were also added for testing or information purposes which were taken out of the above table. It was important to not have too many logging entries or it would have been difficult to parse both the log file or the console window.

5.6 Function Library

It was a much neater way to port most of the code into functions and save it into a separate script. The implementation details of each of the functions will not be detailed in this section. Instead a short description of each function will be given.

The following is the complete list of functions created.

runHead
: This function is used to trim the original dataset down to the sample size specified. It uses the linux system command *head* and saves the new file into the same directory. There is also a logging command when the command is successful. This function is only called when the required file does not exist.

createArguments
: Due to the large number of arguments required, this function creates all of the arguments, then returns the argument parser to the main script.

initLogger
: This function declares the logger based on a few command line arguments (verbose or quiet). It also adds both a command line and a file logger to the object.

parseFileName
: parseFileName takes in 3 arguments: test number, directory, and the direction of delay (called column in this function). Using the summary test sheet page stored in an .ODS file, this function will take the directory name based on the test number. Some checks will be made to determine whether the directory or test number is valid. A list is then returned consisting of the fileName, the test number, the delay index and the test sheet object.

readFile
: This function attempts to read the file that the previous function had derived based on the input paramenters. If the file is not read successfully an error is thrown. The runHead function is then called to create the correct file, and the file is read again. If a second error is thrown then the script quits with a non-zero return value.

readFileDirect
: Compared with the previous function, this once takes in a raw filename and will exit the script if it fails on the first time round. This function is only used when the user knows which file they want to parse.

purgeData
: Once the data has been written, it will be purged. Depending on the data type (the old or new format), it needs to be handled in two different ways. Therefore an if statement is used to determine which file format it is in, then handled accordingly. The two arrays of delays and time are returned along with the correct number of samples.

plotArray
: This function plots the metrics as a line graph. It saves them in a postscript format, using the test number, metric type and the sample size in the file name. This function can plot *n* number of plots with a different coloured line.

generateResultsArray
: This creates a much larger array called *result* which consists of all of the results. This is then later used to output the results to particular formats.

outputTable
: This function will convert the results table generated by the previous function to a CSV file or a table in LaTeXif requested using the command line arguments. The CSV file is created using a built in function called write.csv, whereas the LaTeXtable is generated using a custom made function.

calculateMetrics
: The different packet metrics are calculated in this function. Note that this is the final packet metric function and thus only contains the most recent changes. The original design had two functions implemented in R, as detailed in a previous section. Any optimisation techniques are discussed in a latter stage of this report.

convertData

Using the parseData awk script described previously, this R function calls this script to parse the original data file into the new format. It firsts checks if the correct directory exists, and creates it if it doesnt using $mkdir - p$. It will then return the path that the file was created in.

interactiveMenu

Finally the last function creates an interactive menu in case the user does not know all of the input argument flags required to run the script.

5.7 Testing

5.7.1 Overview

Throughout the development of these scripts there was testing involved, both formalised unit testing and general development testing of each function. Testing is not just used to determine where bugs lie in the code, as it also helps to identify possible areas for optimisation, determining that all edge cases are covered, and that all input arguments are handled correctly.

General code development will not be discussed in this section because that is part of any programming project. Instead some information on the unit testing performed will be highlighted as well as the major problems that the script faced during its development.

5.7.2 Unit Testing

Unit testing is a formalised method in making sure that when changes are made to a function or a piece of code, that all functions are still working to specification. These are usually combined with building tools, but in the case for R there is a unit testing library available that can be used.

Formalised testing takes a long time to put together to catch all errors when developing a program, thus not as much testing was performed on this script as what should have been done. The following table (Table 11 outlines the list of tests required for each of the functions, with Appendix **??** showing the unit tests created.

Table 11: Unit Tests Identified

Test Name	Function	Type of Test	Pass Value	Fail Value
Sample Size Bounds	runHead	Integer Test	0 Value Max	0 or Max
Sample Size Type	runHead	Type Test	Integer	Not Integer
Arg Parser	createArguments	Library Exist	Library exists	Library does not exist
Init Logger Logical	initLogger	Input	Set to correct mode	Throw error
Init Logger Dir. not exist	initLogger	Directory	Successfully created	Throw error

The tests above are some of the tests performed on the functions detailed above.

5.7.3 Runtime and Memory Requirements

As well as testing to ensure that the script meets the required functionality, it also needs to meet a requirements in that it needs to run in a reasonable amount of time within a suitable memory bound. Defining both of these was difficult as it was hard to predict ahead of time what a reasonable amount of time would be for both of them. It was decided that a time bound for a set of metric data would be **5 minutes** and the memory boundshould be **1GB**. Note that this time refers to the entire script, but some tests may be performed on a per iteration basis.

The script was run using the example data gathered before the project began. It was deduced that there would not be a significant time difference between the data sets provided that the length of it was kept the same.

There are profiling tools available to R, including Rprof , but the simplest method to profile the speed of the script is to record the time when the script is loaded, then record the new time once it has finished. The difference in these two times will be the execution time.

Similarly for memory requirements, Rprof is able to profile this, but instead a more simpler method was used which involved calling $object.size(x = lapply(ls(), get)))$ at the end of the script. An example output can be shown below, Figure 16.

```
2014-04-15 17:11:07 INFO::MATIE-MAFE Plot Created
2014-04-15 17:11:07 INFO::Data written to file
2014-04-15 17:11:07 INFO::Total Run Time:  0.372
2014-04-15 17:11:07 INFO::Total memory requirement:  392424
```

Figure 16: Example Output for Runtime and Memory Requirements

The following table (Table 12) outlines both the runtime and memory requirements for a number of different sample sizes. All of these tests were performed on the same computer.

Table 12: Runtime and Memory Requirements for ExampleData

Sample Size	Per Iteration (ms)	Total Time (ms)	Memory (KB)
50	100	1460	380.064
100	400	11060	392.424
500	15000	-	491.176
1000	120000	-	614.536
5000	-	-	1601.240
10000	-	-	2834.600

The dashes signify that the test took too long to complete and thus no data was calculated for this point. 100 data points only corresponds to around three seconds worth of data which is not enough to be able to determine any pattern to the delay results. Therefore the scripts can either be run on a much faster computer or they can be optimised.

Whilst the first method would speed up the process, there needs to be a considerable increase in speed before this script can be beneficial. It is estimated that there should be at least a four orders of magnitude increase for the script to be usable.

5.8 Optimisation

Based on the above results, it is clear that there needs to be some speed improvement to the script in order for it to be useful. Because the script will not be run on an embedded system, this problem is not bound by memory, therefore only speed optimisations will be made.

It was important to determine which parts of the script were running slowly. It was fairly clear where the slow part of the script lies from the data gathered in the previous section.

The metric calculation (per iteration) took much longer than previously anticipated. The calculation involves a single for loop of different length, but the length varies on the sample size.

The following were a few ideas to speed up this process.

Use Existing Data If possible, only incremental calculations can be made on successive iterations, thus cutting out the number of loops required. When looking at this particular application, because the for loop bounds depend on the current iteration number and the total sample size, it would not be possible. In

addition to this the final quotient consists of the sample size and the current iteration, therefore the value will be incorrect if previous sets of data were taken. This means that it would not be possible to use this method to speed up the operation.

Vectorise The second possible method would be to vectorise the solution. Similar with Matlab, R can be slow with iterative methods and excels at vector operations. The function would speed up significantly if a vectorised method was possible. Due to the use of the built in functions *mean*, *min* and *bandMean*, there is not an obvious way in going about this. Therefore this method would not be suitable.

Port to a lower level language The final method involves porting the for loop into a more lower level language, such as C or Fortran. This would enable the inner part of the for loop to be executed much quicker which would decrease the time of execution.

Based on the above, C was chosen to implement the inner loops for both TDEV and MATIE / MAFE R files. The code was directly ported into C from R, but some extra work was required due to C not having the ability to slice arrays. The amended code in C can be found in Appendix 11 for the TDEV method in C and 12 for the MATIE/MAFE method.

Some care had to be taken when converting the code from C to R. Due to the difference between how C and R handle array indices (C is zero indexed and R is one indexed), the for loops had to be changed by 1. Unfortunately there is no direct way to compare the results due to the likelyhood of floating point differences, so the C code will be taken for granted as being correct. A table below of the result differences between the C code and the R code have been provided.

The speed improvement from making these changes was more than three orders of magnitude. The table below shows the speed improvements that porting the code to C has made.

Table 13: Speed Improvements from moving from R to C

Sample Size	R per iteration (ms)	C per iteration (ms)	R Full Script (ms)	R C full script (ms)
50	100	4	1460	544
100	400	4	11060	644
500	15000	4	-	1640
1000	120000	4	-	4316
5000	-	60	-	199660
10000	-	150	-	706584

As before, the dashes signify that the result were not collected for that test. The results above have not been plotted on line graph due to the missing data points, but it is clear that the C implementations have improved the running of the script by a few orders of magnitude. In addition to this larger data sets can now be used in contrast with the R scripts.

6 Analysis Methodology

6.1 Overview

This section details the analysis methods used on the data gathered. This will include the different plotting types used as well as any other relevant statistics. There are three ways that the data could be displayed or handled to be able to draw conclusions from them. These have been identified to be:

Tables Tables would be the first method to display the data as the raw data can be added to the table. The problem is that it is difficult to infer trends from a large data set, and thus would not be a suitable method for this application. Note however that it may be a good method to take a snapshot of the data and be able to determine some characteristics of the data.

Statistics The second way to process the data is to use some basic statistics in a tabulated form. This allows it to be possible to quite quickly compare sets of data between each other, but it makes it difficult to be able to look within a set of data. Each of the statistic types will be computed using inbuilt functions where possible. The possible statistic types that will be used are: minimum, maximum, mean, median, standard deviation and variance.

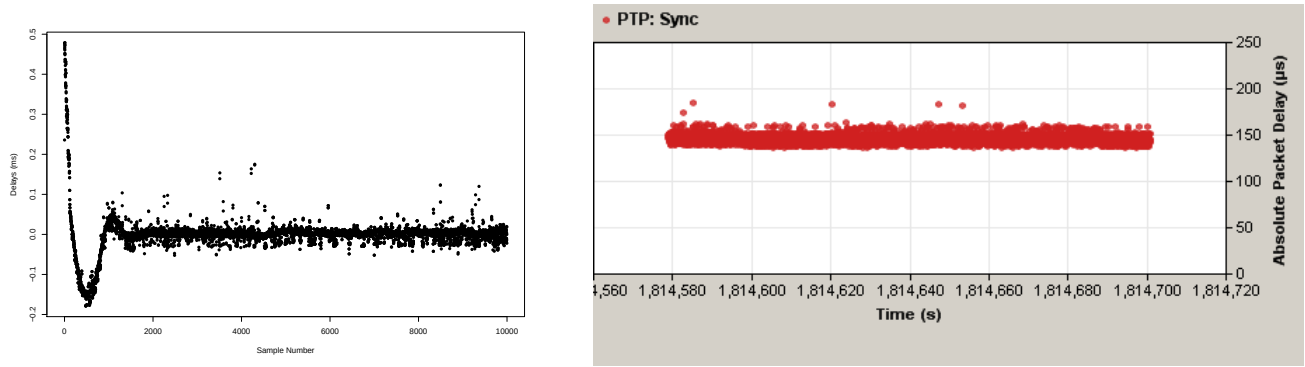
Plotting The final method was to plot the results. Due to there being a large number of possible plot types, this would be a suitable method to choose. The data needs to be displayed in such a way that trends can be spotted quite easily. Therefore the following plot types will be used: scatter plots, histograms, heat maps and line graphs. Each of these will be used for a different set of data, and will be discussed below.

6.2 Plot Types

In order to best display and process the data gathered, several plots need to be created. It is important as to what plot types are used as it can affect the conclusions drawn from them. The plots detailed below were the ones chosen for this project.

6.2.1 Scatter Plots

Scatter plots will be the first type of plot used, mainly on the raw delay data. Provided there are enough data points, it is a useful method to be able to see general trends in delay data and to see if there are any abnormalities. The plot below (Figure 17) shows two different examples of a scatterplot. Figure 17a is one created using an R script, and the second is from the Syncwatch Lab program.



(a) Scatter Plot Example in R

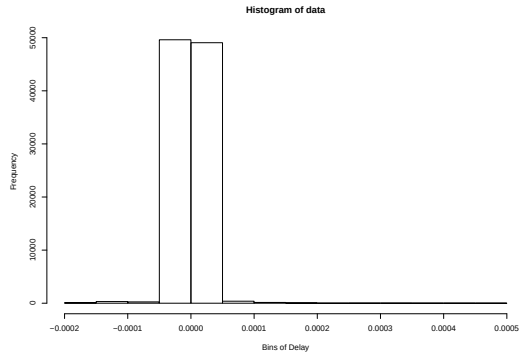
(b) Scatter Plot Example in Syncwatch Lab

Figure 17: Scatter Plot Examples

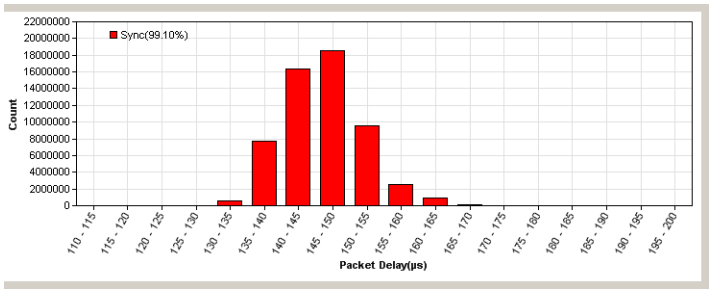
Both of these plots are very similar, with the R plot being more flexible in the fact that you can plot the whole data set instead of part of it. The Syncwatch scatter plot will instead be used to show some anomalies in the results.

6.2.2 Histograms

The second plot type is histograms of the raw delay values. This is another method of displaying the same data as before, but it is more related to the distribution of delays across the entire data set. The figure below (Figure 18) shows the two examples of histogram that will be used.



(a) Histogram Plot Example in R



(b) Histogram Plot Example in Syncwatch Lab

Figure 18: Histogram Plot Examples

The two histogram plots will show similar results, so instead the second histogram plot will be used for the long term data collection as that was generated with the Syncwatch Lab program. It can also be seen that the number of bins selected can have a large effect on the histogram, therefore care needs to be taken when choosing the correct bin size.

