Final Year Report - Investigation into the Precision Time Protocol

James Cox
Department of Electrical and Electronic Engineering

University of Bath

April 17, 2014

Abstract

Abstract goes here

Contents

Acronyms

BMC Best Master Clock

CSAC Chip Scale Atomic Clock

CSMA/CD Carrier Sense Multiple Access with Collision Detection

CSV Comma Seperated 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

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

TDEV Time Deviation

UTC Coordinated Universal Time

List of Figures

List of Tables

*Listings

1 Introduction

In a few applications there is an ever increasing requirement for high accurate clocks. These clocks mmay be used for either a timestamp, such as in the telecommunications industry, or a way to synchronise processes, as is what happens in the automotive industry. Multiple clocks will be used in these applications, and thus they all should be synchronised with one another.

In some applications it is not feasible to have a high accuracy atomic clock (or Chip Scale Atomic Clock (CSAC)) due to space or cost constraints. An alternative to this could be to use a Global Positioning System (GPS) receiver and time can then be synchronised to the accurate clocks onboard the GPS satellites. Using high accuracy clocks like the ones mentioned would help to reduce any clock accuracies.

There are two types of clock inaccuracies when synchronising clocks. Firstly they may have started at a different time relative to the others, and adjusting for this is called offset correction. The second effect is that clocks do not necessarily run at exactly the same speed. Therefore clocks 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 would require highly accurate clocks.

Automation Industry
Processes will need to be synchronised exactly, and can only be 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. [6]

Power Transmission
Time synchronisation is very important in the power transmission industry. An example of a situation where timing would have mitigated an event from occurring is the North American blackout in August 2003 [7]. 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 [8] 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 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. But if there is an issue with the system, for example a jamming incident, then there will be some major consequences should timing drift out of line. It is known that jamming of GPS receivers is becoming more common and thus an alternative method of time synchronisation should be used. [9]

One way of realising this is by using a distributed timing system, such as NTP or PTP. This would allow for nodes to be able to synchronise their clocks with a much more accurate time source without having to rely on GPS

1.1 How NTP Works

This is a technology originally designed in 1985 and is used to synchronise clocks over a packet switched network. It is able to achieve synchronisation with UTC within a few milliseconds, but 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 decrease this accuracy by 100ms or more. [10]
NTP uses a client-server hierarchy split into ”Stratums”. Figure 1 on the next page shows the Stratums numbered from 0 to 3.

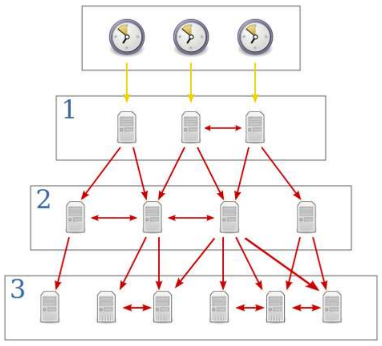


Figure 1: NTP Network Hierarchy [4]

The reference clocks located in Stratum 0 are high precision such as atomic clocks or they use GPS synchronisation. The clocks in Stratum 2 will base their time off of the clocks in Stratum 1. A number 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 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 [11].
Simple Network Time Protocol (SNTP) is used in applications which do not require a high timing accuracy. It does this by ignoring drift values. Therefore it is recommended that SNTP is only used in the higher Stratums [12]. The SNTP specification is part of the NTP specification, cited here [13].
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 sub millisecond accuracies. 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 PTP 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 PTP Works

PTP was developed as a successor to the existing NTP standard which aims to reach sub millisecond accuracies. 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 PTP as this packet delay must be taken into account when working out the clock offset. One part of this project will be to investigate the extent of this variation in packet delay on an ethernet network.

There are a few important elements of how PTP works in order to be able to start the project. These are: general PTP operation and the BMC algorithm. There are other aspects that will be covered in this report (such as token passing and Secure PTP), but these will be covered in a later section. A comparison with NTP, a similar technology will also be made. This will be important when considering some security aspects of the technology.

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 this can occur up to one message every 32 and a quarter milliseconds.

The list below shows the basic steps that PTP follows [14]:

1. Broadcast begins at t_1 where the master sends a *sync* message to all clocks on the domain.

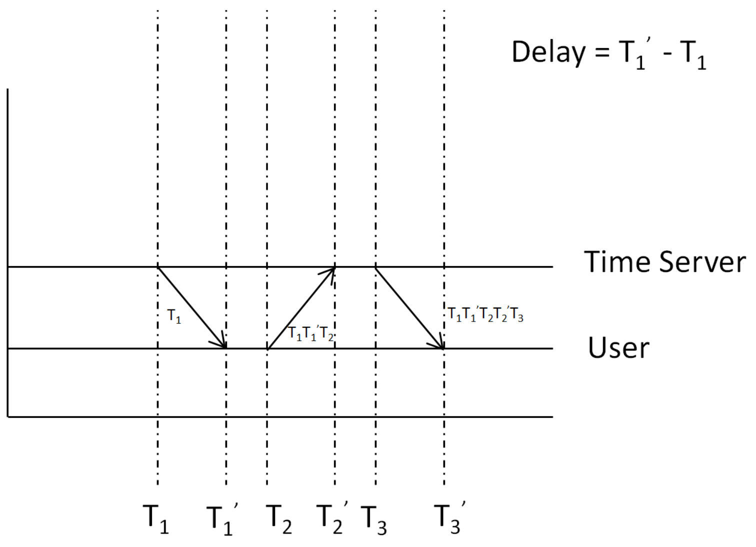


Figure 2: PTP timing diagram (based on [5])

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 that 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:

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

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

If we rearrange equation 2 for d :

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

Substituting equation 3 into 1:

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

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

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

(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 a short period of time that the delay is 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).

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. Version 1 improvement improved on NTP in a number of different ways. A new PTP standard was introduced in 2008 with some new features such as boundary clocks and using domains instead of stratums. 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	**TAI!** (**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 / Clock Class
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

There are several different configurations for a PTP network to take using the following pieces of hardware.

Grandmaster Clock The grandmaster clock is the main source of time synchronisation using PTP 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 detailed in the next section.

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.

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. These 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.

