Stream Handling in Multimedia Communication Systems

Gavin John Stark

Christ's College

A dissertation submitted for the degree of
Doctor of Philosophy in the University of Cambridge

April 1996

Summary

Recently there has been an explosion of interest in Asynchronous Transfer Mode (ATM) networks as they provide scaleable high bandwidth communication. This has introduced the capability to handle more demanding communications such as real-time video and audio in addition to the more common computer data traffic. One system designed to take advantage of these new capabilities is the Pandora box, which uses the 50Mbps Cambridge Fast Ring network or a 100Mbps fast packet switch network to provide real-time multimedia communications. Systems like this require careful design to achieve seamless integration of different multimedia streams for presentation to users, particularly at the network level.

Other technologies such as the Navstar Global Positioning System have become available to help provide precise, accurate clock synchronisation among distributed systems. The ability to maintain such clocks on multimedia communication systems eases the implementation of the inter-stream and intra-stream synchronisation that is required of such systems.

This thesis provides a thorough examination of the Pandora box, and presents a set of network principles for all multimedia communication systems, from which an implementation of the network subsystem for the Pandora box was built. The network subsystem implementation is described in detail, then further work combines it with the Navstar Global Positioning System to provide very accurate, very precise global clock distribution. This allows a very detailed examination of the operation of the Pandora boxes and how the stream handling inside and between Pandora boxes performs. The results show that the network principles upon which the network subsystem for Pandora was built are fundamental to stream handling in multimedia communication systems.

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

Acknowledgements

I would like to thank my supervisor, Andy Hopper, for all the support he has provided, academically, financially, and in coffee through Olivetti Research Laboratory. All the staff at ORL have helped me in my work in many ways, but particularly David Clarke and Alan Jones have never stopped looking out for me. I must thank Advanced Telecommunication Modules Limited for allowing me time to complete the dissertation, despite the workload requirements of a start-up company. Without my family and friends, whose belief in me has always given me confidence to continue, I would be lost, so most of all I thank them.

This work was supported by a studentship from the Science and Engineering Research Council and also by Olivetti Research Laboratory.

Contents

List of Figures

Stream Handling in Multimedia Communication Systems

Chapter 1

Introduction

There are two main issues examined in the research for this thesis: the handling of real-time multimedia traffic on high bandwidth ATM networks; the accurate distribution of a precise global clock for synchronisation and performance evaluation of these systems. The work covers an examination of real networks carrying real-time multimedia traffic between Pandora boxes. Timing of these experiments is achieved using the global clock, which in turn is derived from global positioning system hardware. The advantages of using global timestamping are looked at, as well as the methods of distributing this global clock around various networks with high accuracy.

1.1 Perspective

Recently there has been an explosion of interest in Asynchronous Transfer Mode (ATM) networks ([68], [49]) as they provide scaleable high bandwidth communication between computers and distributed computer systems. This has introduced the capability to handle more demanding communications such as real-time video and audio in addition to the more common computer data traffic. The traffic can then be divided into two forms: burst data and continuous data. The burst data consists of, for example, file transfer and database accesses, and such traffic needs to be transferred as quickly as possible without interfering with continuous data traffic. Continuous data provides a constant stream of information, for example from a video camera or an analogue-to-digital converter, and so the information has the additional quality of time.

The extra temporal information in continuous data is usually included with timestamps, which are a measure of the time at which the sample of continuous data to which they refer was captured. These timestamps may be implied from a sequence number and sample rate, or times relative to the start of capture or some similar epoch, or perhaps absolute times (i.e. relative to a standard epoch, for example the first of January 1900 at 00:00).

1.2 Multimedia communication systems

A prolific source of continuous data is a multimedia system. Multimedia is a term which describes ‘containing many forms of media’, covering any data such as computer generated graphics, photographic images, video sequences, and samples of audio. The term tends to be abused to the extent that it is used to refer to purely text, still video images, and audio samples, but its wider use encompasses any information media. Multimedia communication systems [50], or distributed multimedia systems [10], use a network to communicate some forms of multimedia data: a video phone system would be an example. [40] provides a very good overview of the requirements of such systems.