6.2.3 Heat Maps

If slices of time are taken, then converted to histograms and flattened to a one-dimensional coloured bar, a heat map of these time slices can be produced. These can be produced in any language, such as Matlab or R. An R implementation has been produced, and an example can be found below in Figure 19.

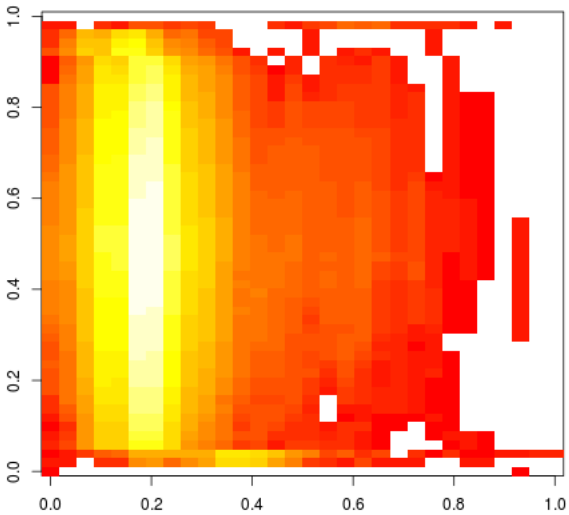


Figure 19: Heat Map Example in R

This is a very pictorial way in showing changes in delay across a set of data. Note that the histogram above was from a data set which had a few zero values in, which causes the white areas on the image. When the results are discussed this image be truncated so only the coloured portions of the image can be analysed.

It can be seen above that whilst the heat map is a useful method, it was not possible to have a complete map due to in some cases there being an NA value. This is due to taken the log value of 0. A Matlab implementation was created by Dr Watson and it would have been beneficial to be able to compare both the R and the Matlab

implementations to see where the differences are. Some heat maps will be created of the data collected, but these may not be used to draw any conclusions from.

6.2.4 Line Graphs

The final method of plotting the data will plot the metric data for both TDEV and MinTDEV. As it is difficult to determine how this data should be plotted, a set of simple line graphs will be produced initially. If a further plot type is required this will be highlighted in the results section. Two plot examples for a line graph are shown below, in Figure 20.

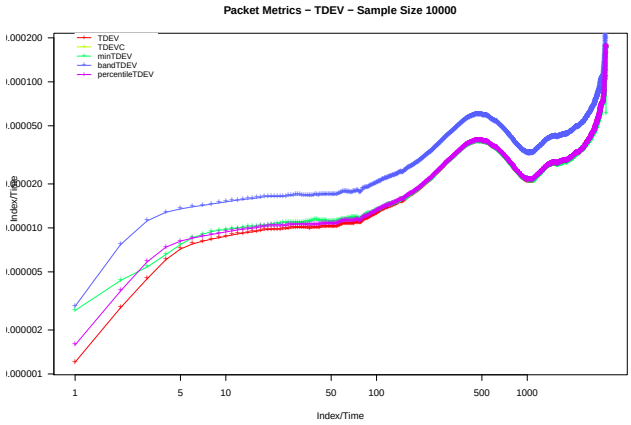


Figure 20: Line Graph Example in R

It can be seen in the figure above that the legend (situated in the top left corner of the plot) is small. It was kept this size so the plot was not too busy when looking at it. Instead the legends will be added to the start of the results section.

6.3 Plotting Implementation

Taking these plots above, these plots can be implemented in R. The correct axis labels, legends (if appropriate), and line colours are also automatically generated. The implementations of the plotting functions will not be explained in much detail, but they can be found in Appendix 15, **??** and **??**.

6.4 Summary of Plot Types

With the plot types mentioned, each of them will be used for a different reason. Firstly the scatter plot was used as an easy way to determine any outliers and to get a snapshot of the delay distribution. It is however more difficult to see the total distribution of points in comparison to a histogram. Therefore that is why histograms were also used to analyse the results.

Thirdly heat maps were very useful to visualise the delay data and to spot periods in the data set where the delay was higher. Note that it was difficult to implement this function, and therefore may not be used extensively.

Finally line graphs were used primarily to display the packet delay data as this is how they were displayed in the recommendation report.

7 Results

This section will detail the results gathered from the above metrics as well as the other testing that was performed. The summary table of all of the tests performed can be found in Appendix 6. The following

elements of PTP will be investigated using these results: Long Term Grandmaster Drift, Delay Distribution, Delay anomalies, Metric Results, Master to Slave vs Slave to Master and Comparing times of Day / locations.

7.1 Long Term Grandmaster Drift

The first set of results gathered was measuring the long term drift of the Chronos TimePort Grandmaster clock with respect to a Rubidium clock available in the lab. The clock used to compare it is a Rubidium oscillator. The full specification sheet has been cited here [22].

The Rubidium clock used is a Rubidium Frequency Standard FS725 made by Stanford Research Systems (SRS). It integrates a rubidium oscillator (model PRS10) with an AC power supply and distribution amplifiers. It is a very stable oscillator with anticipated aging of less than 5×10^{-9} over a period of 20 years. A photo of the Rubidium clock used can be seen below, Figure 21. The datasheet is referenced here on the SRS website [22].

Figure 21: Rubidium Clock

The Chronos TimePort was compared with the Rubidium clock mentioned above and this difference was plotted using the Allan Deviation. The plot was performed using TimeLab, which is software that takes precise frequency and time measurements and then plot them. There are other timing source reference plots available with the TimeLab software.
A counter is used to compare the Rubidium clock pictured above to the TimePort clock, which the TimeLab software then uses to calculate the Allan Deviation. The counter used was an Agilent 53230.

With all of the above, the complete Allan deviation plot is given below, Figure 22.

Figure 22: Allan Deviation Plot with the FS725 reference

The light blue thick line is the theoretical drift of the rubidium oscillator. This information was gathered from the datasheet for some of the points along the graph (1000s, 10000s, 100000s). Time interval sweeps are taken of the drift, and as the timeframe that the plot is generated across increases, the error bars on the graph as shown will decrease.

Up to the 10000s point of the graph the results seem to match up with the theoretical curve really well. They are not expected to to match up perfectly because the TimePort oscillator is not of the same standard as the Rubidium, but the shape of them should match. An interesting point on this plot however is the oscillatory behaviour at 100,000s. It is a point of discussion for two main reasons: it is below the theoretical delay curve of the Rubidium clock, and that it oscillates. The latter point could possibly be smoothened out if more results are collected. There is no immediate reason for the plot to drop below this theoretical mark. SRS most likely will quote a worst case performance for the FS725 on their datasheet (which is where the light blue line was derived from), and thus this line would typically be lower than what is above.

A second Allan deviation plot was created using a different reference light blue curve.

Figure 23: Allan Deviation Plot with a different reference

It can be seen that this line follows the plot gathered much closer. To get a conclusive answer data must be

collected over a lot longer than the twelve days allocated to the collect the data above. The only conclusions that can be drawn from this is that the TimePort is fairly stable over the measured time period, but a longer test would make this more conclusive.

7.2 Delay Distribution

The next set of results is to compare delay distribution of the sets and to see if there are any interesting anomalies with the results. See below the first set of data in Figure 24. The time that these results were taken was unknown, but this will be used as a base line to compare the other results to.

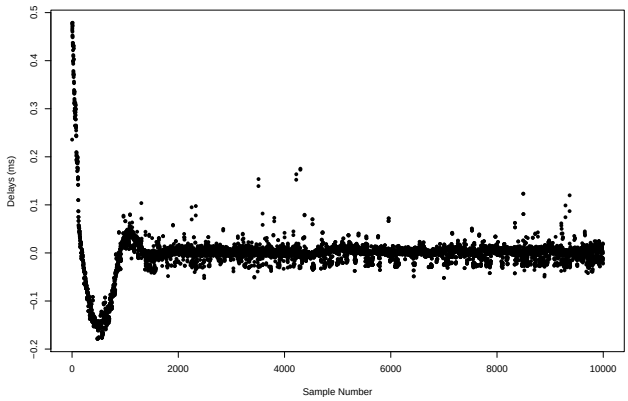
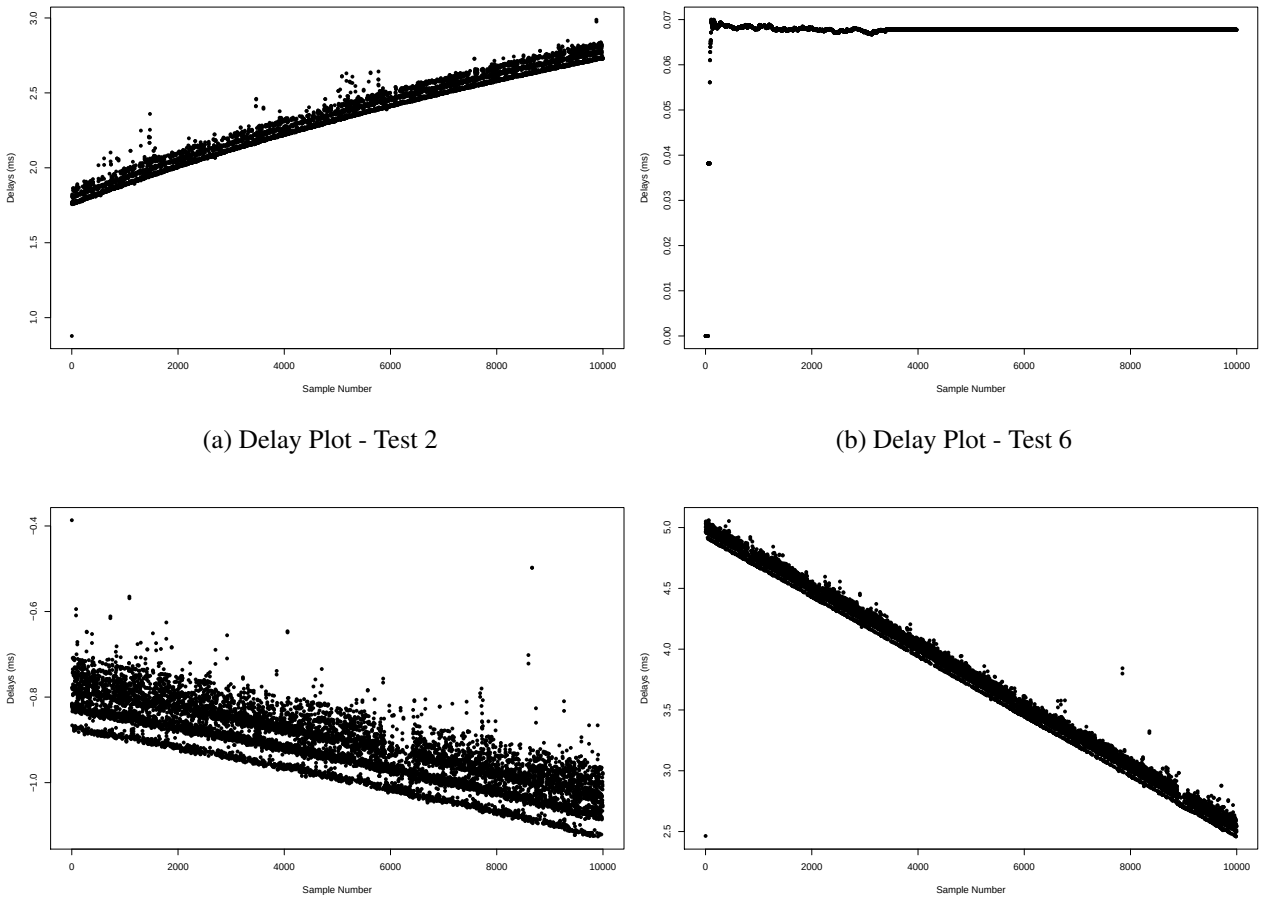


Figure 24: Example Data Set Delay Plot

The first thing to note is that this has been taken at the start of the data set, and thus that is why there is a large delay that slowly tails off. After around 1800 samples (which is about a minute), the delays have stabilised to their steady state condition. The important part of the results is this stabilised region and its overall spread. This same distribution can be seen throughout the data sets collected, with a sample of them shown in the multi figure below.

The four plots below are from Tests 2, 6, 7, and 9. The tests taken earlier on in the project were using a freewheeling grandmaster, and thus these would not be as suitable to analyse.



(a) Delay Plot - Test 2

(b) Delay Plot - Test 6

(c) Delay Plot - Test 7

(d) Delay Plot - Test 9

Figure 25: Example of Delay Plots

Notice that there are some differences between these four examples. Test 6 was the first Beaglebone test performed and it seems to show very similar results to the example data mentioned previously. The reason for the high values of delays in some of these tests were due to the freewheeling mode that the Grandmaster (GM) clock was under, and thus only the distribution can be looked into in any detail.

One interesting point to note however is with Test 7. The delay spread look initially a wide spread, but this is because as the scale has automatically changed, the total delay spread is actually much less in this case. Between samples 6000 and 6500 there is also a noticeable contraction in that delay spread.

As mentioned previously all of these plots were taken using the first 10000 samples. Because it takes some time for the delays to settle down and stabilise, the same tests as above were used but the last 10000 samples were now taken.

7.2.1 Delays During the Test

The delay plots above have been taken from the first 10000 samples of the test. The next set of plots will show the same tests covered above but further on in the data set. The data sets have been calculated 10000 samples in from the end of the data set.

Same as before, the first figure below is with the example data set.