There are several different network configurations that could be used when using a PTP network. The diagram below outlines some possible topologies for PTP networks.

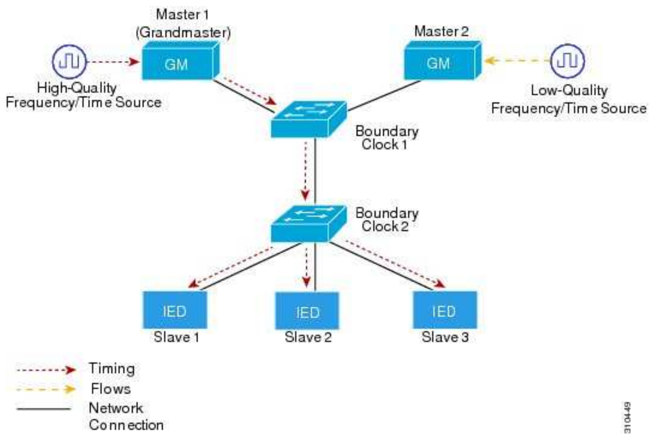


Figure 3: A Typical PTP Network

(a) One Grandmaster Multiple Slaves (b) One Grandmaster Multiple Slaves with Boundary Clock (c)

Figure 4: Examples of some PTP Networks

1.2.4 Best Master Clock (BMC)

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

Identifer 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 admin assigned precedence to select a particular grandmaster. In IEEE1588-2008 there are 2 8bit priority fields.

Variance This refers to the stability of the clock based on itsd performancr 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.

1.2.5 Management Commands

- explanation of the management portion of PTP + explain the distributed maangement structure etc

1.2.6 Issues

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

1.3 Project Description

This project aims to investigate PTP performance on a heavily used Ethernet network, and to attempt to quantify PTP performance using 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 involved in order to complete them have been detailed in the Gantt chart, as seen in Appendix 2.

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.

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 some 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
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 in case the hardware documentation is limited.

Investigate into security measures
Some research into how to secure PTP will be made along with some recommendations in possible.

2 Literature Review

With the following objectives and tasks in mind, a literature review was performed with some suitable documentation: mainly packet metric related, but also on cryptographically signing PTP packets.

The reports below will be discussed in some detail:

Definitions and terminology for synchronization in packet networks [1] A standard regarding different packet metrics that could be used in order to try and quantify network delay.

Prevention of Packet Collisions [15] A journal article describing an algorithm that aims to prevent packet collisions in an Ethernet network.

2.1 Definitions and Terminology for Synchronisation in Packet Networks [1]

The first paper 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 Methods

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 [1]. 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).

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. These will be discussed in turn and will be implemented.

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 Packet Metrics with Pre-filtering

The other method is using pre-filtering before the metric is calculated. An averaging function is applied to the set of data, but care must be taken to not over-filter the input. This filtered packet sequence can then applied to the metrics mentioned previously in the report. Prefiltered metrics are useful as they can help specify network limits.

2.2 Other Relevant Reading

There was other relevant reading performed in the first week of the project to do with cryptography and how packet collisions can be prevented. -cryptography ?

3 Project Methods

Based on the objectives mentioned previously, the project can be split into some distinct sections:

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

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

Calculating/Analysing Results Once the metrics have been implemented fully, there needs to be some supplementary scripts written to process some of this data.

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

Methods to reduce Packet Collisions Due to the way Ethernet networks work, there is a high probability of packet collisions. 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 identifed as being available to use for the duration of this project.

- Hardware Grandmaster - Chronos TimePort [2]
- Hardware Slave - Chronos Syncwatch [**?**]
- Hardware Slave - Beaglebone Black [3]

(a) Chronos TimePort Outside

Figure 5: Chronos TimePort Labelled Diagrams

- Software Grandmaster - PTP Daemon (PTPd)
- Software Slave - 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.

4.1 Hardware - Timeport [2]

4.1.1 Description

The Chronos CTL4540 Timeport 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 without needing to be connected to GPS. It also has an internal LiPo battery. This enables the device to be used to transport and measure time.

With the above features in mind, it is thus suited for a number of 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 involve using a Caesium atomic clock [REF] or setting up a GPS attenna and connecting this to some other equipment. The TimePort is best suited over these two operaitons because it is much lower power and much more transportable than an atomic clock. It also removes the requirement of GPS equipment.

Appendix **??** shows the full specifications of the CTL4540 TimePort. Below are a few labelled photos of the clock.
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. The university has close links with Chronos thus it should be straightforward to either get our issues solved or to receive a new TimePort.

This clock will mainly be kept in the same position on the network and will acct as a Grandmaster.

In terms of documentation there is not much available for this device apart from some emails sent between Chronos and Dr Robert Watson. Therefore Appendix **??** shows some documentation put together for my own use during this project. The documentation includes details on how to interface with the clock and a list of basic commands.

To access the device it needs to be accessed locally over a USB to Serial connection. SSH is unavailable as the control port has not been implemented yet.

4.1.2 How to Set Up the TimePort

The firmware version that the TimePort we have available does not have the control port active. Therefore the only method to connect to the TimePort is via a serial connection.

To connect to the TimePort: connect the serial to USB cable (seen pictured below, Figure **??**).

Figure 6: USB to Serial Converter

Type the following command into a linux terminal:

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

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

This will connect you to the first layer of the TimePort. The system is a restricted linux distribution, so some commands you may be familiar with do exist. A full list of commands and the rest of the documentation can be found in Appendix 4. Figure 7 is a screenshot of this first layer
A complete list of commands can be found in the TimePort documentation that has been written (Appendix 4.

Figure 7: TimePort Access

(a) Chronos Syncwatch Outside

(b) Chronos Syncwatch Inside

Figure 8: Chronos Syncwatch Labelled Diagrams

4.2 Hardware - Chronos Syncwatch [?]

4.2.1 Description

The Chronos Syncwatch 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, Frequencies(64k-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 modular 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 show labelled diagrams of a block diagram and the outside of the Chronos Syncwatch.

This product is similar to the TimePort in the fact that there isn't much documentation around for it. Therefore Appendix 5 shows the documentation written up for the Syncwatch.