A practical multimedia communications system is the Olivetti Pandora Box ([35], [27], [43]), which uses an ATM network as a communication subsystem. The Pandora Box provides multiple streams of video and audio for replay, and can capture continuous data from a single video source and a single audio source. Coupled with a host workstation it can then provide real-time video conferencing and access to and generation of video mail. All the video and audio data is handled by the Pandora Box, including the communications, while the host workstation provides a user interface under UNIX and X windows. Video data is captured by the Pandora box from a standard video camera. The frames of video (individual video samples) are captured at 25 Hz, as defined by the PAL broadcast system, in 256 grey levels. The audio samples are generated by a telephone quality CODEC at 8000 Hz.

A further development by Olivetti Research has been the Medusa system [74], which uses high performance workstations with hardware additions to display high quality video images. Video capture is performed, again at 25 Hz, in full colour and with large images, by special systems with dedicated network connections. Audio is captured and replayed by an additional system, again with its own dedicated network connection, using compact disc quality CODECs, providing two stereo 16 bit samples at up to 48 kHz. On top of these basic systems, hardware for compression as specified by the JPEG standard [69] was developed for the video streams. To provide similar functionality to the Pandora Box, a user is supplied with a camera and video capture system, an audio system with microphone and speakers, and the workstation to display the video images. The communication system used by Medusa is a 100 Mbit ATM switch network.

Much theoretical work has also been done in this area, and other practical systems include the Desk Area Network [23] and the Multimedia Network Interface [9].

1.3 Continuous media and timestamping

Continuous media describes continuous data for human consumption. Examples might be a sport's radio commentary, or the data coming from a video cassette player. Both of these examples contain a quality of time. In the first case the time is absolute - at 2 p.m. England were 64 for 3, in cricket. The second example has at least three different but related times. The first is the time corresponding to the amount of tape since the beginning of the video cassete which is being played. The second is the absolute time at which the data was recorded onto the video casette. A third is the absolute time the data itself was generated, or captured, in a film or television studio. If the video source were in fact a film then the shooting process usually takes place in 'Hollywood' time, where successive scenes may be shot (captured) out of their final order.

A similar example is a compact disc containing two classical symphonies, and its time information. Here there are many absolute times: the time the music was composed, the time it was played and recorded, the time the disc was pressed, the time the data is read off the disc to be converted to sound, and others. None of these times are stored on the disc, although an idea of some of them may be given on the accompanying booklet. The compact disc does store time information, however. It stores the length of each section of music, and the position of each groups of samples within the section. This information may be presented to the user of the compact disc system, and it has many uses.

So before deciding on how to timestamp continuous media, it is important to understand to what uses that information can and will be put. It might be best, when capturing a video stream using a camera, to start with a timestamp of zero in the first frame, and make all timestamps relative to that. A problem occurs if, after some time of capturing, the stream is split to be delivered to two destinations. Does the copy of the stream to the new destination have its timestamps starting from zero, or are they copied straight from the data in its companion stream? Another problem occurs when trying to record continuous data from two seperate sources, for example audio and video from a microphone and camera. When creating the streams for recording, the timestamps can start from zero, but what about the correlation between the audio time and the video time?

The uses of timestamps in continuous media can be split into two areas. The first area is data capture, where all data can be timestamped according to a common clock – global time. The second area is data replay, where the continuous media should be timestamped in a manner suitable for synchronization. If the continuous media is processed between data capture and its replay, perhaps with editting tools, then the timestamping must be adapted appropriately [39].

1.4 Clock distribution

There are many ways of distributing a timing signal for synchronisation. Perhaps the simplest method is to get a suitable number of very accurate clocks, atomic clocks for example. Once they have all been synchronised, they can then be physically distributed, one for each timestamping node. This method is more distributed clocks than distributed timing. Another simple method is to use a broadcast system dedicated to supplying time information and synchronization pulses. This can be done using a piece of wire, with good results, although physical distribution then becomes difficult. Alternatively radio signals can be used, as with the Rugby clock system, which is supported by the National Physics Laboratory. The accuracy achievable using a radio system depends on extra knowledge in addition to the radio receiver itself, as the time information distribution mechanism only works at a finite speed, the speed of light. One hundred miles from the radio source is about half a millisecond of inaccuracy. This error can be partially removed if you know your distance from the radio source, and this is the solution chosen for this thesis. The Navstar Global Positioning System (GPS) uses satellites to transmit coded signals to triangulate the position of a receiver. The satellites use atomic clocks, and include time information in the coded signal they transmit. Most Navstar GPS receivers can produce precise timing information to an accuracy better than one microsecond.