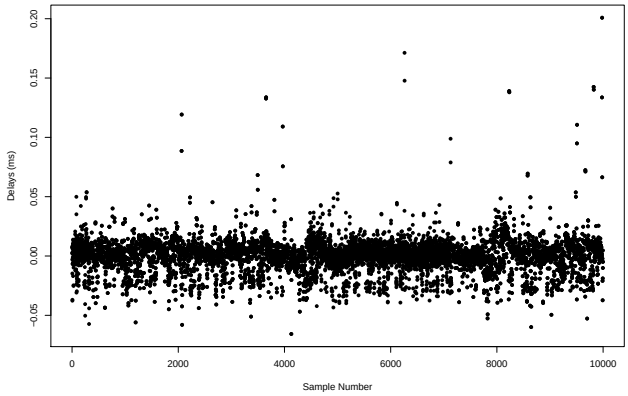
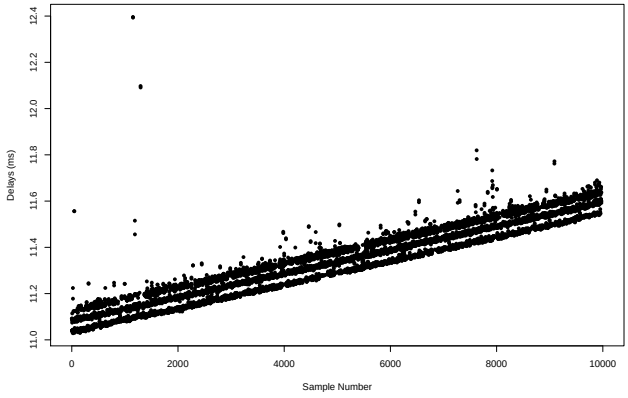
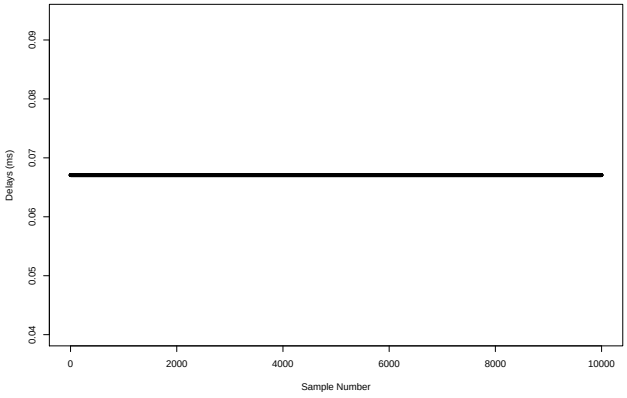


Figure 26: Example Data Set Mid Delay Plot

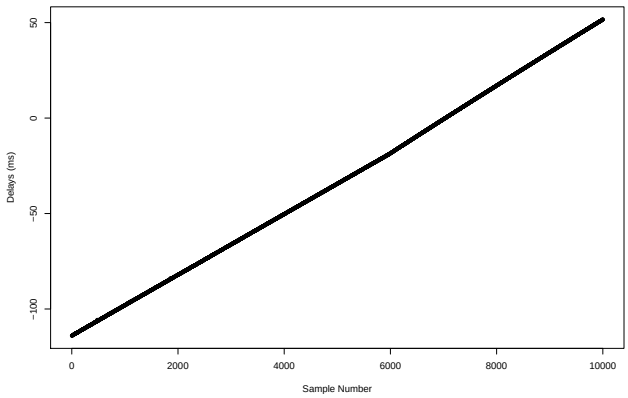
Note that the delay has reached its steady state point and the delay variations around the median are due to errors. The delay is mainly around 10 μs, but has peaks of up to 200 μs. The noise distribution seems to be fairly uniform with no clear discrete noise sources causing these delays.



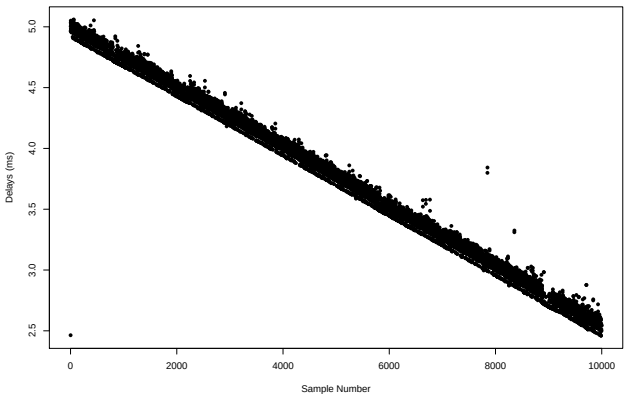
(a) Delay Plot Mid Set - Test 2



(b) Delay Plot Mid Set - Test 6



(c) Delay Plot Mid Set - Test 7



(d) Delay Plot Mid Set - Test 9

Figure 27: Example of Delay Mid Plots

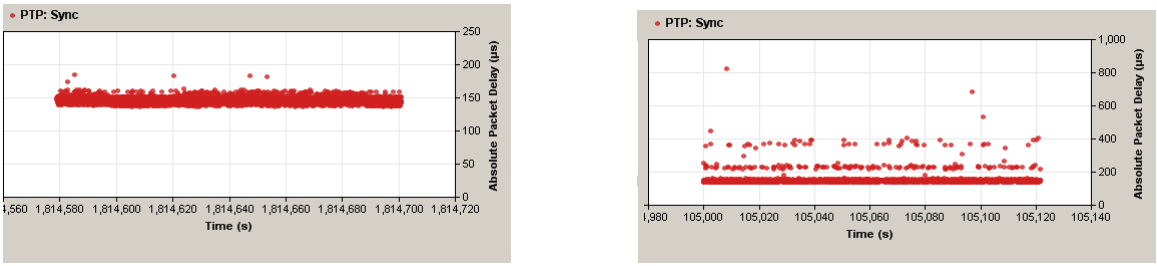
The plots above mean that the majority the three tests (2, 7 , and 9) involved a freewheeling grandmaster and thus the slave clock was not being steered accurately. Test 6 has a very stable delay value with the Beaglebone and the example test data has an expected distribution of delays. As expected with the sixth test the delays

are all the same in the last 10,000 samples thus proving that in this case the grandmaster not freewheeling and that the delays were consistent.

One interesting point to note is with Test 2. Ignoring the fact that the delays are abnormally high and that there are increasing linearly, there are three distinct values of delay (with a noise contribution added to this) along that line. This is highlighted by the fact that there are white lines in between this band of delays. It will be shown later on that this occurs in another test set.

7.2.2 Irregular Delay Magnitudes

A second observation was made with the Syncwatch Plots. See Figure 28a, which shows the delay profile in steady state. As expected the delay is relatively static with some noise over the top. This is in contrast to Figure 28b, where it seems like there are discrete delay variations surrounding the mean value.



(a) Syncwatch Plot Normal

(b) Syncwatch Plot Abnormal

Figure 28: Syncwatch Delay Plots

It can be seen in Figure 28b that there are around three discrete delay values with a couple of other noise peaks. This was very interesting when you compare it to the data discussed in the previous section where there was a clear spread of delay values without seeing any discreteness.

Throughout the data collection phase of the project several occurrences of a very high delay peak was recorded. A screenshot (Figure 29) of some general test statistics is shown below for the main Syncwatch test.

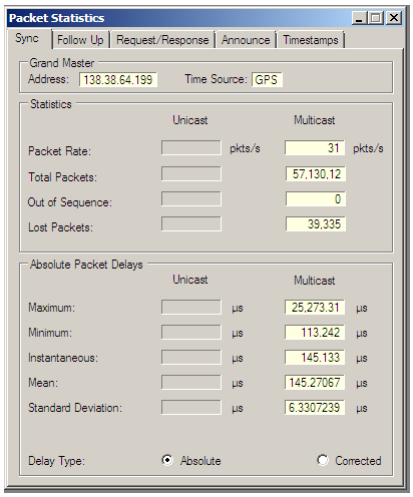


Figure 29: General Packet Data for the Syncwatch Test

The figure above shows that the maximum packet delay value was over 25ms. This is clearly an extremely high packet data considering the test was only set up going through 7 switches. This packet delay variation is

therefore very interesting to note when PTP is run on a typical Ethernet network.

Referring back to Figure 28b, it was not easy to determine what cases this occurred. The day which this was most noticeable coincided with a few network engineers working on a network cabinet downstairs on Level 2, but it was not conclusive whether these two events were related or just a coincidence.

In addition to this it seemed as though if a slave clock running on a lab desktop machine was run in tandem with the Syncwatch test, this delay quantisation effect was reduced to almost nothing, and the delay looked like the normal figure in Figure 28a. Some reasons as to why this happens will be explored at the end of this report.

The final plot for this section (Figure 30) below was also noted on the same day as results day. This is the only occurrence that was noted of this sort of activity happening, and it only seemed to occur this one time. The figure has been taken with a camera phone instead of a screenshot.

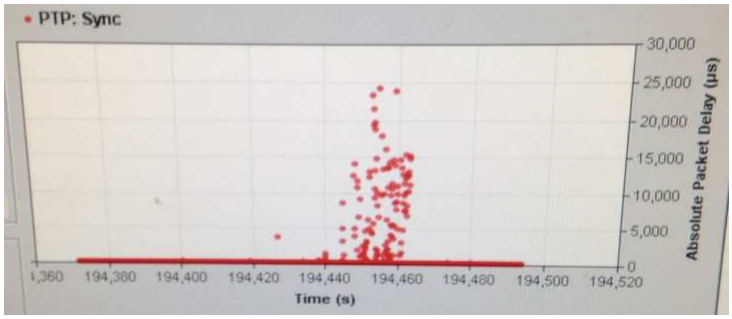


Figure 30: Strange Delay Magnitudes (Taken by Dr Robert Watson)

As it can be seen the period from sample 194,440 to 194,460 has a very high noise source which is causing delays up to 25,000μs (or 25ms). This delay is most definitely not due to a rerouting time difference because of it being very high. Most likely this is caused by the PTP packets being held in a switch for a long time period. It is interesting that it has only happened in this fashion at this one particular time, and that there is a solid column of high delay values (anywhere from 500μs, to 25000μs).

7.2.3 Histogram Distribution

Histograms have been used to be able to easily work out the histogram delay distribution. For example, the plot below is from the example data set. R calculates the number of bins automatically. The complete data set has been used in this case.

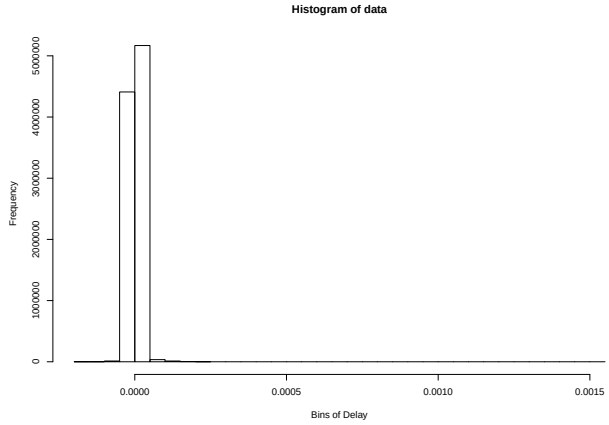
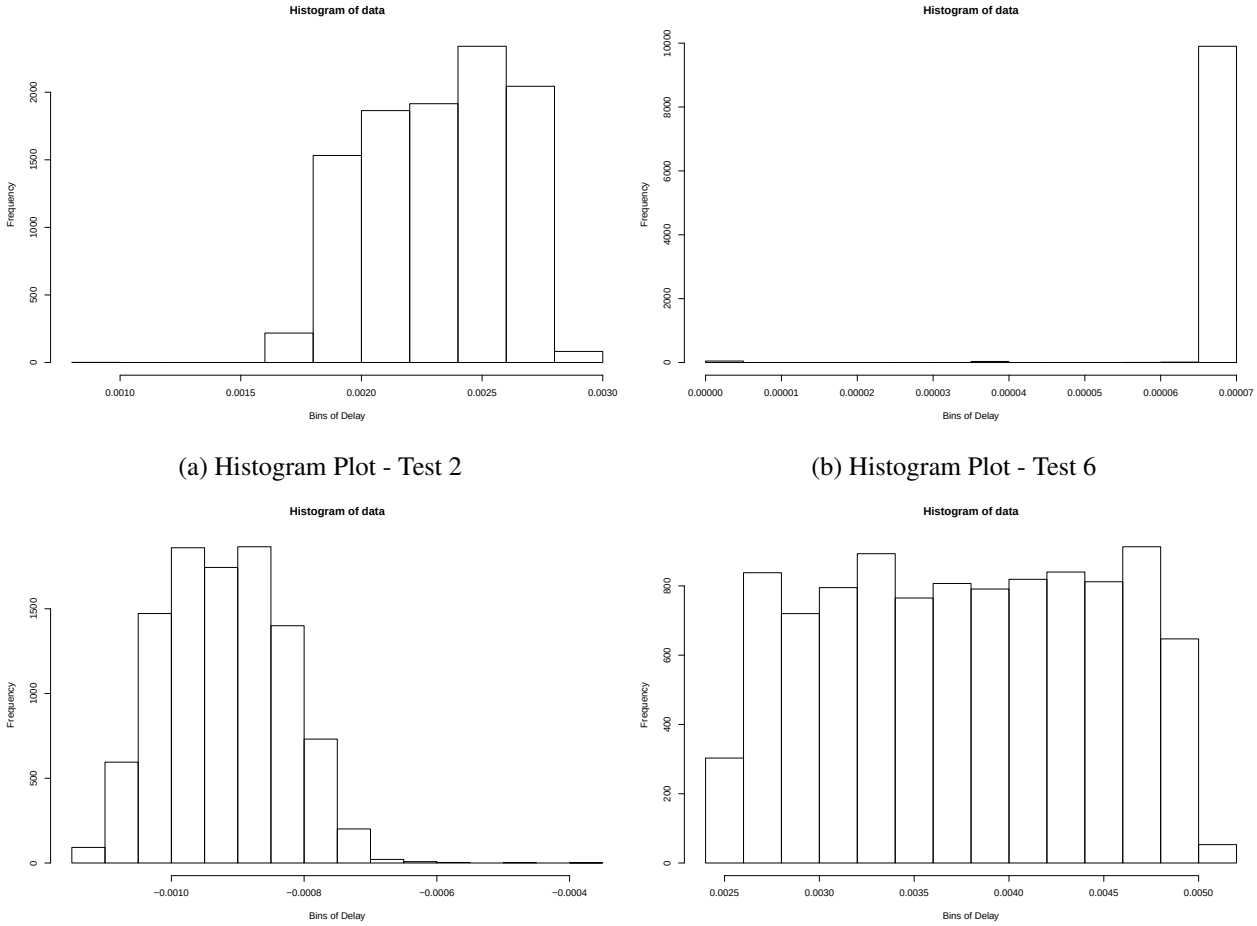


Figure 31: Histogram of Example Set

It can be seen from the above figure that even though the bins have been chosen automatically, the majority

of the delays occur around the zero delay point, and thus it appears that there are only a few bins. There is not an obvious distribution type in this plot due to the large number of delays in those first couple of bins.

The same tests as before we used and histograms were drawn with that data, shown below in Figure 32.