The Syncwatch will be mainly kept in the upstairs Level 3 Communications lab as it is a larger device. As this device is a release product, all of the ports used for controlling the device are enabled. Therefore the syncwatch can be set up via SSH or using the program. This is all explained in the documentation in Appendix 5.

4.2.2 How to Set Up the 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 at the same time.

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

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

using password: *syncwatch* .

You are now in the first layer of the Syncwatch device. The alternative method to access this same terminal window is by using a USB to Serial converter, as pictured earlier, Figure **??**.
Plug in the device to a computer, and connect to it using the following command:

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

The username is root and the password is syncwatch. At this point the first layer has been connected to. This brings up the following prompt (shown in Figure **??**.

Figure 9: Syncwatch First Layer Screen

Figure 11: Labelled BeagleBone Black

To access the PTP console, type the following into the screen connection:

```
minicom -S
```

This then brings up the 2nd layer : the PTP console. The screenshot below (Figure **??** shows a list of commands available. The complete list of commands in both of the modes is shown in the documentation, shown in Appendix 5.

Figure 10: Syncwatch Second Layer Screen

4.2.3 Hardware - Beaglebone Black [3]

4.2.4 Description

The Beaglebone Black is a hardware device but it is running a software PTP Daemon (called PTPd). Throughout this report the Beaglebone Black will be called a hardware clock, but in reality it is running a PTP software implementation.

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.

As it runs Linux and can be connected to the network, 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 11 is a labelled picture of the 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 How to Set up the Beaglebone Black

When the project was started, Robert Watson had the BeagleBone Black working with PTPd already, so there was only some work to be done in order to automate the process.

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.

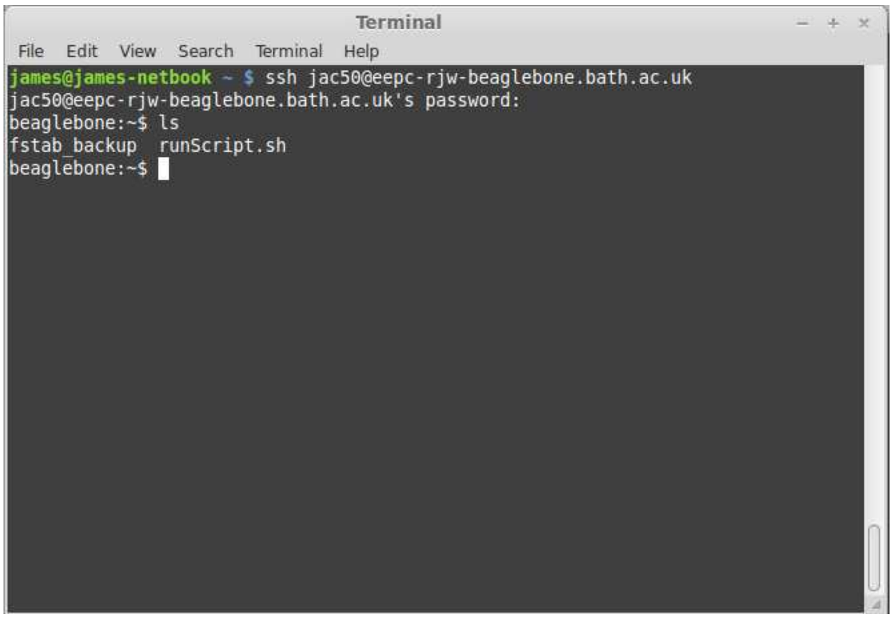
Type:

```
ssh jac50@eepc-rjw-beaglebone.bath.ac.uk
```

to log in, replacing jac50 with the username on the device. The users on the device were eerjw, jac50, and root.

The screenshot below (Figure 12) demonstrates this. The ls command was typed to show that the connection was successful.
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 on 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 this 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



```
james@james-netbook ~ $ ssh jac50@eepc-rjw-beaglebone.bath.ac.uk
jac50@eepc-rjw-beaglebone.bath.ac.uk's password:
beaglebone:~$ ls
fstab_backup  runScript.sh
beaglebone:~$ 
```

Figure 12: SSH to Beaglebone

of *timeport_YYYY_MM_DD.txt* was decided.

The full bash script is shown below, in Code Extract 2

```
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 2: 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 mmcblk1p1, 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 5.

Once the test is completed, the script can be killed by using *kill* − 9 on ptpd2. 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.

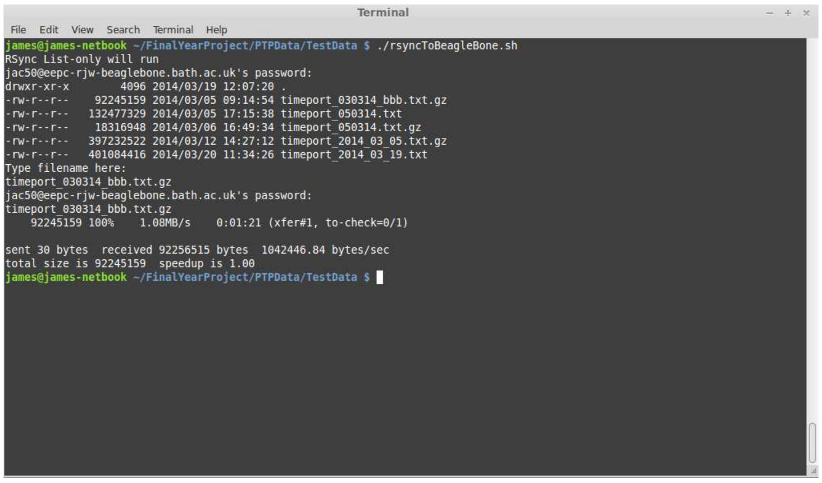
4.2.6 Sending Data to the Local Machine

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 3 below is the rsync script used.

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

Code Extract 3: Rsync Script