1.5 Objective of work

With the expansion in multimedia communication systems at Olivetti Research, and in particular the increase in availability of high performance ATM networks, it was seen that the handling of continuous media in real-time was important. The Pandora project showed that real-time video and audio streams could be manipulated and communicated with few sophisticated algorithms to cope with network jitter and delay, if the Cambridge Fast Ring (CFR) were used as the physical network. One of the big questions hanging over the Pandora system was why did it work? The Pandora system consists of six Transputers, with internal communication between these processors being handled by the INMOS OS-links and FIFOs, making a complicated multiprocessing system. The five seperate cards which make the Pandora system were designed and programmed by different people, with the interaction between the cards not well understood.

A major aim of the work was to understand what permitted the Pandora system to provide user satisfaction, in terms of synchronisation, latency and corruption of video and audio streams due both to the network communications and internal continuous media management. To effectively evaluate the internal performance of a Pandora system, and to monitor the interactions between two or more seperate Pandora systems, some precise accurate clock distribution software was proposed. The solution decided on for this clock distribution was a simple broadcast system ideally suited to the Cambridge Fast Ring network, which used a single clock source node. With the migration of the Pandora system onto a heterogeneous internetwork it became necessary to synchronise more than one of these nodes, so that internetwork delays could be measured. Thereafter the ATM switch network supplied other challenges.

Synchronising physically distant clock source nodes could be done in many ways, but the solution decided on was to use the GPS system to provide a precise accurate global clock to wherever the clock source node happened to be. This leads to a clock node's accuracy in the order of microseconds, which fortunately fits well with the accuracy of distribution of time around the networks.

1.6 Overview

The remainder of the dissertation is structured as follows:

Chapter 2 This chapter examines the Pandora system in detail, with an aim to understanding the networking requirements of a Pandora box. Each of the hardware and software components of the Pandora system is examined, and from each a set of networking requirements is drawn up. These are combined at the end of the chapter to provide the guiding principles behind any networking software for real-time multimedia networked systems.

Chapter 3 This describes the implementations of networking software for the Pandora box based upon the principles specified in the previous chapter. The software is examined in detail, and where the guiding principles were used in the design of a piece of software this is described.

Chapter 4 This chapter looks at the second aspect of the research for this thesis, the accurate, precise distributed global clock system. In particular this chapter discusses the technology which has enabled these clock systems to be built cheaply, and looks at the way in which the requirements of such a system may be fulfilled in the Pandora system.

Chapter 5 This describes the experiments carried out for evaluating methods of fulfilling the requirements of the global clock system, to provide the most suitable clock for measuring the performance of the Pandora system.

Chapter 6 This describes the implementation of the global clock system, and the way it was added to the Pandora system.

Chapter 7 This chapter describes the experiments performed on the implementation of the Pandora system, the global clock system, and the combined implementation, with a view to evaluating the real performance of the Pandora hardware and software, and determining the usefulness of the global clock for this form of evaluation.

Chapter 8 This chapter draws some conclusions from the chapters, highlighting the requirements for both real-time networked multimedia systems and for the accurate distribution of precise clocks. It also details some conclusions drawn from the experimentation described in chapter 7 with regard to the real performance of a Pandora system. Ideas for further work are then presented.

Chapter 2

Networking for Pandora boxes

The Pandora box is a remarkable piece of equipment. It provides a complete multimedia system which integrates almost seamlessly with the X workstation to which it is attached. The applications to which the system has been put include video phone, video conferencing, display of live television on the workstation, and video mail. The workstation is not burdened with either the video or audio data, neither are the applications themselves. Synchronisation of streams is performed where necessary by the Pandora box and associated systems.