(a) Histogram Plot - Test 2

(b) Histogram Plot - Test 6

(c) Histogram Plot - Test 7

(d) Histogram Plot - Test 9

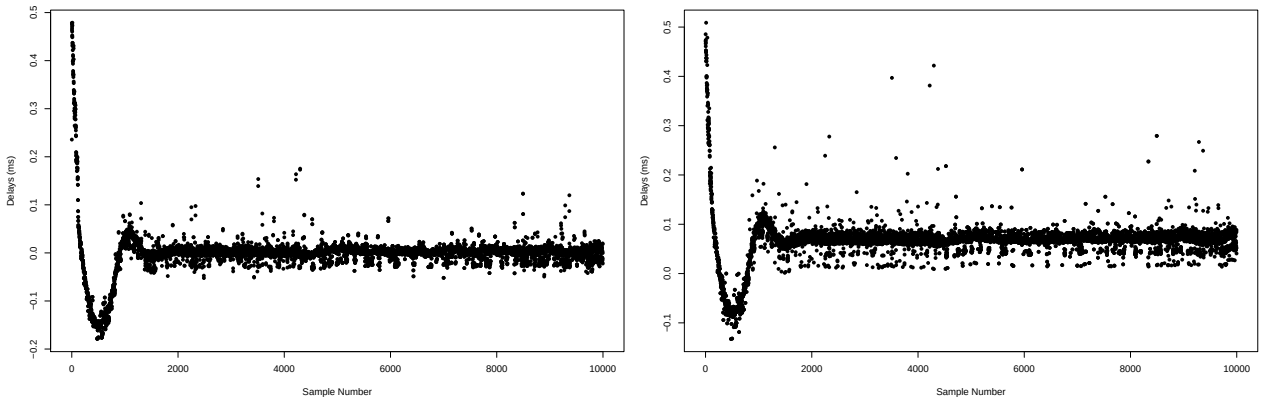
Figure 32: Example Of Delay Plots

As expected Test 6 produces a heavily weighted histogram towards the mode delay value, with the others producing some more distributed histograms. Histogram 7 looks fairly gaussian as expected when looking at the delay plots earlier.

Due to test 6 being very stable at the beginning of the test (based on the delay plots), it is clear as to why the histogram consists mainly of the one value. This is not expected as a normal distribution was hypothesised on the whole. This proves that the delay is very consistent across the 7 switches even though standard network switches are used and there are no transparent clocks.

7.3 Master to Slave vs Slave to Master

The next test would be to compare the one way delay direction against each other for similar tests. The example data set will first be used to see what the delay differences are if any.



(a) Master to Slave Direction (b) Slave to Master Direction

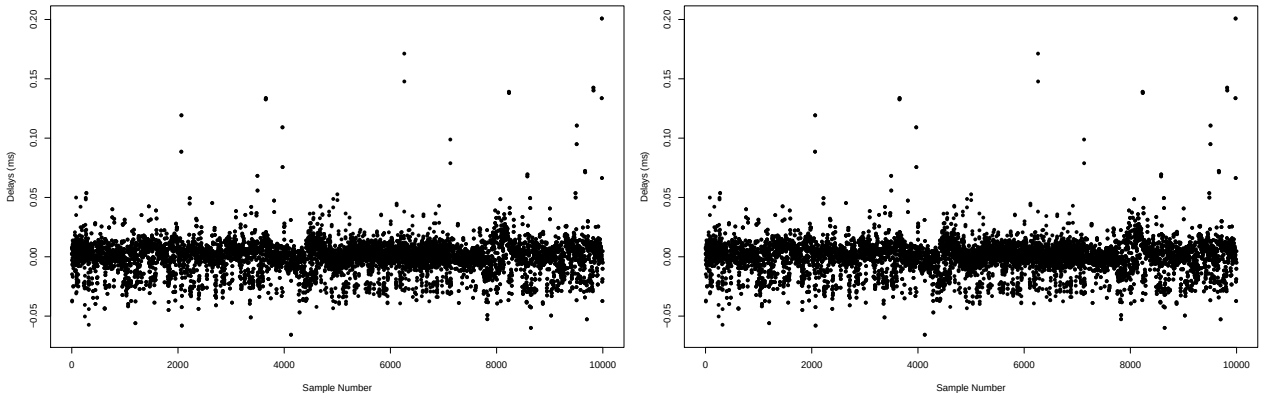
Figure 33: Different Direction Delay Plots

Firstly it is clear that both of the plots above follow the same shape as before, with the damped response before the delays settle down to their normal value. There is also noise points around the set of normal values as expected.

The differences though are surprising. Firstly there are more noise points around the mean value of Figure 33b than the other one. It can also be seen that there are higher noise peaks, at around 450μs compared with the Master to Slave plot that peaks at around 200μs. It is also noted that there more noise points below the average number than in the previous plot.

The next observation is that the average value in the Slave to Master direction is higher (around 100μs) compared with the MAster to Slave direction (around 10μs). There are not any obvious reasons for this increase in delay, but it will be discussed in a later part of this report.

To make sure that this did not only occur at the beginning of the data set, a plot of the end of the data set was also created. (Figure 34)



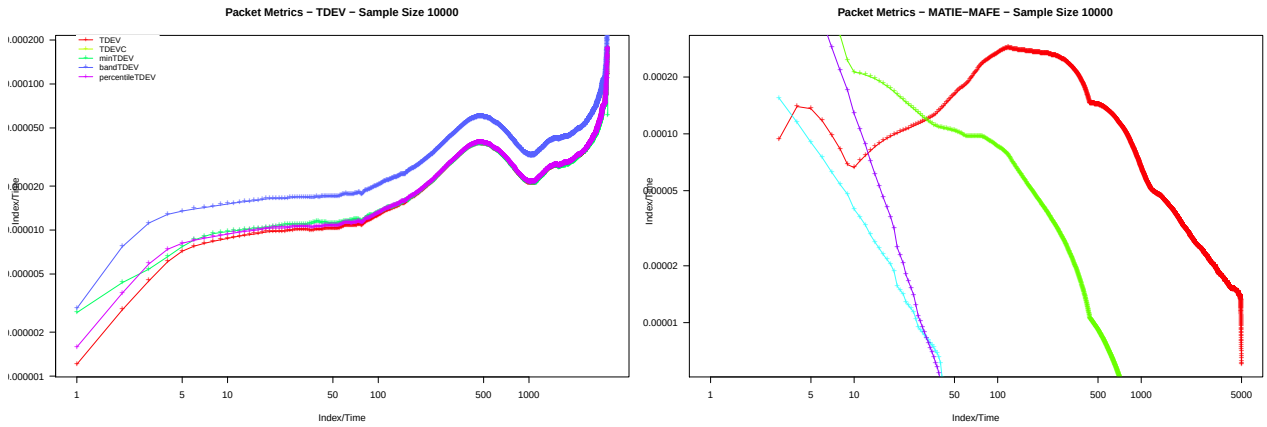
(a) Master to Slave Direction (b) Slave to Master Direction

Figure 34: Different Direction Delay Plots (End of Data Set)

As it can be seen these plots are identical, and thus it seems like the one way delay differences gets better to the point where they are equal at the end of the test.

7.4 Metric Results

The next set of results calculated were using the packet metric scripts created in a previous section of this report. Same as before, the scripts were run using the Example test data first and some initial conclusions were drawn. The scripts then calculated both metrics and plotted them for all of the data sets collected. The example data set metric plots can be seen below : Figures 35a and 35b for TDEV and minTDEV respectively.



(a) Test 0 : TDEV Plot

(b) Test 0: MATIE MAFE Plot

Figure 35: Example Data Metric Plots

The packet metric recommendation report [13] detailed in the literature view did not go into any detail in how to process this metric data or any method in determining what to look for when regarding these plots. Therefore at this stage only comparisons between data sets will be made which can then be compared to the raw delay data discussed previously. Some effort in comparing these two sets of data will be made to try and then see what the packet delays can show that the delay data cannot.

As it can be seen above all of the TDEV metrics have been plotted on one plot, with the four MATIE / MAFE plots on the second (Figure 35b). The conditions of the example test are not known completely, therefore no conclusions can be drawn from the metric data above.

The plots below do not show the legends on each of them, but all share the same colour set. The legends have been provided below, Figure 36a for the TDEV plot, and Figure 36b for the MATIE / MAFE plot.

(a) TDEV Legend

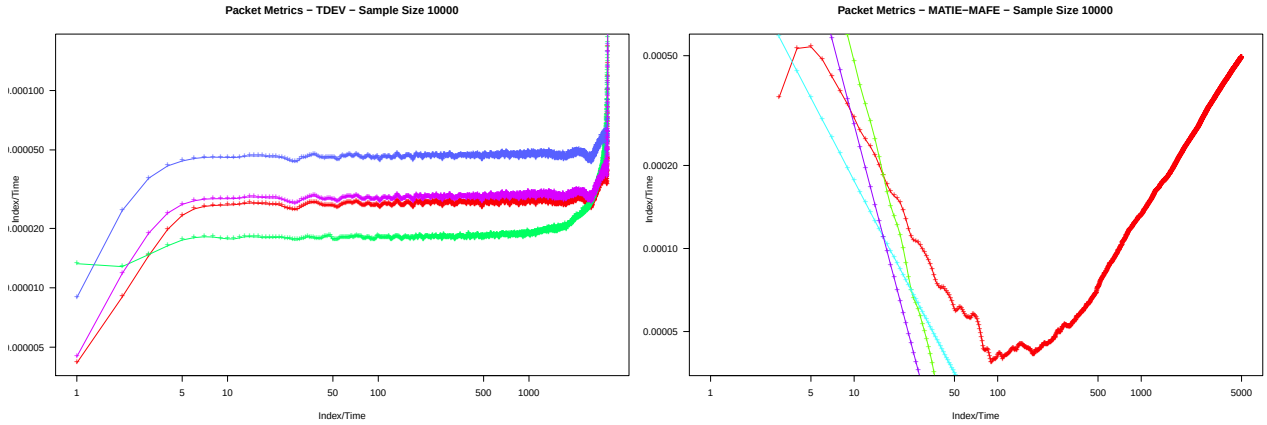
(b) MATIE/MAFE Legend

Figure 36: Plot Legends

The plots are plotted on the same figure such that they can be compared against each other.

7.4.1 Test 2 Packet Metric Results

The first packet metric scripts run were on the second set of test results (excluding the example set.) Other repetitions of this test was also performed. The metric plots can be seen below: Figures 37a and 37b.



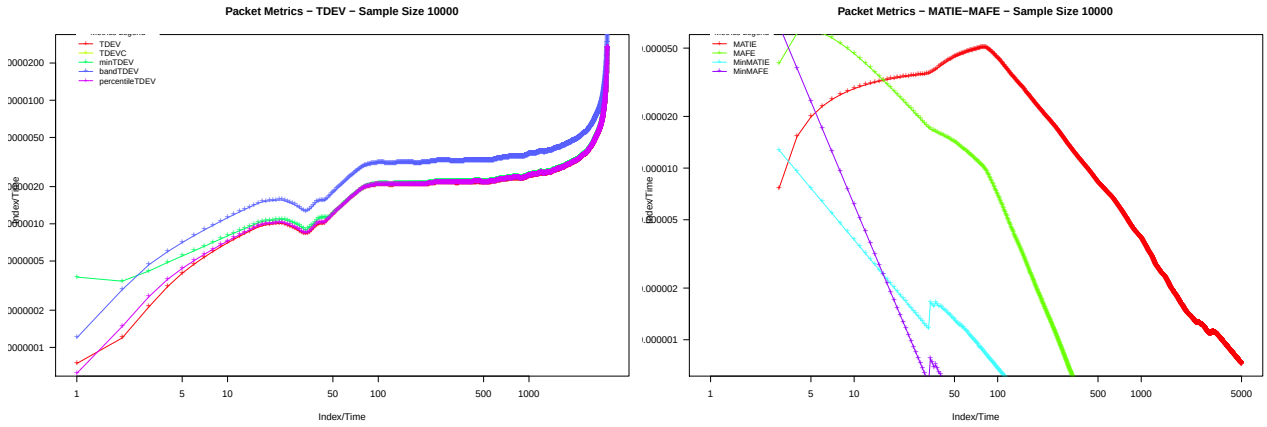
(a) Test 2 : TDEV Plot

(b) Test 2: MATIE MAFE Plot

Figure 37: Test 2 Metric Plots

The TDEV plots are noisier and oscilliatory during 100 and 1000 samples, but eventually stabilises to just noise later on in the plots. The MATIE and MAFE plots should be a smoother line than this, but this seems reasonable given the conditions that this test was in.

7.4.2 Test 6 Packet Metric Results



(a) Test 6 : TDEV Plot

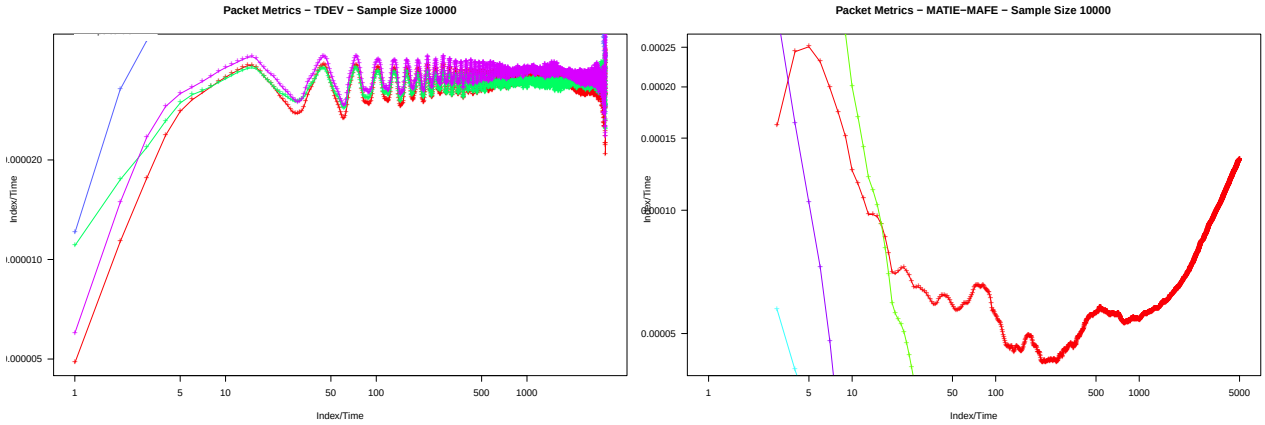
(b) Test 6: MATIE MAFE Plot

Figure 38: Test 6 Metric Plots

Test 6 were the best results gathered, both from the delay data mentioned previously and the above. The MATIE and MAFE plots are smooth linear lines when plotted on a log scale and the TDEV plots are not oscillatory as they were in the other examples. Therefore PTP seems to hold up well under these conditions.