```
james@james-netbook ~/FinalYearProject/PTPData/TestData $ ./rsyncToBeagleBone.sh
RSync List-only will run
jac50@eepc-rjw-beaglebone.bath.ac.uk's password:
drwxr-xr-x        4096 2014/03/19 12:07:20 .
-rw-r--r--    92245159 2014/03/05 09:14:54 timeport_030314_bbb.txt.gz
-rw-r--r--   132477329 2014/03/05 17:15:38 timeport_050314.txt
-rw-r--r--    18316948 2014/03/06 16:49:34 timeport_050314.txt.gz
-rw-r--r--   397232522 2014/03/12 14:27:12 timeport_2014_03_05.txt.gz
-rw-r--r--   401084416 2014/03/20 11:34:26 timeport_2014_03_19.txt
Type filename here:
timeport_030314_bbb.txt.gz
jac50@eepc-rjw-beaglebone.bath.ac.uk's password:
timeport_030314_bbb.txt.gz
    92245159 100%    1.08MB/s    0:01:21 (xfer#1, to-check=0/1)

sent 30 bytes  received 92256515 bytes  1042446.84 bytes/sec
total size is 92245159  speedup is 1.00
james@james-netbook ~/FinalYearProject/PTPData/TestData $
```

Figure 13: Rsync Example

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 **??** shows the rsync script transferring across a gzipped data file.
It was important to gzip the file beforehand or the transfer would have taken a lot longer. The rate of data collection is around RATE OF DATA HERE

4.3 Software - 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.

In-depth code analysis of the script will not be provided in this report. Instead the different flags that may be used for this project will be tabulated below (Table 2).

Table 2: Flags used for PTPd

Flag Name	Flag Letter V2.3.0 or above	Old Flag Letter	Description
Interface	i	b	Network Interface to use
Domain	d	i	PTP Domain Number
Foreground	C	C	Run program in foreground
Verbose	V	None	Run in Verbose mode
No Clock Adjust	n	t	Do not adjust the local clock
Slave only mode	s	g	Set PTPd as Slave

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, but some software grandmaster clock tests may be performed.
The script call has already been given for the beaglebone. The code extract below has been used for the PTPd_Netbook

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

The difference in the call to ptpd2 above is that 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. Note that the above script in Extract 4.3 is already run as a root user. The other method to do this would be to run the script with sudo.

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. This is a local address which isn't 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	...
002	Chronos SyncWatch	Hardware	syncwatchptp	syncwatch	00:16:C0:17:20:A1
003	BeagleBone Black	Hardware	beaglebone	beaglebone	...
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 IPs listed are just part of the full name. To access one of them on the university network, add the prefix eepc-rjw- and the suffix .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	Differnet subnet
8W Rooms	8W 2...	Lecture room	No	Different subnet
East building	EB 2..	Lecture room	No	Differnet subnet

The full list of clock locations have been placed on the map below, Figure **??**

Figure 14: University Map with clock locations labelled

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. A network topology map has also been added, see below Figure **??**.

Figure 15: Network Topology Map

4.6 Test Sheets

As there will be quite a few 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 6

Each test 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 identifer 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 come up will be noted here.

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 will include the Test name, date, directory, and start and end times of the tests. This will show up later in the report, in Appendix **??**.

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 like 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 identifed as important tests to run have been included in the table below.

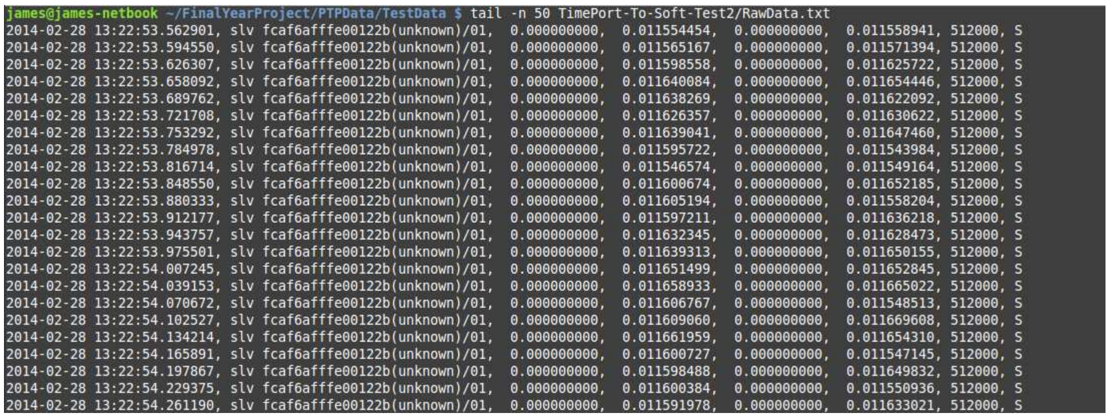
Table 5: Some Tests to Run

GM Clock Type	GM Clock Location	Slave Clock Type	Slave Clock Location	Duration
Hardware (TimePort)	Watson's Office	Hardware (Syncwatch)	2E 3.Comms lab	24 hours
Hardware (TimePort)	Watson's Office	Software (PTPd)	Watson's Office	24 hours
Hardware (TimePort)	Watson's Office	Software (PTPd)	2E 2... 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 and for 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 for the scripts.



```
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 16: Example of the File Output

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 16.
This data is formatted as a Comma Seperated 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.###### type format, with the hashes denoting the decimal after the seconds field.

State The state is what state the clock is in, 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. [16].

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

Last Packet Received This is a single character which is the type of packet received. 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 quite a bit of excess data not required. Assuming that there will be 32 of these messages every second, the RAM 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 arbitary value will also be added into this script.
Because this was a preprocessing step, it is important to try and keep the execution time of this script to 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 [17].
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[18]. 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). Because it's 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 identifed 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, 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 **??**.

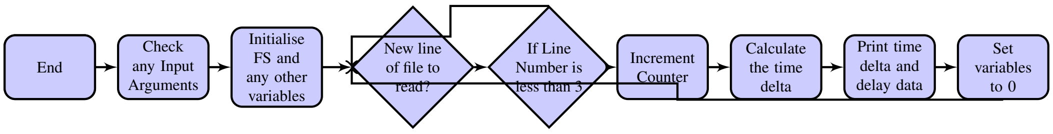

End
Check any Input Arguments
Initialise FS and any other variables
New line of file to read?
If Line Number is less than 3
Increment Counter
Calculate the time delta
Print time delta and delay data
Set variables to 0

Figure 17: Awk Script Flow Chart

This flowchart was then used to create the final awk script, shown in Appendix 8. 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.