The Pandora box grew out of experimentation with digital video [72] and digital audio [70] over fast networks. When the research for this thesis was started the Pandora box was attached to a Cambridge Fast Ring (CFR) network [28], running Unison Data Link (UDL) ([62], [21]), with the network connection attached to the server card (see below). The initial task of the research was to implement MSNL [44] as the networking layer on a new network-specific card. To perform this implementation required an examination of the data types inside the Pandora system that the networking layer would be required to support. Broadly speaking these data types are real-time audio and video, but more detailed knowledge of the streams was required, so research was carried out into how the Pandora hardware integrated with its software to provide the data streams. This chapter describes that research. It is divided into the following broad sections: an overview of Pandora; a detailed examination of the individual parts, and the implications of these on the network; a summary of the impact on the networking software of the Pandora hardware and software.

- An overview of the Pandora system

  The Pandora system is complex, consisting of a multiprocessing multimedia engine attached to a Unix workstation, and controlling a Pandora system is done using a daemon running on the workstation. This section examines both the features and facilities the networking software must provide, and also the impact those have on the management of a Pandora box at the highest level, interaction with the user. Following on from this it describes an addition to the facilities whereby control of the Pandora box may be managed through the network interface.

- Constituent parts of Pandora

  The Pandora box consists of many interacting subsystems, each having differing demands to place on the network. This section descibes the research into the subsystems of the Pandora box, from the hardware and the software driving that hardware to the implications of these on the facilities the network must provide to fully support the Pandora box capabilities.

- Network principles

  This final section brings together the features required by the Pandora box described in the previous two sections; to summarize the facilities the network must provide in a set of principles for the design of multimedia network software.

This chapter is therefore a summary of the research that was undertaken prior to writing the software for the networking of Pandora using MSNL.

2.1 Overview of the Pandora System

As shown in figure 2.1, The Pandora system consists of an Acorn Unix workstation [1] running X-windows attached to a Pandora box, with communication between the two using an INMOS OS (oversampled) serial link [31] running at 10 Mbps. The Pandora box is also connected to a camera, a microphone, a speaker, a network, the workstation's analogue video output and the workstation monitor's analogue video input. (The video images generated in a Pandora box are mixed with the workstation video signal by the mixer card in a Pandora box.) Currently there are two Pandora system network interfaces, one for the Cambridge Fast Ring (CFR) and one for the Olivetti 100Mbps fast packet switch network, although when this research started only the CFR interface existed. The CFR is an early Asynchronous Transfer Mode (ATM) network, which uses 32 byte packets with a four byte header. The Olivetti fast packet switch network is an ATM-layer ATM Forum standards compliant ATM network using four byte headers on 48 byte cells in a switch fabric.