7.4.3 Test 7 Packet Metric Results

The next set of results was Test 7.



(a) Test 7 : TDEV Plot

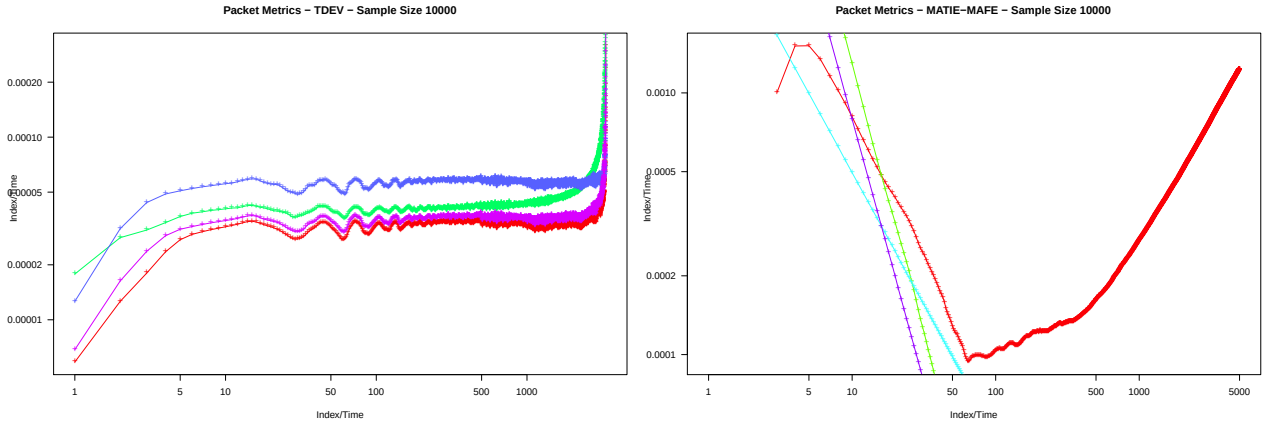
(b) Test 7: MATIE MAFE Plot

Figure 39: Test 7 Metric Plots

This contrasts quite heavily with the previous set of results as there is a lot of oscillatory behaviour in the TDEV results and the MATIE plots are not as smooth as before. It is also interesting to note that there are not many points for the minimum MATIE plot (light blue line). This is most likely because there were several NA values which are not plotted on the graph.

7.4.4 Test 9 Packet Metric Results

The final set of results used was Test 9.



(a) Test 9 : TDEV Plot

(b) Test 9: MATIE MAFE Plot

Figure 40: Test 9 Metric Plots

Similarly with the previous set, there is some oscillatory behavoir in the TDEV results, but the MATIE plots look correct.

8 Conclusion

Overall the project has gone very well. The majority of the objectives that were set at the beginning of the project were completed and some interesting results were found.
Packet delays seem to be fairly consistent even with it being measured on a busy network. The packet metrics could also be correlated with this delay data and patterns were noticed.

In conclusion the PTP protocol was investigated successfully in a number of different ways. PTP data was collected in a number of different locations throughout the university on several different clock configurations. This data was then analysed: both through looking at the raw delays and from the use of packet metric scripts.

8.1 Objectives Completed

The following milestones have been identified for this project:

Gained Knowledge on PTP Knowledge on a technology not covered during university was learnt thoroughly to then help with the project. Several papers were covered on the topic, both from a technical aspect on the current protocol and look into the future for PTP in terms of collision detection and securing the protocol.

Set up a basic PTP Network With the knowledge gained from above and with the availability of both grandmaster and slave clocks, a basic network was set up with the TimePort as the main Grandmaster and several different slave implementations. This was then used to collect delay data from a few locations on the university network.

Implementing Packet Metric Scripts Several packet metric script were implemented from scratch in an unfamiliar language. The packet metric ITU recommendation report was read and the metric scripts were created based off of this. The script created was built in a user-friendly way and has tried to be written in a self-documentating fashion.

PTP Delay Results PTP delay results were gathered and analysed and some surprising findings were noticed even with standard networking switches. These results can then be used to compare against subsequent work undertaken on PTP.

8.2 Challenges

Several challenges were identified throughout this project, which have been summarised below. They were split up into the different subtasks identified for this project.

Research into Packet Metrics and the Technology

The main challenge when researching various parts of this project was the limited amount of reports on the specific packet metrics used. Excluding the ITU report already covered in the literature review, there was very little information on these metrics. Thus it made it difficult to understand how this packet metric data was processed and analysed.

Data Collection

The challenges associated with data collection mainly consisted of working with the hardware. With the TimePort, there were some issues with the device not holding a solid GPS lock and thus was freewheeling. After this the TimePort was not able to get a GPS lock at all. The GPS module was replaced with a spare one and the TimePort was able to maintain a lock successfully.

One other issue was that the position of the TimePort seemed to determine whether this lock occurred. Once a firmware update fixed a serial corruption issue (as noted in a screenshot earlier in this report), the oreintated of the TimePort error was also solved. The Syncwatch also had one main issue of the Ethernet ports on the device were not activated. Instead the other ports had to be used instead.

The one other issue of data collection arose due to the availability of either docking ethernet ports or available ethernet ports. This issue was solved by choosing areas of the university where there were ports available.

Packet Metric Implementation
During the packet implementation stage of the project the challenges faced were usually based on the implementation or of the syntax of the language, but this would occur during any programming project. The only challenge was verifying that the results calculated were correct.

9 Further Work

Due to the openness of the project, there is a number of additional tasks that could be completed to continue the investigation. These have been identified below.

Different Network Topologies Due to the equipment available, it was difficult to test PTP on different network types. Therefore it would be beneficial to perform the same tests as before but with different topologies. This would enable to better test the protocol in more common scenarios with the use of boundary and transparent clocks.

Gather more data More network data is required to be able to better analyse the performance of PTP on the network. More network locations and at different times of day should be tested.

More Packet Metrics Once a better understanding of the packet metrics is gathered, more packet metrics can be implemented where necessary.

Verify Packet Metric Scripts against existing programs Because there were no existing packet metric programs available to be used during this project, the scripts were not tested for validity. The results looked correct hence why these were used for investigating PTP performance. Chronos has a packet metric software suite as an add on to the Syncwatch software

Secure PTP There has been some work done when describing Secure PTP (Section **??**), but if there was more time more work on this would have been investigated. Some work into making a PTPd secure implementation would be looked at. In tandem with this a PTP network could be set up and attempts would be made to try and damage the network through its various attack vectors to demonstrate PTPs weaknesses.

Comparing time of day As network data was not collected soon enough into the project, plus there was not sufficient data collected from different locations, it was difficult to be able to correlate time of day and the data already collected. This would be an interesting area to investigate as it would show the extent at which packet collisions have an effect on PTP traffic. If this would be an issue then work into looking at token passing as an option would then be considered. PTP performance with token passing can then be compared with and results can be drawn up to see if this improves PTP.

10 Extra Work

10.1 Securing PTP against attacks

This section has been created using a PTP security tutorial [23]

PTP is inherently an insecure system with no standard methods in either encrypting communications between the master and the slave devices or any method of verifying that a particular grandmaster is legitimate. Therefore the slave clocks rely on trusting the grandmaster is an accurate time reference and blindly synchronising with the masters. The issue arises if a master or grandmaster was compromised and clock shift delays were spoofed. This would cause all of the slave clocks on the network to drift away from the actual time reference. The impact on critical systems mentioned in this report would be huge: timestamps for telecommunications data would be invalid or power transmission relays would trigger in the wrong order. Therefore this section of the report will highlight some of the attack vectors that could affect PTP and methods in which to help mitigate this.

10.1.1 Possible Attack Vectors

There a number of different methods that are possible with the existing PTP standard. These are:

Control Plane Attack

This attack is an attack specifically on the Best Master Clock (BMC) algorithm which is highlighted in Section 1.2.4. It works by having a compromised host (pictured below in red, Figure 41 [23]) announcing to the network that it has set the highest priority flag to 1.

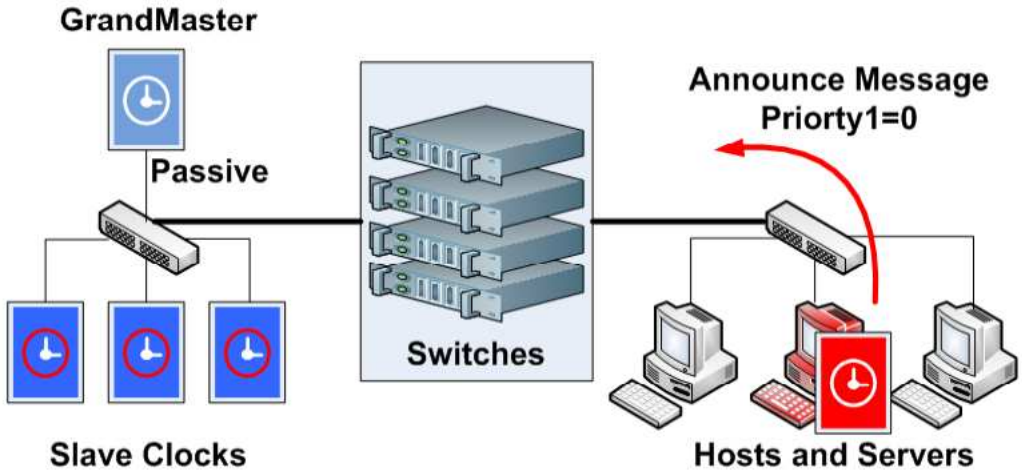


Figure 41: Control Plane Attack Vector

This would set it to become the Grandmaster clock on the network based on the BMC algorithm. The existing GM clock is now passive and the new grandmaster can now steer all of the slave clocks on the network by reporting false timestamps. Provided that the compromised host keeps their priority level at the highest, the only method of fixing this type of attack would be for another master clock to also set their priority to the highest. The BMC algorithm will then drop to the next level, which is Clock Type (need to check).

A more sophisticated version of this attack could be for the compromised clock to eventually mirror all of the parameters for the current grandmaster, such that it would be a random choice which clock would be chosen as the Grandmaster.

Sync Plane Attack

This attack type involves the compromised clock to learn enough about the existing grandmaster to be able to spoof messages as if the compromised clock was the grandmaster. It does this by learning the GM identity, addresses, sync sequence number and interval.

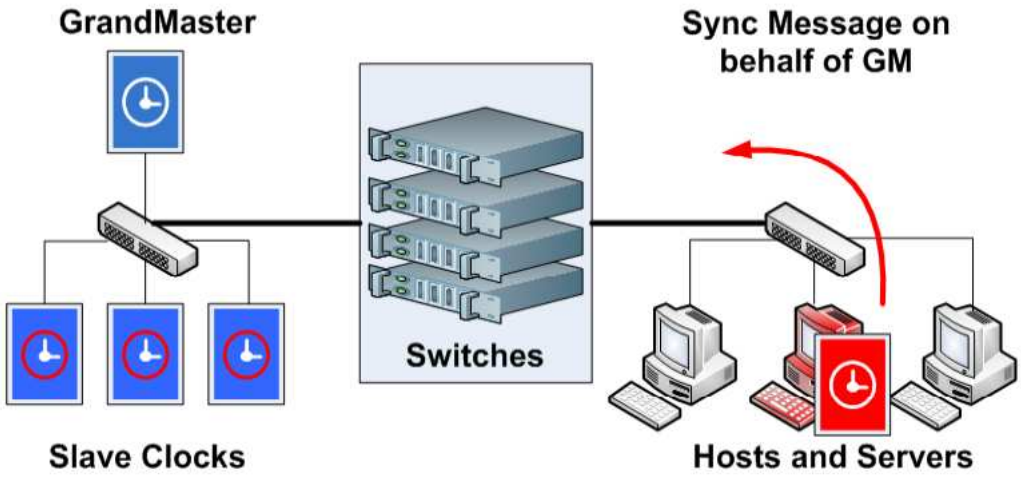


Figure 42: Sync Plane Attack Vector

Figure 42 above was taken from the PTP Security Tutorial [23]. From the perspective of one of the slaves, this compromised clock is exactly the same as the current grandmaster. The compromised clock can then send sync messages, thus hijacking the slave clocks on the network. This is a form of a masquerade attack.

Due to the nature of this attack, it would be difficult to attempt to mitigate the risk of this type of attack occurring on a PTP network because the real grandmaster and the hacked host look identical.

One way that network administrators could attempt to mitigate this however is to restrict any two devices with the same parameters on the network from communicating. This would involve some sort of authentication process that cross checks all possible masters.

Management Plane Attack

The final attack type involves using a management command to gain grandmaster access on the network. Figure 43 [23] is a diagram of an example of a management plane attack.

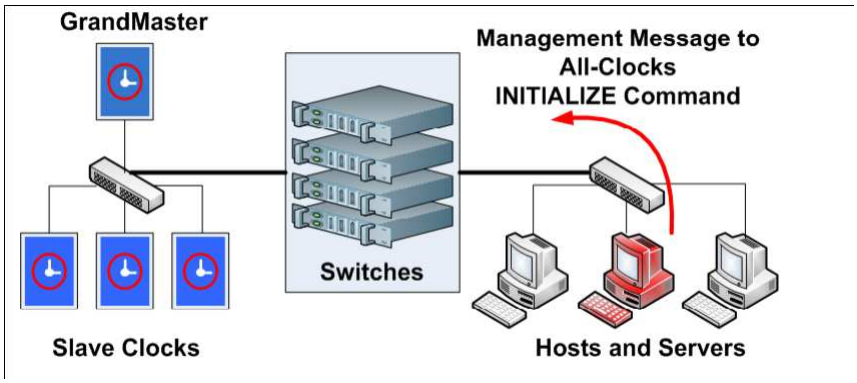


Figure 43: Management Plane Attack Vector

It involves gaining access to the clocks to disrupt network operation by sending an initialise command.