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 *mktime* to be called. Therefore the following steps were performed to convert the time to the correct format.

Regex Before the function is called, a regular expression substitution is used to replace all colons with spaces and all dashes to spashes. This was important because the end goal is to convert the current time format into one seperate by spaces.

Split Seconds portion by full stop This step would parse the milliseconds away from the seconds.

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.

Calculate delta Convert time using mktime and take the differences between the delays to work out a delta. Return this difference.

This set of code was tested quite 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 Methods

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
- 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.

Note that parts of this section has been copied from the Individual Technical Report for the Third Year Group Business and Design Project as there are some parts that are applicable.

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 may be acceptable however if there is a much better language ssuited 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 so **[REQ7]** can be met. 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
Does the language support complex plotting as standard or are external libraries required?

REQ4 - External libraries already available
Spec: All available
Some external libraries may be needed in case some specific functionality is required. Examples of external libraries that will be 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]**	6	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]**	6	Yes, it is compatible.	No. Windows Only
Development Time	How long would it take to develop the first program	**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.

Table 8: Language Options Ranking Tables

Programming Language				
Ranking Criteria	Weight	Language		
		R	Matlab	C
Familiarity with the Language	9	18	36	45
Plotting Functionality	8	40		0
External Libraries Available	7	28	21	14
Speed	7	14	14	35
Linux Compatibility	7	35	21	35
Development Time	7	21	28	14
Documentation	6	24	24	12
Total Figure of Merit		180	176	195

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.

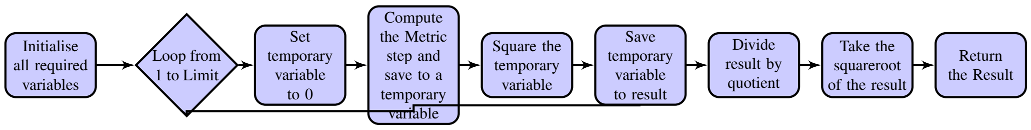
R is a programming language that used primarily for statistical computing. It is similar to the S programming language but under the GNU's Not Unix (GNU) umbrella. R has many strengths in statistics due to the wide range of libraries available to it. In addition to this it has extensive plotting functionality,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

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 run 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 thus this will be exploited where possible.

Therefore 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 **??**)



Initialise all required variables
Loop from 1 to Limit
Set temporary variable to 0
Compute the Metric step and save to a temporary variable
Square the temporary variable
Save temporary variable to result
Divide result by quotient
Take the squareroot of the result
Return the Result

Figure 18: General Packet Metric Flowchart

The flow chart above has been converted into psuedo code, seen below.

```
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 2̂
    result += interimStep
end
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 and has been discussed in the previous section.

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

5.3 TDEV and TDEV derivatives

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

The four metrics that will be implemented were: TDEV, minTDEV, bandMeanTDEV and percentileTDEV.

TDEV The TDEV script requires means to be taken of the individual slices. 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 7. This can also be found in Appendix 9.

```
#!/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 4: 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 10 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 that 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 MATIE 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's 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 11.

5.5 Overall Packet Script

Once the packet metric functions were created, these will need to be both called from a seperate file which will handle file IO and plotting. The following is a list of functions 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 quite a bit 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 19.
With the following flowchart, there are some decisions to be made on what libraries to use and how to go about implementing certain functions.

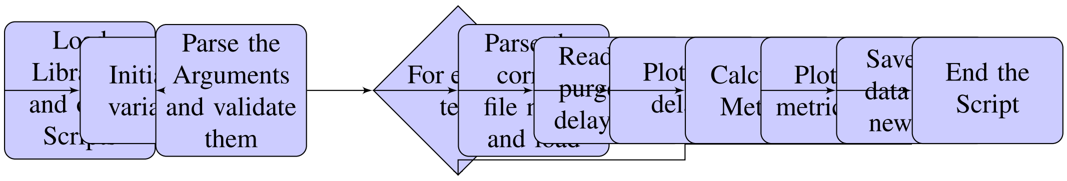


Figure 19: Overall Packet Script Flow Chart

5.5.2 Input Arguments

It was intended for this packet metric script to be fairly comprehensive. Thus it would require 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 would be better to use a more substantial input argument library such that optional flags can also be used.

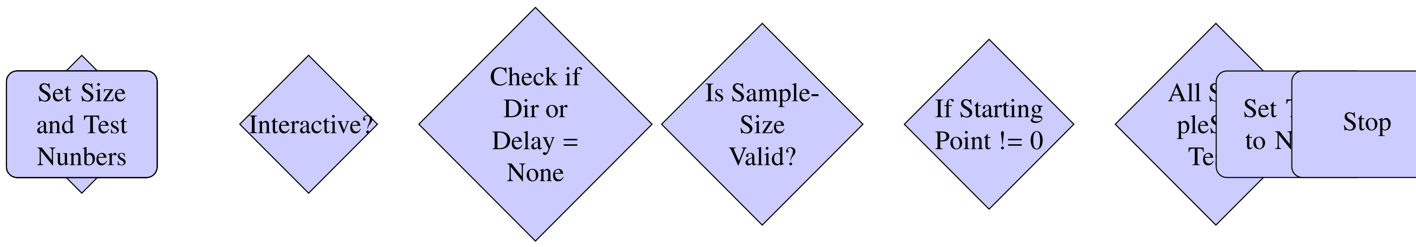
Argparse [19], a library originally written for Python but ported to R, was the most suitable choice. It has numerous useful features including: generates a help command automatically, optional flags, and argument parsing to a data structure.

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. The following flowchart explains the logic that the parsing argument function goes through in order to check if everything is valid.



The flow chart above was implemented in the main packet metric script as well as some of it ported to another function.

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 3rd party libraries available for R, a logging library was used.