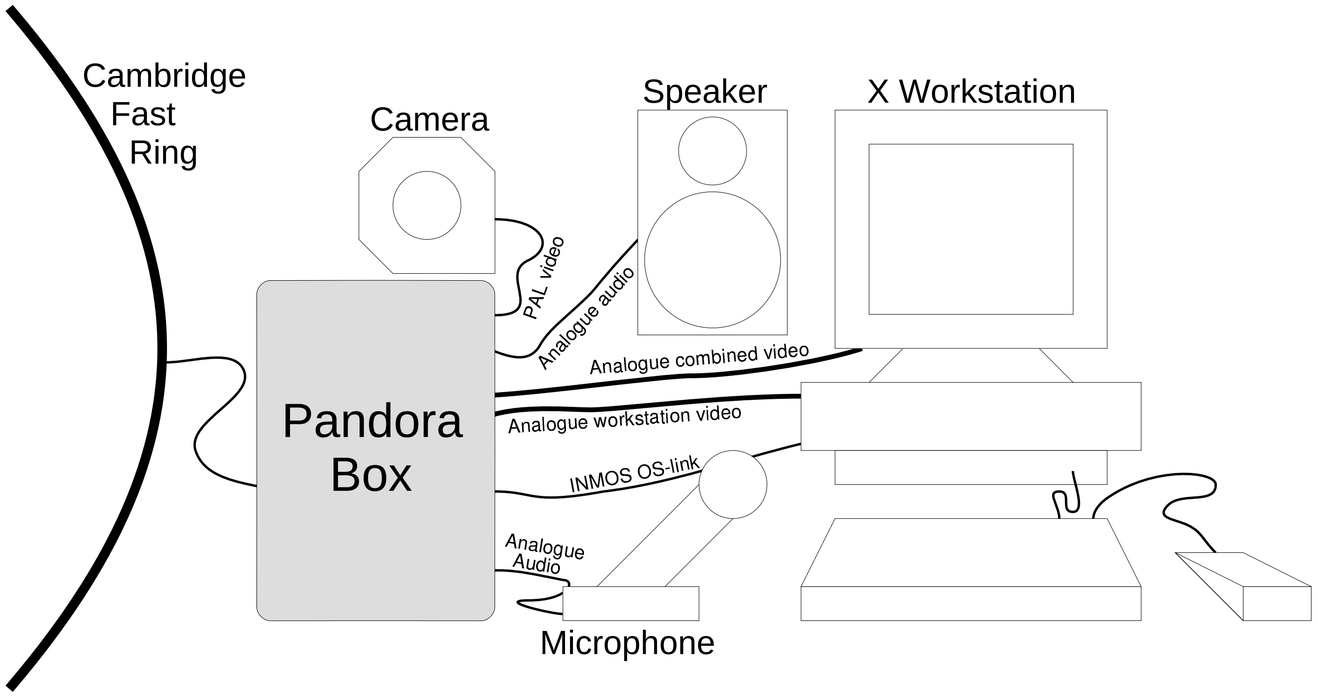


Figure 2.1: A Pandora System and its Connectivity

2.1.1 The Pandora system and its workstation

The workstation for a Pandora system is a standard Unix workstation. The multimedia X applications can run on any Unix system to which the workstation is connected, and they control the Pandora resources they require through the Pandora daemon running on the workstation. The Pandora daemon controls the Pandora box through a device driver which handles the 10Mbps INMOS OS-link; no data passes through the workstation, so operating system support requirements are low [64]. The Pandora daemon is very simple. It supplies multiplexing of the control from client applications onto the Pandora system, and it retains no knowledge of the state of its clients. Communication between these client applications and the daemon is handled using the Unix socket interface, and consists of requests and responses. After every request the application awaits the response. This method provides synchronisation between the Pandora daemon and the X application, and places the burden of contention for access to the Pandora box on the Unix system, but it has drawbacks:

- Pandora box resource management

  The first drawback is that, if the client application crashes or is killed, it may be unable to free any resources it has been allocated by the Pandora box. As the Pandora daemon does not maintain a record of all the resources requested by a client application, it cannot release any of the resources a client may have had allocated to it.

The Pandora box does not have any information about applications. It is not told which application is requesting resources, nor is it told when an application dies, and so there can be no mapping of resources to client application, and no automatic freeing of the resources.

Therefore there can be a gradual building up of wasted resources inside the Pandora box, which can only be released with a system reset. There is folklore amongst Pandora users which states that if anything should not be working on a Pandora box, the first possible solution to try is to reset the box. This is unfortunate in two respects: firstly it is frequently not the solution to the problem (Pandora is a complicated networked system with many parts which can fail, including cameras, microphones, and the network); and secondly it leads to a possibly undeserved reputation of unreliability.

- Speed of control

  The second drawback with the Pandora daemon, X application and Pandora box interactions is speed related. The Pandora box contains six processors, which themselves handle requests and responses very quickly even under quite high loads. The Unix workstations are not so fast (they are relatively cheap, and now old, machines), especially if there is a requirement to switch between processes. For every request from the X application the Unix kernel must switch to the Pandora daemon, use the device driver to send the request to the Pandora box, which must then block until it has read a complete response from the Pandora box, before returning it along the reverse path to the X application. As the software interface to the Pandora box is quite low level, with many requests and responses required for even simple tasks, the setup time for individual applications can be quite long in a user's eyes. An application like '**xvlookall**', which may attempt to open twenty or more small networked video windows, requires such a long time to start up that it becomes almost useless.