Delay Attacks

The final attack type is a delay attack, which involves a compromised switch instead of a host.

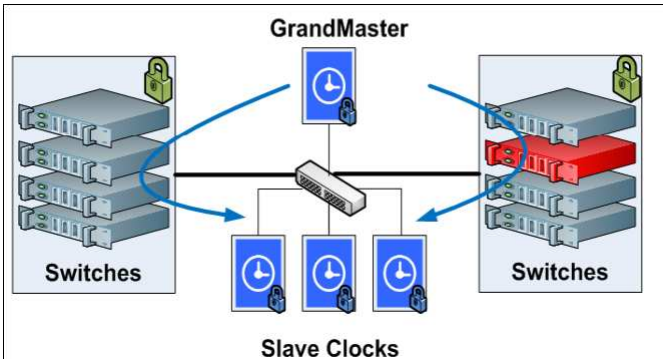


Figure 44: Delay Attack

Figure 44 demonstrates what would happen if this occurred. The hijacked switch (when acting like a boundary clock) would be able to report incorrect delay values which would then propagate these errors across the network.

The problem with mitigating this type of attack is, unlike the three previous attack types, is that it would be difficult to redirect routing of the PTP packets around the compromised switch. One way of mitigating is that the delays could be sanity checked with a number of different routes, similar to how clocks in every stratum in NTP are sanity checked.

10.1.2 Methods to mitigate the above attack vectors

There are several general methods of securing a PTP network in general, with some other specific methods also mentioned in relation to the attacks mentioned above. The general methods are:

- Physically Securing the Network
- Use separate sub-networks to limit the effect of multicast communication
- Limit the grandmasters to a pre-defined list by implementing a whitelist
- Limit the management address to a pre-defined list
- Snoop source address to attempt to identify masquerade attacks

The above methods would be able to mitigate some of the attack types but they are either not always possible to implement, don't cover some of the more serious threats to the network, or the goal of minimising central administration would not be reached. For example, physically securing the network may not always be foolproof or may not be possible in certain circumstances. The issue with limiting either the grandmasters or the management addresses is the extra overhead involved in administrating the network, which goes against the original goals of PTP to have distributed control.

IEE1588-2008 standard includes a section on secure PTP which aims to address these concerns. This is in Appendix J of the standard. Some of the solutions mentioned in the standard will be discussed here, along with some comments and suggestions to move forward, and areas that could be later worked on to better improve the security of PTP.

10.1.3 Security Protocol Recommendation

Annex K of the IEEE1588-2008 specification outlines a recommendation to how secure PTP could be implemented. It explicitly states that this section is not a requirement to meet the standard, just a possible way of implementing it. The extension to the standard includes group source authentication, message integrity and

replay attack protection, which would help to mitigate some of the attack vectors mentioned previously.

The security protocol includes two main elements: an integrity protection mechanism and a challenge-response mechanism. Symmetric message authentication code functions are used which provides the advantages of replay protection, group source authentication and message integrity. The standard recommends two main authentication standards (HMAC-SHA1-96 **??** and HMAC-SHA256-128 **??**), but there is a possibility for the standard to support more than these.

Users on the PTP network will share symmetric authentication keys, which can either be shared across an entire domain or in subsections of it. There are two ways of key distribution: either manual or an automatic key management protocol.

Security Associations The method of communication between users on the PTP network is through Security Associations (SAs). The contain the following fields:

- Source (Source port and Protocol Address)
- Destination (Destination Address and Protocol Address)
- key (either SHA256-128 or SHA1-96)
- a random lifeTimeID
- a reply counter

The SA is a unidirectional transaction, therefore each node on the network needs to maintain a list of both incoming SAs as well as outgoing. They can be shared by a single sender and multiple receivers, but each receiver holds its own copy of the SA. This will work provided that each of the receiver copies holds a different value of the reply protection counter at the same time. All of them must be smaller than the counter stored in the sender's copy. The SA is generated by the sender, and can be sent to all of the receivers, or a separate one to each of them.

11 Acknowledgements

Firstly I would like to thank Dr Robert Watson for his help with this project. I would also like to thank Chronos for both the availability of the equipment as well as being there with help if necessary.

References

[1] J. Cox, "Interim report: Investigation into the precision time protocol."

[2] D. S. Mohl, "Ptp applications," http://www.ieee1588.com/IEEE1588_PTP_Applications.html, 2010, accessed: 2014-02-11.

[3] M. R. Bernhard Baumgartner, Christian Riesch, "Ieee 1588/ptp: The future of time synchronization in the electric power industry," https://www.picotest.com/downloads/OTMC100/IEEE_1588_PTP_-_The_Future_of_Time_Synchronization_in_the_Electric_Power_Industry.pdf", accessed: 2014-02-11.

[4] NERC, "Disturbance monitoring equipment installation and data reporting," http://www.nerc.com/files/prc-018-1.pdf, accessed: 2014-02-11.

[5] T. R. A. of Engineering, "Global navigation space systems: Reliance and vulnerabilities," http://www.raeng.org.uk/news/publications/list/reports/raoe_global_navigation_systems_report.pdf, accessed: 2014-04-08.

[6] L. Carroll, "Executive summary: Computer network time synchronization," http://www.eecis.udel.edu/~mills/exec.html, accessed: 2014-02-11.

[7] D. L. Mills, "Executive summary: Computer network time synchronization," http://www.eecis.udel.edu/~mills/exec.html, accessed: 2014-02-19.

[8] M. R. CLOCKS, "What is the difference between ntp and sntp?" http://www.meinbergglobal.com/english/faq/faq_37.htm, accessed: 2014-02-17.

[9] J. B. D. Mills, J. Martin, "Network time protocol version 4: Protocol and algorithms specification," http://tools.ietf.org/html/rfc5905, accessed: 2014-02-11.

[10] W. P. N. C. S. W. Group, "1588-2008 - ieee standard for a precision clock synchronization protocol for networked measurement and control systems," http://standards.ieee.org/findstds/standard/1588-2008.html, 2008, accessed: 2014-02-13.

[11] RadhaKrishna.Arvapally, "Ieee1588 communication mechanism," http://en.wikipedia.org/wiki/File:IEEE1588_1.jpg, accessed: 2014-02-15.

[12] Cisco, "Cisco cgs 2520 switch software configuration guide for ios release 12.2(58)ey," http://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cgs2520/software/release/12_2_58_ey/configuration/cgs_2520_swcg.html, accessed: 2014-04-04.

[13] ITU, "Definitions and terminology for synchronization in packet networks," *SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS : Packet over Transport aspects Quality and availability targets*, 2012.

[14] Chronos, "Ctl4540 timeport," timePort Datasheet.

[15] ——, "Syncwatch standalone," chronos Slave Clock Datasheet.

[16] G. Coley, "Beaglebone black system reference manual," http://www.jameco.com/Jameco/Products/ProdDS/2176149.pdf, accessed: 2014-04-07.

[17] http://www.makershed.com/BeagleBoard_BeagleBone_Black_p/mkcce3.htm, accessed: 2014-04-23.

[18] E. Technologies, "Precision time protocol white paper," http://www.endruntechnologies.com/pdf/PTP-1588.pdf, accessed: 2014-04-07.

[19] M. Brennan, "The gnu awk user's guide," http://www.gnu.org/software/gawk/manual/gawk.html, accessed: 2014-04-07.

[20] T. L. Davis, "argparse: Command line optional and positional argument parser," http://cran.r-project.org/web/packages/argparse/index.html, accessed: 2014-04-11.

[21] M. Frasca, "Logging," http://logging.r-forge.r-project.org/, accessed: 2014-04-12.

[22] S. R. Systems, "Fs725 benchtop rubidium frequency standard," http://www.thinksrs.com/downloads/PDFs/Catalog/FS725c.pdf, accessed: 2014-04-26.

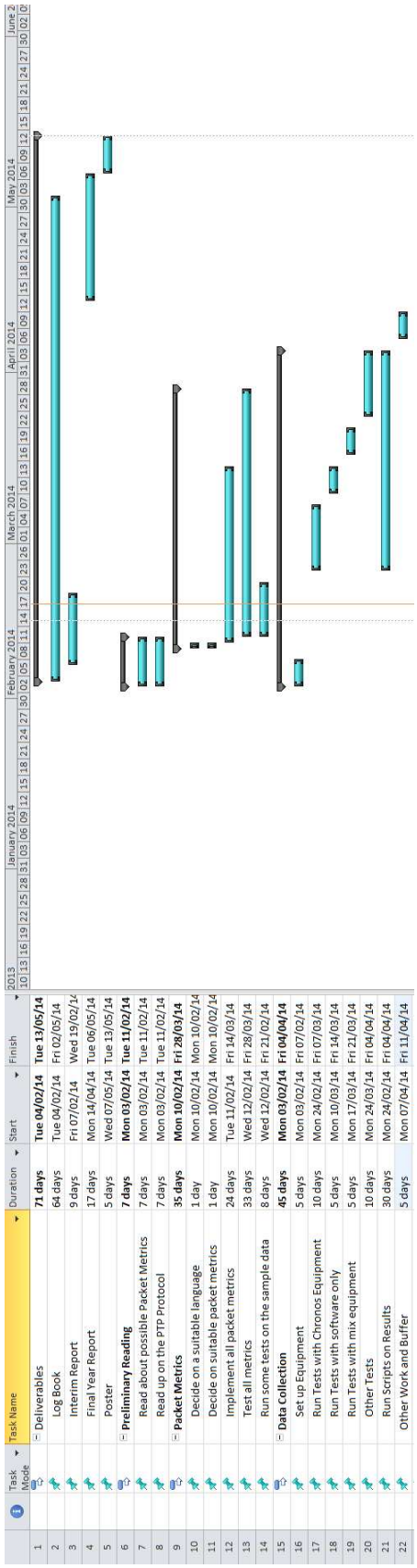
[23] R. Cohen, "Ptp security tutorial," http://www.ispcs.org/security/downloads/PTPSecurityTutorial.pdf, accessed: 2014-04-26.

1 Clock Accuracies

Table 14: Clock Accuracies

	Quartz Oscillators	Atomic Oscillators		
	OCXO	**Rubidium**	**RbXO**	**Ceasium**
Accuracy (per year)	1×10^{-8}	5×10^{-10}	7×10^{-10}	2×10^{-11}
Ageing (per year)	5×10^{-9}	2×10^{-10}	2×10^{-10}	0
Temperature Stability	1×10^{-9} (-55 to 85)	3×10^{-10} (-55 to +68)	5×10^{-10} (-55 + 85)	2×10^{-11} (-28 to +65)
Stability, Sy t = 1s	1×10^{-12}	3×10^{-12}	5×10^{-12}	5×10^{-11}
Size (cm^2)	20-200	800	1200	6000
Warmup Time (minutes)	4 (to 1×10^{-8}	3 (to 5×10^{-10})	3 (to 5×10^{-10})	20 (to 2×10^{-11})
Power (W)	0.06	20	0.65	30
Price ($)	2000	8000	10000	40000

2 Gantt Chart and Table



	Task Mode	Task Name	Duration	Start	Finish
1		**Deliverables**	**71 days**	**Tue 04/02/14**	**Tue 13/05/14**
2		Log Book	64 days	Tue 04/02/14	Fri 02/05/14
3		Interim Report	9 days	Fri 07/02/14	Wed 19/02/14
4		Final Year Report	17 days	Mon 14/04/14	Tue 06/05/14
5		Poster	5 days	Wed 07/05/14	Tue 13/05/14
6		**Preliminary Reading**	**7 days**	**Mon 03/02/14**	**Tue 11/02/14**
7		Read about possible Packet Metrics	7 days	Mon 03/02/14	Tue 11/02/14
8		Read up on the PTP Protocol	7 days	Mon 03/02/14	Tue 11/02/14
9		**Packet Metrics**	**35 days**	**Mon 10/02/14**	**Fri 28/03/14**
10		Decide on a suitable language	1 day	Mon 10/02/14	Mon 10/02/14
11		Decide on suitable packet metrics	1 day	Mon 10/02/14	Mon 10/02/14
12		Implement all packet metrics	24 days	Tue 11/02/14	Fri 14/03/14
13		Test all metrics	33 days	Wed 12/02/14	Fri 28/03/14
14		Run some tests on the sample data	8 days	Wed 12/02/14	Fri 21/02/14
15		**Data Collection**	**45 days**	**Mon 03/02/14**	**Fri 04/04/14**
16		Set up Equipment	5 days	Mon 03/02/14	Fri 07/02/14
17		Run Tests with Chronos Equipment	10 days	Mon 24/02/14	Fri 07/03/14
18		Run Tests with software only	5 days	Mon 10/03/14	Fri 14/03/14
19		Run Tests with mix equipment	5 days	Mon 17/03/14	Fri 21/03/14
20		Other Tests	10 days	Mon 24/03/14	Fri 04/04/14
21		Run Scripts on Results	30 days	Mon 24/02/14	Fri 04/04/14
22		Other Work and Buffer	5 days	Mon 07/04/14	Fri 11/04/14

Figure 45: Gantt Chart

	Task Name	Duration (days)	Start	Finish	Predecessors
A	**Deliverables**	72	Mon 03/02/14	Tues 13/05/14	
1	Log Book	65	Mon 03/02/14	Fri 02/05/14	
2	Interim Report	9	Fri 07/02/14	Wed 19/02/14	B, 1
3	Final Year Report	17	Mon 14/04/14	Tues 06/05/14	B,C,D,1,2
4	Poster	5	Wed 07/05/14	Tues 13/05/14	1,3
B	**Preliminary Reading**	**7**	**Mon 03/02/14**	**Tues 11/02/14**	
1	Read up on PTP	7	Mon 03/02/14	Tues 11/02/14	
2	Reading about Metrics	7	Mon 03/02/14	Tues 11/02/14	
C	**Packet Metrics**	**35**	**Mon 10/02/14**	**Fri 28/03/14**	
1	Decide on Language	1	Mon 10/02/14	Mon 10/02/14	
2	Decide on Metrics	1	Mon 10/02/14	Mon 10/02/14	
3	Implement Metrics	24	Mon 11/02/14	14/03/14	1,2
4	Test all Metrics	33	Wed 12/02/14	Fri 27/03/14	3
5	Run some tests on sample Data	8	Wed 12/02/14	Fri 21/02/14	3
D	**Data Collection**	**45**	**Mon 03/02/14**	**Tues 04/04/14**	
1	Set up Equipment	5	Mon 03/02/14	Fri 07/02/14	
2	Run Tests w/ Chronos Equipment	10	Mon 24/02/14	Fri 07/03/14	1
3	Run Tests w/ Software	5	Mon 10/03/14	Fri 14/03/14	1
4	Run Tests with w/ Mix	5	Mon 17/03/14	Fri 21/03/14	1
5	Other Tests	10	Mon 24/03/14	04/04/14	1
6	Run scripts on Results	30	Mon 24/02/14	Fri 04/04/14	2,3,4,5
E	**Other Work and Buffer**	**12**	**Mon 07/04/14**	**Fri 18/04/14**	