Similar with ArgParse, the logging library [20] 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 (Listing 5) details the steps to initialise the logger. Once initialised a logger entry can be written using one of the functions detailed in the documentation [20].

```
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 5: 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

5.5.4 Overview

With all of the above, the script was then created. The final version can be found in Appendix 14

5.6 Function Library

It was a much neater way to port most of the code into functions and save it into a seperate 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, and where applicable some implementation details if they are important to highlight.

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 type, this function will grab 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. This is so the test sheet can be used later if necessary.

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 error message.

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* whicvh consists of all of the results. This i sthen 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 LaTeX if requested using the command line arguments. The CSV file is created using a built in function called write.csv, whereas the LaTeX table 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've been done. The following table (Table **??** 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

- Issues found when testing + very slow when running large data sets + explain that these sets of data need to be large (32 samples per second)

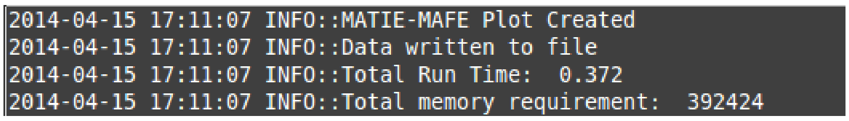
5.7.3 Runtime and Memory Requirements

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 time bound for a set of metric data would be **5 minutes** and the memory bounds work should 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 noted that there wouldn't 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 [21], 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 20.



```
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 20: Example Output for Runtime and Memory Requirements

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

Table 12: Runtime and Memory Requirements for ExampleData

Sample Size	Per Iteration (ms)	Total Time (ms)	Memory (KB)
50			380.064
100			392.424
500			491.176
1000			614.536
5000			1601.240
10000			2834.600

The data has been plotted in Figure **??** below.

(a) Runtime Requirements

Figure 21: Runtime and Memory Requirements for Example Data

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 unknown length, but the length varies on the sample size.

The following were ideas of ways to be able 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.

Some time was spent trying to vectorise the above functions. But due to the use of the built in functions *mean*, *min* and *bandMean*, there doesn't seem like 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 MATIEMAFEAllMethod 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. Therefore the code looks longer than before, but that's because the window needs to be iterated over.

The amended code in C can be found in Appendix 12 for the TDEV method in C and 13 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.

6 Analysis Methods

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 relavent 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. This can still be derived from a plot though.

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.

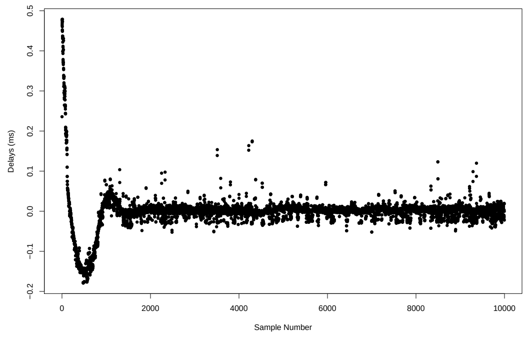
Plotting The final method would be to plot the results. Due to there being a large number of plot types, this would be a suitable method to choose. The data needs to be displayed in such a way that trends can be spotted quite easily. Therefore the following plot types will be used: scatter plots, histograms, 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

The types of plot types that will be used are:

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 **??**) shows two different examples of a scatterplot. Figure 22a is one created using an R script, and the second is from the Syncwatch Lab program, discussed briefly in an earlier section.



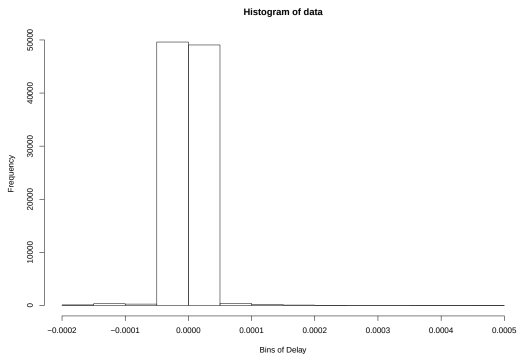
(a) Scatter Plot Example in R

Figure 22: 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 **??** shows the two examples of histogram that will be used.



(a) Histogram Plot Example in R

Figure 23: 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.

6.2.3 Heat Maps

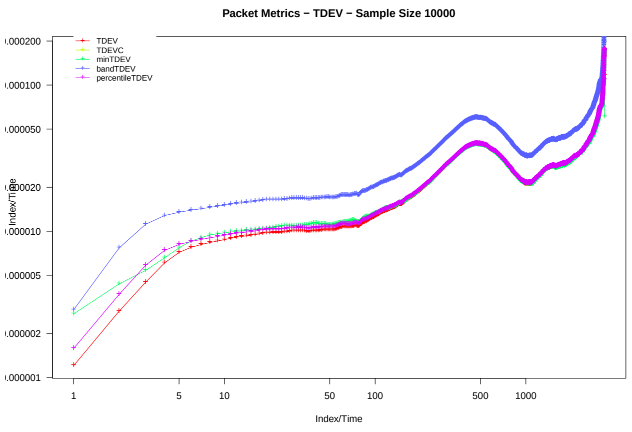
If slices of time are taken, then converted to histograms and flattened to a 1dimensional coloured bar, a heat map of these time slices can be produced. These can be produced in any language, such as Matlab or R. An R implementation has been produced, and an example can be found below in Figure 24.

Figure 24: Heat Map Example in Matlab/R

This is a very pictorial way in showing changes in delay across a set of data.

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 **??**.



(a) Line Graph Example in R

Figure 25: Line PlotExamples

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 **??**.

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 7. 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
- 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 ....

7.2 Delay Distribution

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

of Delays - Test:0 Sample -9996 size.eps of Delays - Test:0 Sample -9996 size.eps

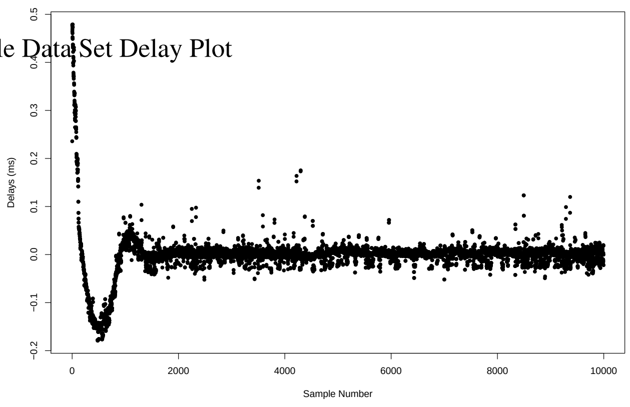


Figure 26: Example Data Set Delay Plot

7.3 Master to Slave vs Slave to Master

7.4 Comparing times of day

7.5 Metric Results

8 Discussion

9 Challenges

10 Milestones

11 Further Work

12 Other Work

12.1 Securing PTP against attacks

PTP Security Tutorial http://www.ispcs.org/security/downloads/PTPSecurityTutorial.pdf

PTP is inherently an unsecure 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 legitamate. Therefore the slave clocks rely on trusting the grandmaster is an accurate time reference and blindly syncronising 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.