2.1.2 Fundamental concepts in Pandora architecture

There are some fundamental concepts behind the Pandora system architecture, in both the software and hardware:

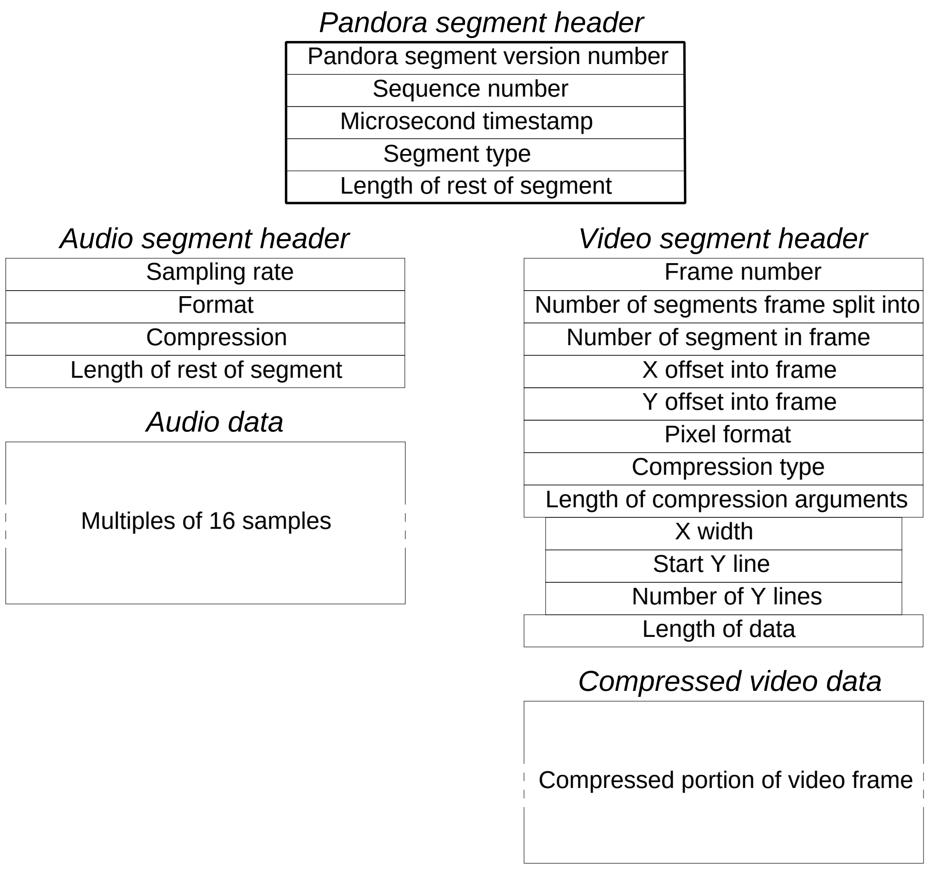


Figure 2.2: Pandora segment format

2.1.2.1 Pandora streams

All continuous data in the Pandora system is handled as streams. These streams of data travel from their source through any intermediate switching elements to their destination. In some cases such as audio streams, the streams may split to go to two or more destinations, so as not to overload the data source. The management of the streams requires control information to handle setting up, routing, and closing them down. Independence of streams is important: the performance of any stream should not affect the performance of any other stream, so that should a particular stream degrade due to overloading of one of the resources it requires, the user only sees degradation on that stream and so may rectify the problem [34].

2.1.2.2 Pandora segments

The continuous streams of data, both audio and video, are divided into Pandora segments for handling. Every Pandora segment contains a Pandora segment version identifier, a sequence number, a microsecond timestamp, a segment type and a length of the segment data in bytes. Following this header is the segment type specific data (see figure 2.2) [33].

For audio segments the segment specific data is a sampling rate in Hz, a format identifier, a compression identifier, and a length field giving the length of the following audio data in bytes. Standard Pandora audio segments use 8-bit μ-law coding (format 0, compression 0) at a sampling rate of 8000Hz, and multiples of 16 samples in a segment. At 8kHz, 1 sample every 125 microseconds, this implies multiples of 2 milliseconds of data. Audio segments transferred between the audio card and the server card use just 4 milliseconds of data, giving a header of 9 32-bit words, and 32 bytes of data, a total of 17 32-bit words at 250 Hz (136 kbps). To improve network performance, or rather not waste performance on sending unnecessary header information, the segment length can be controlled to give more data per segment, although it usually runs with 4 milliseconds of data per segment as above. The aim of small (i.e. temporally short) audio segments is to reduce the time delay for real-time streams of audio between networked Pandora systems. As this is not necessary for non-live streams (recorded and replayed, or those being recorded) larger segments may be used in these cases. In fact the video repository replays audio segments with 40 millisecond long samples (40 32-bit words) with a suitably changed header.