3 Test Sheet Example

Test: Timeport_to_Software Test One				
Test Name:	TimePort_To_Software Test One			
Test ID:	001			
Test Date	2014-02-27			
File Name:	RawData.txt			
Directory:	./PTPData/TimePort_To_Software_Test1			
Start Time:	1037			
End Time:	2200			
Clock #1 Type:	Hardware		**Clock #2 Type:**	Software
Clock #1 Name:	TimePort_1		**Clock #2 Name:**	PTPd_Netbook
Clock #1 Model:	TimePort		**Clock #2 Model:**	PTPd
Clock #1 Location:	Watson's Office		**Clock #2 Location:**	2E 2.13
Network Activity:	Normal			
Test Description:	An initial test to collect data to supplement the example data already received.			
Comments	1342: Data seems to be collecting fine. 3hrs20mins: 45MB			

4 Test Sheet Summary Sheet

Test Number	Date	Directory	Master	Slave	Location Master	Location Slave	Start Time	End Time
001	27/02/14	TimePort-To-Software-Test1	TimePort_1	PTPd_Netbook	2E 4.6	2E 2.13	1037	1720
002	28/02/14	TimePort-To-Software-Test2	TimePort_1	PTPd_Netbook	2E 4.6	2E 2.13	1140	1700
003	28/02/14	TimePort-To-Software-Test3	TimePort_1	PTPd_Desktop	2E 4.6	2E 4.6	1015	Unknown
004	03/03/14	TimePort-To-Software-Test4	TimePort_1	PTPd_Netbook	2E 4.6	2E 2.13	0810	1400
005	03/03/14	TimePort-To-Software-Test5	TimePort_1	PTPd_Netbook	2E 4.6	Library	0929	1702
006	03/03/14	TimePort-To-Beaglebone-Test1	TimePort_1	Beaglebone_1	2E 4.6	2E 4..	Unknown	Unknown
007	04/03/14	TimePort-To-Software-Test6	TimePort_1	PTPd_Netbook	2E 4.6	2E 2.13	1217	1728
008	05/03/14	TimePort-To-Beaglebone-Test2	TimePort_1	Beaglebone_1	2E 4.6	2E 2.13	Unknown	Unknown
009	05/03/14	TimePort-To-Software-Test7	TimePort_1	PTPd_Netbook	2E 4.6	2E 2.13		
010	05/03/14	TimePort-To-Beagleobone-Test3	TimePort_1	Beaglebone_1	2E 4.6	2E 4.6	Unknown	Unknown
011	06/03/14	TimePort-To-Soft-Test8	TimePort_1	PTPd_netbook	2E 4.6	2E 2.13	1111	1600
012	14/03/14	TimePort-To-Beaglebone-Test4	TimePort_1	Beaglebone_1	2E 3.10	2E 4.6	0900	1700
013	19/03/14	TimePort-To-Beaglebone-Test5	TimePort_1	Beaglebone_1	2E 3.10	2E4.6	0900	1700
014	04/04/14	TimePort-To-Beaglebone-Test6	TimePort_1	Beaglebone_1	2E 3.10	2E 2.13	1000	1300
015	10/04/14	TimePort-To-Beaglebone-Test7	TimePort_1	Beaglebone_1	2E 3.10	Library	1000	1300

5 Awk Script

```
#!/usr/bin/awk -f
#-         Script Name: parseData.awk
#--- Description: strips unnecessary data from the text file and saves it to a new file.
#-         Input: Only input varialbe is RATIO - wich needs to be defined using RATIO=10
#-         First argument is the file to run the script on
#-         Use > filename at the end of the script to pipe the data to the correct output file

BEGIN {
# Checks to see if Ratio is correct (ie greater than 0, not too high, and is an integer
if (RATIO < 0 || RATIO > 1000000) print "Illegal value of ratio. Will default to 10\n" > "/dev/stderr";\
if (RATIO == 0) print "RATIO variable not found. will default to 10" > "/dev/stderr" ;\
if (!(RATIO ~ /^[0-9]+$/)) print "RATIO must be an integer. Defaulted to 10" > "/dev/stderr";
FS = ","; RATIO==10; num=0; sum[0]=0; sum[1]=0; #Sets the file seperator, a default ratio value, and some initial values
printf "# TimeDelta, Master2Slave, Slave2Master\n"
};\
{\
if (NR < 4) next; #Ignores the first three lines
if (num==0) { firstfield = $1}; # Sets the firstfield (time) to the variable firstfield
num = num + 1;\
sum[1] = sum[1] + $4; #Adds the first delay to the first sum
sum[2] = sum[2] + $6; #Adds the second delay to sum
if (num == RATIO) { #If we've added up RATIO number of delays
# --- This section parse
split(firstfield,arrayFirstField," ") #Split old time (at num=0) by space
split($1,arraySecond," ") #Split the new time (at num=RATIO) by space
gsub(/:/," ",arrayFirstField[2]) #Replace all semi colons with a space
gsub(/:/," ", arraySecond[2])#Replace all semi colons with a space
gsub(/-/, " ", arrayFirstField[1])#Replace all dashes with a space
gsub(/-/, " ", arraySecond[1]) #Replace all dasheswith a space
timeDelta = add_ms(arrayFirstField,arraySecond)

printf "%s, %g, %g \n",abs(timeDelta),sum[1]/num,sum[2]/num;\
sum[1] = 0;#reset counters
sum[2] = 0; \
num = 0;\
}; \
lastfield = $1\
}\

END {
  if (num != 0) { #If the number of delays is nonzero, do one final calculation as above

    split(firstfield,arrayFirstField," ")
    split($1,arraySecond," ")
    gsub(/:/," ",arrayFirstField[2])
    gsub(/:/," ", arraySecond[2])
    gsub(/-/, " ", arrayFirstField[1])
    gsub(/-/, " ", arraySecond[1])
    timeDelta = add_ms(arrayFirstField,arraySecond)
    printf "%s, %g, %g \n",abs(timeDelta),sum[1]/num, sum[2]/num
  }

}
function add_ms(time, time2,  delta, delta2) {
  split(time[2],delta, "."); #split the old time by .
```

```
  split(time2[2],delta2,"."); #split the current time by .
  #delta?[2] is the ms difference
  if (int(delta[2]) > int(delta2[2])) {
    # If delta is > delta 2, the time is of the form similar to 3.873 and 4.210.
    # You can't take one away from the other, so I did 1000 - 873, then added on the
        210. (for example)
    return (1000 - int(delta[2]) / 1000 + int(delta2[2]) / 1000)
  } else  return( (mktime(time2[1] " " time2[2]) + (int(delta2[2]) / 1000 ))) - (mktime(
    time[1] " " time[2]) + (int(delta[2]) / 1000));
}
function abs(value)
{
  return (value<0?-value:value);
}
```

Code Extract 11: Awk Script

6 Band Mean

```
#!/usr/bin/env Rscript
#-------------------------------------------------------
#-------------------------------------------------------
#--                 Function Name: bandMean           --
#--                 Name: Band Mean                   --
#--           Input: window - the samples             --
#--                        a   - lower band           --
#--                        b   - upper band           --
#--           Output : mean of window                 --
#-------------------------------------------------------
#-------------------------------------------------------
bandMean <- function(window,a,b){
  sum <- 0
  a <- (a / 100) * length(window)
  b <- (b / 100) * length(window)
  a = round(a) + 1 # 1 indexed
  b = round(b)

  for (i in a:b){
    sum <- sum + window[i] # sum window from a to b
  }
  average <- sum / (b - a + 1)
  return (average)
}
```

Code Extract 12: Band Mean Implementation

7 TDEVAllMethods

```r
#!/usr/bin/env Rscript
#-------------------------------------------------------
#-------------------------------------------------------
#--                 Function Name: All Methods        --
#--                 Name: Time Deviation              --
#--           Input: nTo - position in list           --
#--                  N   - number of samples          --
#--                  x   - vector of samples          --
#--           Output : time deviation                 --
#-------------------------------------------------------
#-------------------------------------------------------
source("bandMean.r")
TDEVAll <- function(To,n, N,x,a,b){
# To <- 0.1 #time between samples
# n <- nTo / To #number of samples to current point
  window <- 35 # Set window Size
  windowSide <- (window - 1) / 2 # Set the length of Side of window
  outerStep <- 0
  interimStep <- c(0,0,0,0)
  outerStep <- c(0,0,0,0)
  result <- c(0,0,0,0)
  for (i in windowSide+1:(N-3*n + 1) - windowSide){
    interimStep <- c(0,0,0,0)
      interimStep[1] <- interimStep[1] + mean(x[(i + (2*n)) - windowSide:(i + (2*n)) +
          windowSide]) - 2* mean(x[i+n - windowSide : i + n + windowSide]) + mean(x[i -
          windowSide : i + windowSide]) #TDEV (mean)

      interimStep[2] <- interimStep[2] + min(x[(i + (2*n)) - windowSide:(i + (2*n)) +
          windowSide]) - 2* min(x[i+n - windowSide : i + n + windowSide]) + min(x[i -
          windowSide : i + windowSide]) #minTDEV
      interimStep[3] <- interimStep[3] + bandMean(x[(i + (2*n)) - windowSide:(i + (2*n))
           + windowSide],a,b) - 2* bandMean(x[i+n - windowSide : i + n + windowSide],a,b
          ) + bandMean(x[i - windowSide : i + windowSide],a,b) #bandTDEV
    interimStep[4] <- interimStep[4] + bandMean(x[(i + (2*n)) - windowSide:(i + (2*n)) +
        windowSide],0,b) - 2* bandMean(x[i+n - windowSide : i + n + windowSide],0,b) +
      bandMean(x[i - windowSide : i + windowSide],0,b)

    interimStep <- interimStep ^ 2
    result <- result + interimStep
  }
  result <- result / (6 * (N - (3*n) + 1))
  result <- sqrt(result)
  return(result)
}
```

Code Extract 13: TDEV All Methods implementation

8 MATIEAllMethods

```r
#!/usr/bin/env Rscript
#-------------------------------------------------------
#-------------------------------------------------------
#--                 Function Name: TDEV               --
#--                 Name: Time Deviation              --
#--           Input: nTo - position in list           --
#--                  N   - number of samples          --
#--                  x   - vector of samples          --
#--           Output : time deviation                 --
#-------------------------------------------------------
#-------------------------------------------------------
```

```r
MATIEAllMethods <- function(To,n, N,x){
# To <- 0.1 #time between samples
# n <- nTo / To #number of samples to current point
  window <- 5 # Set window Size
  windowSide <- (window - 1) / 2 # Set the length of Side of window
  outerStep <- 0
  #MATIE, MAFE, minMATIE, MAFE
  result <- matrix(0,4)
  interimResult <- matrix(0,(N - 2*n + 10),4) #fudged because of the 4 truncated rows.
      will fix what N is.
  interimStep <- matrix(0,2) #only need MATIE and minMATIE
  for (i in 1:(N-2*n + 1)){
    interimStep <- c(0,0,0,0)
    for (j in i + windowSide:(n+i-1) - windowSide){
      interimStep[0] <- interimStep[0] + mean(x[i+n - windowSide: i + n + windowSide]) +
          mean(x[i - windowSide : i + windowSide])
      interimStep[1] <- interimStep[1] + min(x[i + n - windowSide : i + n + windowSide])
          + min(x[i - windowSide : i + windowSide])
    }
    for (k in 1:2){

      interimStep[k] <- abs(interimStep[k]) / n
    }
    interimResult[i,1] <- interimStep[1]
    interimResult[i,2] <- interimStep[1] / (n * To)
    interimResult[i,3] <- interimStep[2]
    interimResult[i,4] <- interimStep[2] / (n * To)
  }
  result[1] <- max(interimResult[1])
  result[2] <- max(interimResult[2])
  result[3] <- max(interimResult[3])
  result[4] <- max(interimResult[4])
  return(result)
}
```