12.1.1 Issues with NTP

* find documentation on the NTP DDOS type attack style * * comment whether it's possible with PTP*

12.1.2 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 27) announcing to the network that it has set the highest priority flag to 1.

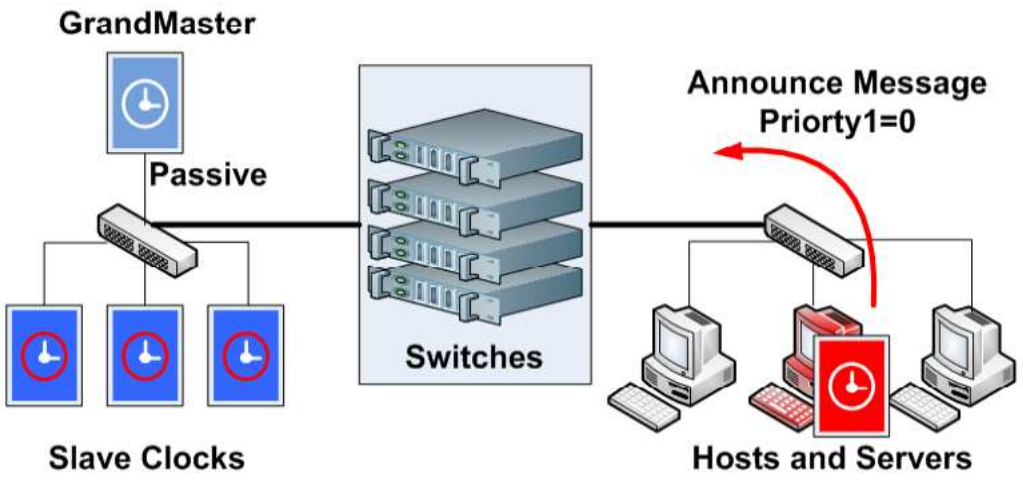


Figure 27: Control Plane Attack Vector (taken from ref)

This would set it to become the Grandmaster clock on the network based on the BMC algorithm. The existing Grandmaster (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.

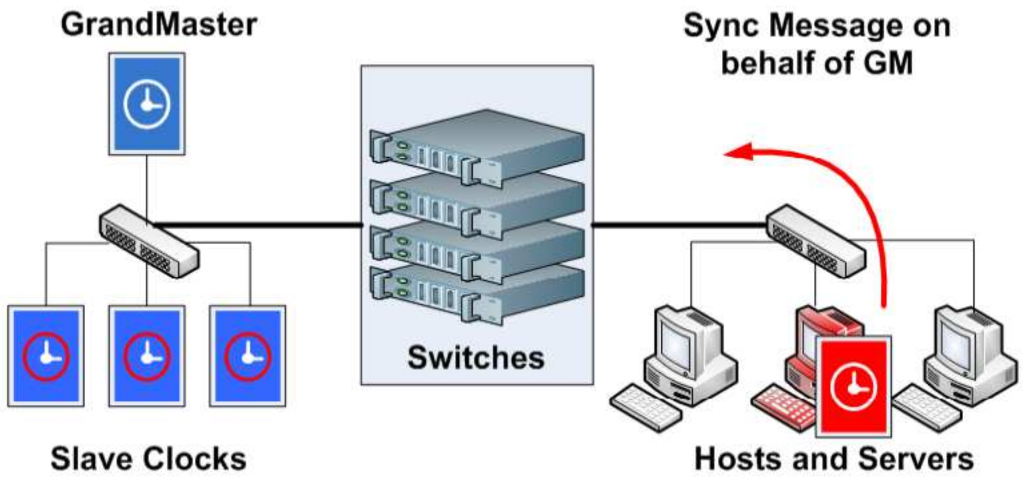


Figure 28: Sync Plane Attack Vector (taken from ref)

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.

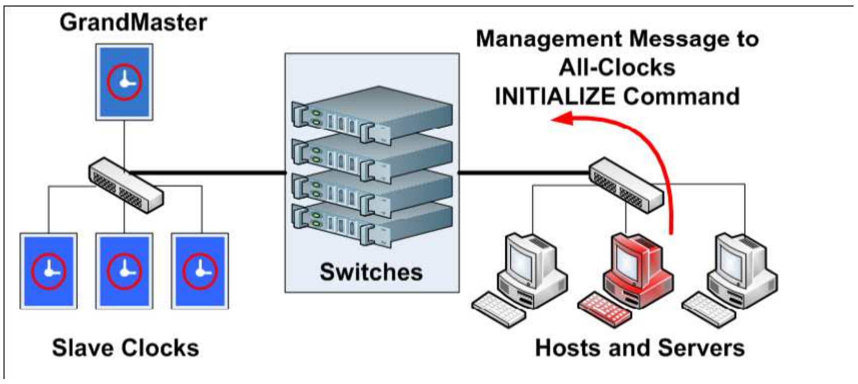


Figure 29: Management Plane Attack Vector (taken from ref)

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.

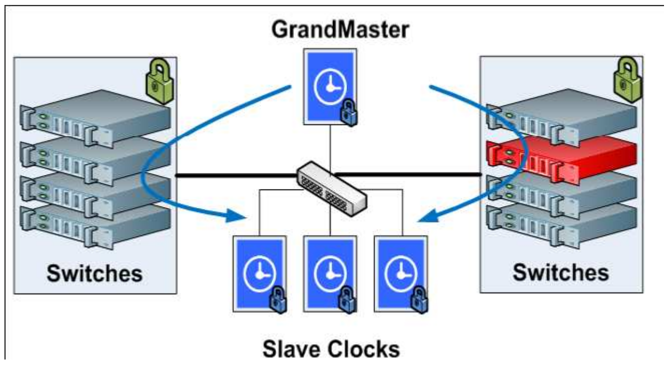


Figure 30: Delay Attack

Confidentiality and Non-Repudiation

Test. add in some info here

Figure 30 demonstrates what would happen if this occured. 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.