The video data is more complicated, as a frame of video corresponding to an instant of time may be split into many segments. Its segment type specific data contains the frame number of the video data, the number of segments into which the frame has been split, the number of this particular segment, the X and Y offsets into the frame of the segment, pixel format, compression type, the length of the arguments for the compression, compression specific data, the length in bytes of the video data, and finally the compressed video data itself. The compression types supported by Pandora store the X width of the segment data, the Y line number of the segment, the Y height in lines of the segment. So with any of the Pandora compression systems the total length of the Pandora segment header is 17 32-bit words. The compression systems supported are no compression (8 bits per pixel) and DPCM (4 bits per pixel), and frame sizes in normal use vary from 64 pixels by 56 pixels to 128 pixels by 128 pixels. These values give segments sizes from 465 words to 4113 words. As 4113 words is deemed too large to handle the larger frames are split into segments of at most about 4096 bytes, 1024 words long. Also the larger frames are only used compressed as otherwise they use up too much of the Pandora box's resources. So in reality the video segments are between 465 words (one segment per frame, small) and 1024 words long (two segments per frame, large compressed). The frame rate of the video is under control of the application, and is usually either 12.5Hz or 25Hz.

2.1.2.3 Transport of data

The standard Pandora audio segments require continuous data rates of 17 32-bit words at 250 Hz, or 136 kilobits per second (see above), whereas Pandora video segments need an average rate of up to 2048 words at 25 Hz, or 1.64 Mbps. Transporting the audio data inside a Pandora box using 20 Mbps INMOS OS-links is easy without introducing any unnecessary delays. The video data would suffer more, however, and so the architects of the Pandora box hardware put FIFOs on the video capture and video mixer cards for communication with the server card. The INMOS OS-links between the relevant transputers are used to signal data flow through the FIFOs.

As the original Pandora architecture did not allow for a seperate network interface card, using the server card to control a network interface instead, there was no need for FIFOs for data transport to the network. With the change in the Pandora architecture to add a seperate network card it was deemed necessary to use the INMOS OS-link between the server transputer and the network card to carry both audio and video data.

2.1.2.4 Requests, responses and events

The control of the Pandora box by the workstation uses the INMOS OS-link connecting the workstation to the audio card of the Pandora box. The protocol running over this link [73] uses requests acknowledged by responses, with asynchronous events allowed from the Pandora box to the Pandora daemon. (These asynchronous events are generally used for informational purposes during debugging, although they may highlight overload conditions which the Pandora daemon could act upon in the future.)

In normal operation the Pandora daemon places a single request to the Pandora box and waits for a corresponding response. The request from the Pandora daemon arrives at the interface code on the audio and interface transputer. Here it is validated, then decoded into individual requests for processes running on the transputers. Each of these individual requests travels along the request path inside the box until its destination processes are reached, where they are handled. In some circumstances the processes may need to generate responses to these requests, which will be transferred back along the response path (see figure 2.3). If a request to the Pandora interface does not require a response from the lower levels of the Pandora software then it can acknowledge the request immediately with a suitable response. Otherwise it must wait for the response to arrive on the