Code Extract 14: MATIE All Methods implementation

9 TDEV All Methods in C

```c
//--------------------------------------------------------
//--------------------------------------------------------
//--                Function Name: TDEV                 --
//--                Name: Time Deviation                --
//--           Input: nTo - position in list            --
//--                  N   - number of samples           --
//--                  x   - vector of samples           --
//--           Output : time deviation                  --
//--------------------------------------------------------/
//--------------------------------------------------------
#include <math.h>
#include <stdio.h>
#include <R.h>

void TDEVAllMethods(int To, int* n, int* N ,double * x,double * tempResultTDEV,double *
    tempResultMinTDEV, double * tempResultBandTDEV, double * tempResultPercTDEV){
  int window = 5; // Set window Size
  int windowSide =  (window - 1) / 2; // Set the length of Side of window
  double outerStep[4] = {0.0,0.0,0.0,0.0};

  double interimStep[4] = {0.0,0.0,0,0.0};
  double average[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
  //int i = 0;
```

```
  int a = 1;
  int b = 3; //0 indexed
  double minimum[3] = {10000.0,10000.0,10000.0}; //large enough value so it won't be the
      minimum
  for (int i=windowSide; i <= (*N -(3 * (*n))) - windowSide + 2; i++){
    for (int k=0;k < window ;k++){
      average[0][0] = average[0][0] + x[i + 2*(*n) - windowSide + k - 1];
      average[0][1] = average[0][1] + x[i + (*n) - windowSide + k - 1];
      average[0][2] = average[0][2] + x[i - windowSide + k - 1];
      if (x[i + 2*(*n) - windowSide + k - 1] < minimum[0]) minimum[0] = x[i + 2*(*n) -
          windowSide + k - 1];
      if (x[i + (*n) - windowSide + k - 1] < minimum[1]) minimum[1] = x[i + (*n) -
          windowSide + k - 1];
      if (x[i- windowSide + k - 1] < minimum[2] ) minimum[2] = x[i - windowSide + k -
          1];

    }
    for (int bm = a; bm < b + 1; bm++) {
      average[1][0] = average[1][0] + x[i + 2*(*n) - windowSide + bm - 1];
      average[1][1] = average[1][1] + x[i + (*n) - windowSide + bm - 1];
      average[1][2] = average[1][2] + x[i - windowSide + bm - 1];
    }
    for (int pm = 0; pm < b + 1; pm++){
      average[2][0] = average[2][0] + x[i + 2*(*n) - windowSide + pm - 1];
      average[2][1] = average[2][1] + x[i + (*n) - windowSide + pm - 1];
      average[2][2] = average[2][2] + x[i - windowSide + pm - 1];
    }

    for (int j = 0; j < 3; j++) {
      average[0][j] = average[0][j] / window;
      average[1][j] = average[1][j] / (b - a) + 1;
      average[2][j] = average[2][j] / (b + 1);
    }

    interimStep[0] = average[0][0] - (2 * average[0][1]) + average[0][2];
    interimStep[1] = minimum[0] - (2 * minimum[1]) + minimum[2];
    interimStep[2] = average[1][0] - (2 * average[1][1]) + average[1][2];
    interimStep[3] = average[2][0] - (2 * average[2][1]) + average[2][2];
    for (int j = 0; j < 4; j++) outerStep[j] = outerStep[j] + (interimStep[j] *
        interimStep[j]);

    for (int i = 0; i < 4; i++) average[i][0] = average[i][1] = average[i][2] = 0;
    minimum[0] = minimum[1] = minimum[2] = 10000.0;
    interimStep[0] = interimStep[1] = interimStep[2] = interimStep[3] = 0;
  }
  *tempResultTDEV = sqrt(outerStep[0] / (6 * (*N - (3)*(*n) + 1)));
  *tempResultMinTDEV = sqrt(outerStep[1] /  (6 * (*N - (3)*(*n) + 1)));
  *tempResultBandTDEV = sqrt(outerStep[2] /  (6 * (*N - (3)*(*n) + 1)));
  *tempResultPercTDEV = sqrt(outerStep[3] /  (6 * (*N - (3)*(*n) + 1)));



}
```

Code Extract 15: TDEV All Methods Implementation in C

10 MATIE MAFE All Methods in C

```
//-----------------------------------------------------
//-----------------------------------------------------
//--                  Function Name: TDEV              --
```

```
//--                  Name: Time Deviation                --
//--            Input: nTo - position in list             --
//--                     N   - number of samples          --
//--                     x   - vector of samples          --
//--            Output : time deviation                   --
//-----------------------------------------------------/
//-----------------------------------------------------
#include <math.h>
#include <stdio.h>
#include <R.h>

void MATIEAllMethods(double *To, int* n, int* N ,double * x,double * tempResultMATIE,
    double * tempResultMAFE, double * tempResultMinMATIE, double * tempResultMinMAFE){
  int window = 5; // Set window Size
  int windowSide =  (window - 1) / 2; // Set the length of Side of window
  double minimum[2] = {10000,10000};
  double interimStep[2][*N - (2 * (*n)) + 10];
  double average[2] = {0.0,0.0};
  for (int i=0; i <= (*N -(2 * (*n)))  + 1 -1; i++){
    for (int j=i; j < *n + i - 1 - 1 ; j++) { // - 1 as normal. - 1 for index difference
      for (int k=0;k < window ;k++){
        average[0] = average[0] + x[j + (*n) - windowSide + k - 1];
        average[1] = average[1] + x[j - windowSide + k - 1];
        if (x[j + (*n) - windowSide + k - 1] < minimum[0]) minimum[0] = x[j + (*n) -
            windowSide + k - 1];
        if (x[j- windowSide + k - 1] < minimum[1] ) minimum[1] = x[j - windowSide + k -
            1];

      }
    }
    for (int j = 0; j < 2; j++) {
      average[j] = average[j] / window;
    }

    interimStep[0][i] = (average[0] - average[1]) < 0 ? -((average[0] - average[1]) / *n
        ) : ((average[0] - average[1]) / *n);
    interimStep[1][i] = (minimum[0] - minimum[1]) < 0 ? -(minimum[0] - minimum[1]) / *n
        : (minimum[0] - minimum[1]) / *n;
    average[0] = average[1] = 0;
    minimum[0] = minimum[1] = 10000.00;
    if (interimStep[0][i] > *tempResultMATIE) *tempResultMATIE = interimStep[0][i];
    if (interimStep[1][i] > *tempResultMinMATIE) *tempResultMinMATIE = interimStep[1][i
        ];
    *tempResultMAFE = (double) *tempResultMATIE / (*n * (*To));
    *tempResultMinMAFE = (double) *tempResultMinMATIE / (*n * (*To));

  }

}
```

Code Extract 16: MATIE/MAFE All Methods Implementation in C

11 Main Packet Metric Script

```
#!/usr/bin/env Rscript

# ------------------------------------------------------
# -          Script Name: PacketMetric.r               -
# -        Description: This script will calculate     -
# -        the packet metrics for the given data set. -
# ------------------------------------------------------
# ----------------------------------------
# - Import required functions + libraries -
# ----------------------------------------
#Rprof(filename = "RProf.out", memory.profiling = TRUE, gc.profiling=TRUE,line.profiling
    = TRUE)
system("clear") #Clear screen
options(warn = 1) # Enables warnings
cat("Loading Required Scripts..\n")
suppressPackageStartupMessages(library("argparse"))
suppressPackageStartupMessages(library("gnumeric"))
library("logging")
source("../LaTeXScripts/generateLatexTable.r")
source("TDEV.r")
source("minTDEV.r")
source("TDEVAllMethods.r")
source("MATIEAllMethods.r")
source("FuncsPacketMetric.r")
source("SimpleOperations.r")
dyn.load("TDEV.so") # C function
dyn.load("TDEVAllMethods.so")
dyn.load("MATIEAllMethods.so")
cat("Loaded Required Scripts\n")
startTime <- proc.time()
options(scipen=9)
# ----- Initialises Variables -----
sampleSize <- 0
directory <- ""
fileName <- ""
index <- 0
parser <- createArguments()
# ----- Parses the arguments and checks to see if they are valid -----
args <- parser$parse_args()
sampleSize <- args$sampleSize
directory <- args$directory
NTEST <- args$nTest
start <- args$start
ratio <- args$ratio
initLogger()
if (args$interactiveMode == TRUE) {
  vars <- interactiveMenu()
  sampleSize <- vars$nLines
  nTest <- vars$nTest

}
if (directory == "None" && args$loadDelays == "None") {
  if (NTEST  == -1 && args$AllSampleSize == "None"){
    logerror("Error 1: You need either a directory or the test number\n Program will
        exit. \n")
    return (1)
  }
}
if (sampleSize <= 0 || args$AllSampleSize != "None" ){
  logwarn("Default sample size of 50 will be used\n")
  args$sampleSize <- 50
  sampleSize <-args$sampleSize
```

```
  # ---- Note: Strange that I need to do the above. will investigate. possibly to do
      with deep/shallow copying?
}
if (start > 0 && start < 10000) {
  sampleSize <- sampleSize + start
}
if (args$AllSampleSize!="None") {
  sampleSize <- args$AllSampleSize
  tests <- seq(0,9)
} else tests <- NTEST
ifLoad = FALSE
for (nTest in tests) {
  loginfo(paste("Running script on Test number", nTest))
  if (args$loadDelays != "None") {
    Data <-readFileDirect(paste("/home/james/FinalYearProject/PTPData/TestData/", args$
        loadDelays, sep=""))
    ifLoad = TRUE
  } else {
    tempResult <- parseFileName(nTest, args$directory, args$direction)
    fileName <- tempResult$fileName
    nTest  <- tempResult$nTest
    index <- tempResult$index
    testSheet <- tempResult$testSheet
    fileNameConv <- tempResult$fileNameConv
    if (args$convert) {
      loginfo(paste("Data file is being converted to the new format, and using a",ratio,
          "point average"))
      print(paste("File Name: ", fileName))
      runHead(sampleSize, "ExampleData") #Hard Coded need to fix!
      fileName <- convertData(ratio, fileName) #changes filename to the new file


    }
    Data <- readFile(fileName, nTest, sampleSize, testSheet)
  }
  if (dim(Data)[0] ==1 && Data == 2) return (2) # Returns out of the entire script if an
      error is thrown in the previous function
#
    ------------------------------------------------------------------------------------

#### - May need to be dynamic To ? or at least an option to change / work out #####

  To <- 1/32 #Assume To = 1/16
  dataPacket <- purgeData(Data,ifLoad)
  delays <- dataPacket$delays
  time <- dataPacket$time
  N <- dataPacket$N
#---- Currently To == Time, but will sort out once I understand what to do with To ----
#----- As Delay variable has been written, it can now be plotted into histograms ------

  if (args$hist) {
    plotHistogram(delays)
  }

  if (args$chist) {
    plotCHistogram(delays)
  }

  if (args$pdelay) {
    plotDelay(delays)
  }
  #plotDiff(delays, Data[6])
  print("OK")
  results <- calculateMetrics(To, N, delays)
  ResultTDEV <- results$ResultTDEV
```

```
  ResultMATIEMAFE <- results$ResultMATIEMAFE
  plotArray(ResultTDEV,0)
  plotArray(ResultMATIEMAFE,1)

  if (args$save) {
    fname = paste("../PTPData/DelayData/Data : Test: ", nTest, "Sample Size:", N, ".txt"
        )
    fname2 = paste("../PTPData/MetricData/TDEVData : Test: ", nTest, "Sample Size:", N,
        ".txt")
    fname3 = paste("../PTPData/MetricData/MATIEData : Test: ", nTest, "Sample Size:", N,
         ".txt")
    write.table(delays,file=fname,sep="\t", col.names = F, row.names = F)
    write.table(ResultTDEV, file=fname2,sep="\t", col.names = T, row.names = F)
  write.table(ResultMATIEMAFE, file = fname3, sep="\t", col.names = T, row.names = F)
  loginfo("Data written to file")
  }

  if (args$stats) {
    stats <- calculateStats(delays)
    tabulateStats(stats)
  }

  result <- generateResultArray(ResultTDEV, ResultMATIEMAFE)
  error <- outputTable(result,args$CSV, args$latex)
  loginfo(paste("Total Run Time: ", round(proc.time()[1] - startTime[1],3)))
  loginfo(paste("Total memory requirement: ", format((object.size(x=lapply(ls(), get))),
      units="KB")))
}
```

Code Extract 17: Main Packet Metric Script

12 Function Library

../PacketMetricsFuncsPacketMetric.r

13 Plotting Function - Line

```
plotArray <- function(values,whichPlot) {
  #Global Vars: args$nTest, N,
  rangeOfValues <- range(0,values) #Determines a max range for the plot
  if (whichPlot == 0) metric = "TDEV" #TDEV
  else metric = "MATIE-MAFE"
  outputFileName = paste("../PTPData/Plots/Test: ", args$nTest, " - ", metric, " - ", N,
      " size.eps",sep = "")
  postscript(outputFileName)
  plottingColours = rainbow(ncol(values))
  plot(values[,1], type='o',xaxt='n', yaxt='n',pch='+',col=plottingColours[1],log="xy",
      xlab="", ylab="")
  for (i in 2:ncol(values)) {
    #if (values[1,i] == 0) {
    # loginfo("Column Ignored")
    # continue
    #}
    lines(values[,i], type='o',pch='+',col=plottingColours[i])
  }
  legend(1,max(values[,2:ncol(values)]),colnames(values) , col=plottingColours, cex =
      0.8,lty=1, pch='+', title="Metrics Legend",box.lwd = 0,box.col = "white",bg = "
      white")
  if (metric == "TDEV") title(main=paste("Packet Metrics - TDEV - Sample Size",
      sampleSize),ylab="", xlab="Index/Time")
```

```
  else title(main=paste("Packet Metrics - MATIE-MAFE - Sample Size", sampleSize),ylab=""
      , xlab="Index/Time")
  mtext(side = 2, text="    Index/Time", line = 3)
  axis(side = 1)
  axis(side = 2, las = 1)
  dev.off()
  loginfo(paste(metric, "Plot Created"))
}
```

Code Extract 18: Plotting Function for Line Graphs

14 Plotting Functions - Histograms

```

plotHistogram <- function(data) {

  outputFileName = paste("../PTPData/Plots/HistogramsOfDelays/Histogram of Delays - Test
      :", args\$nTest, " Sample - N", N, " size.eps",sep = "")
  postscript(outputFileName)
  hist(data, xlab = "Bins of Delay", ylab="Frequency", main = )

}
```

15 Plotting Functions - Delay Plot

```

plotDelay <- function(delay) {

  outputFileName = paste("../PTPData/Plots/PlotOfDelays/Plot of Delays - Test:", args\$
      nTest, " Sample -", N, " size.eps",sep = "")
  postscript(outputFileName)
  plot(delay * 1000, pch = 16, cex = .9, xlab = "Sample Number", ylab = "Delays (ms)")
}
```

16 Plotting Functions - Colour Histogram

```

plotCHistogram <- function (delay) {
  # Work in Progress. Comment out all lines until it is completed
  #outputFileName = paste("../PTPData/Plots/PlotOfDelays/Colour Histogram of Delays -
      Test:", args\$nTest, " Sample -", N, " size.eps",sep = "")
  #postscript(outputFileName)
  step <- 32 * 60
  j <- 0
  #delay <- rnorm(n=5000,m=24.2,sd=2.2)
  # -- Flatten the histogram into their corresponding colours.
  nStep <- floor(length(delay) / step ) # 100 steps for the time being.
  colouredDelayArray = matrix(0,nStep,100) #temp matrix. need to define size
  bins <- seq(0.00000, 0.00010, by = 0.00005)

  for (i in seq(1, length(delay),by=step))  {
    if (is.na(delay[(i+step - 1)])) break
    histogram <- hist(delay[i:step + i - 1],breaks=30)\$counts
```

```
17      colouredDelayArray[j,1:length(histogram)] <- histogram
18      j = j + 1
19    }
20    png("simpleIm.png")
21
22    image(log(t(colouredDelayArray[,1:30])), col=heat.colors(30,alpha=1))
23
24  }
```