12.1.3 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 seperate 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 fool-proof 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.

12.1.4 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 proection, 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 seperate one to each of them.

Requirements

12.2 Sub-microsecond accuracy

13 Conclusion

14 Acknowledgements

References

[1] 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.

[2] Chronos, "Ctl4540 timeport." TimePort Datasheet.

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

[4] B. D. Esham, "File:network time protocol servers and clients.svg." `{http://en.wikipedia.org/wiki/File:Network_Time_Protocol_servers_and_clients.svg},September2007`.newblock Accessed: 2014-02-11.

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

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

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

[14] 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.

[15] C. A, "Preventing the collision of requests from slave clocks in the precision time protocol (ptp)," May 2011.

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

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

[18] Grymoire, "Why learn awk?." `http://www.grymoire.com/Unix/Awk.html#uh-0`. Accessed: 2014-04-08.

[19] "Argparse."

[20] Accessed: 2014-04-12.

[21] Accessed:.

1 Clock Accuracies

	Quartz Oscillators	Atomic Oscillators		
	OCXO	**Rubidium**	**RbXO**	**Ceasium**
Accuracy (per year)	1x10-8	5x10-10	7x10-10	2x10-11
Ageing (per year)	5x10-9	2x10-10	2x10-10	0
Temperature Stability	1x10-9 (-55 to 85)	3x10-10 (-55 to +68)	5x10-10 (-55 + 85)	2x10-11 (-28 to +65)
Stability, Sy t = 1s	1x10-12	3x10-12	5x10-12	5x10-11
Size (cm^2)	20-200	800	1200	6000
Warmup Time (minutes)	4 (to 1x10-8)	3 (to 5x10-10)	3 (to 5x10-10)	20 (to 2x10-11)
Power (W) (at lowest temp.)	0.06	20	0.65	30
Price $	2000	8000	10000	40000

2 Gantt Chart and Table

Figure 31: 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 Chronos CTL4540 TimePort Specification

IPPS	200 nanoseconds over 8 hours (±10 °C temp change)
Holdover	100 nanoseconds ove 4 hours (±10 °C temp change)

Inputs

+5V DC:	MiniB USB
GPS antenna:	SMA
Ethernet (PTP and SNTP/NTP):	RJ45 10/100
Ethernet (management):	RJ45 10/100
1PPS (phase 2):	BNC

Outputs

1PPS:	BNC
Frequency 1: 2.048 MHz, 10 MHz	BNC G.703
Frequency 2: 2.048 MHz, 10 MHz	BNC G.703
IRIG-B:	BNC
RS232:	9 way D-Type 9600 band
RS442:	15 way D-Type 9600 band
Ethernet (PTP and SNTP/NTP) (Max 10 clients):	RJ45 10/100
Ethernet (management):	RJ45 10/100

Environmental

Operating Temperature:	0 °C to +50 °C
Maintain holderover tolderance down to:	−10 °C for 15 minutes
Storage temperature:	−20 °C to +80 °C

Physical

Size:	190 x 57 x 170mm (WxHxL)
Weight:	1150g

4 TimePort Documentation

5 Syncwatch Documentation

6 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			

7 Test Sheet Summary Sheet

Test Number	Directory	Master	Slave	Location Master	Location Slave	Start Time
001	27/02/14	TimePort-To-Software-Test1	TimePort_1	PTPd_Netbook	2E ..	2E 2.13
Finished						
002	28/02/14	TimePort-To-Software-Test2	TimePort_1	PTPd_Netbook	2E ..	2E 2.13
Finished						
003	28/02/14	TimePort-To-Software-Test3	TimePort_1	PTPd_Desktop	2E ..	2E 4.
In Progress						
004	03/03/14	TimePort-To-Software-Test4	TimePort_1	PTPd_Netbook	2E ..	2E 2.13
Finished						
005	03/03/14	TimePort-To-Software-Test5	TimePort_1	PTPd_Netbook	2E ..	Library
Finished						
006	03/03/14	TimePort-To-Beaglebone-Test1	TimePort_1	Beaglebone_1	2E ..	2E 4..
Finished						
007	04/03/14	TimePort-To-Software-Test6	TimePort_1	PTPd_Netbook	2E ..	2E 2.13
Finished						
008	05/03/14	TimePort-To-Beaglebone-Test2	TimePort_1	Beaglebone_1	2E ..	2E 2.13
In Progress						
009	05/03/14	TimePort-To-Software-Test7	TimePort_1	PTPd_Netbook	2E ..	2E 2.13
In Progress						

8 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 6: Awk Script

9 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 7: Band Mean Implementation

10 TDEVAllMethods

```
#!/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 8: TDEV All Methods implementation

11 MATIEAllMethods

```
#!/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               --
#----------------------------------------------------
#----------------------------------------------------
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 9: MATIE All Methods implementation

12 TDEV 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>
```

```c

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 10: TDEV All Methods Implementation in C

13 MATIE MAFE All Methods in C

```c
//------------------------------------------------
//------------------------------------------------
//--                Function Name: TDEV         --
//--                Name: Time Deviation        --
//--           Input: nTo - position in list    --
//--                  N   - number of samples   --
```

```c
//--                    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 11: MATIE/MAFE All Methods Implementation in C

14 Main Packet Metric Script

```r
#!/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

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=""))
  } 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 <- 1/16 #Assume To = 1/16
  dataPacket <- purgeData(Data)
  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)
  }

  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 12: Main Packet Metric Script

15 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 13: Plotting Function for Line Graphs

16 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 = )

}
```

17 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)")
}
```

18 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
    colouredDelayArray[j,1:length(histogram)] <- histogram
    j = j + 1
  }
  png("simpleIm.png")

  image(log(t(colouredDelayArray[,1:30])), col=heat.colors(30,alpha=1))

}
```