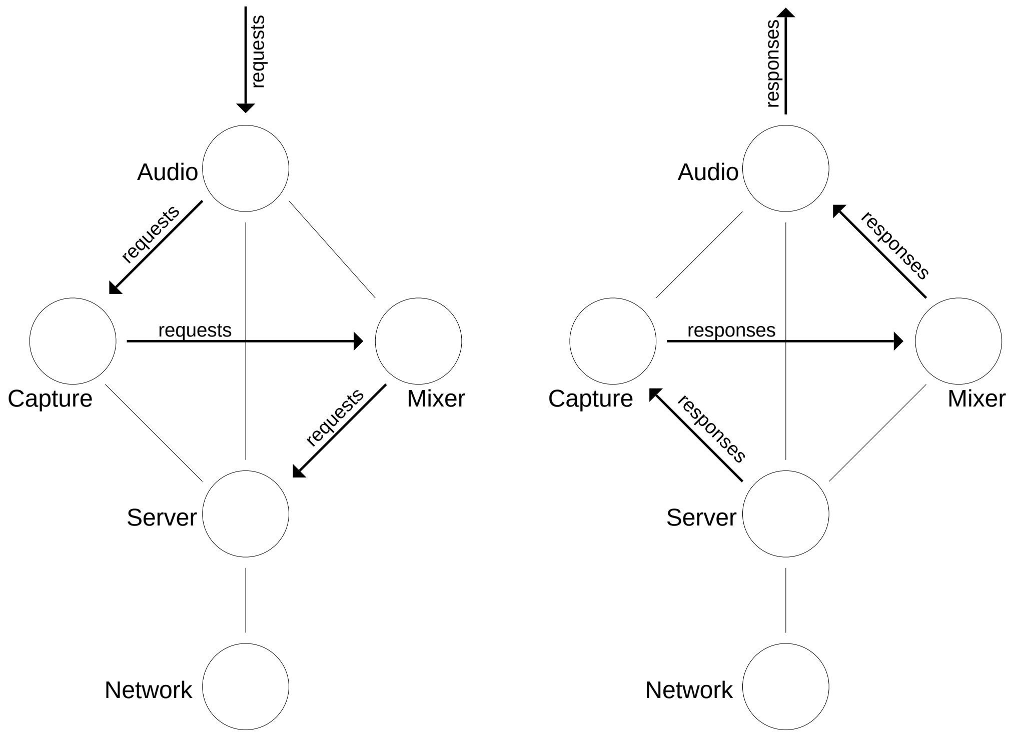


Figure 2.3: The request and response paths between processors

response path before replying to the Pandora daemon. In addition, asynchronous messages may be generated by any of the processes on this response path, and these are passed all the way back up through the interface code to the Pandora daemon.

A request on the request path consists of a header of two 32-bit words (a processor address mask and a message length) followed by the message body. A message body consists of an operation identifier, a process address mask, and a second message length giving the length of the remaining message body. On each processor on the request path an incoming message is checked to see if the request is destined for one of its processes, by checking one bit in the processor address mask. If so the request is passed on in its entirety onto the internal request path for that processor. A request which may also be destined for another processor is passed on to the next processor on the request path. The internal request path for a processor is connected to a *fanout* process, which uses one bit of the process address mask for each of its output channels to work out whether the request (operation, new length and message body) should be routed to a process through its channel. In this way requests may either be sent to a particular process on a particular processor, or they may be multicast to many processes on one or more processors.

A response on the response path consists of a sequence of individual reports. Each report consists of a report type, a length field, and a report body. Report types include string, integer, byte and integer array. As responses are a sequence of reports, and multiplexing the responses from many sources is necessary without corrupting individual responses, a response is defined to be a sequence of reports ending with a report of type *release*. On every processor there is a multiplexer process which understands this response format, and which can multiplex the responses generated by processes on that processor with those it receives on the incoming response path onto the outward response path. A hierarchy of response channels inside a processor is formed with multiplexers used to connect a number of incoming channels to the outgoing channel. The response structure allows addition to a response at any stage by inserting extra reports. One of the uses of this feature is the response *prefixer*, which sits on a response channel and adds a fixed report to the start of every response that passes through it, as a way of describing where the response has been. This helps to identify the origin of a particular response. Another use of the response structure is the response *filter*, which removes low priority responses from the response channel while passing high priority responses on. The priority of a response is set using a special *priority* report type. A response to a request from the interface code, as opposed to an asynchronous events, use the special report type *reply* for one of their component reports.

2.1.3 Network implications

The network card must receive and transmit Pandora segments over a single 20Mbps OS-link, and it may need to generate messages regarding resource overflow or network integrity. In addition the concept of the Pandora stream must be continued into management of network connections, which must be performed under control of requests from the Pandora daemon. The following sections therefore describe in detail the implications that the fundamental concepts described above have on the network software.

2.1.3.1 Pandora streams

To continue the Pandora stream system into the network requires a mapping from network connections to Pandora streams. The connections have to have properties similar to those of the streams internal to a Pandora box, and so they should not interfere with each other unless the network card is overloaded. Individual stream overload must be handled by the network software, with messages being generated to indicate this, and should not degrade other stream performance. Ideally a connection should be under full control of the Pandora box, including full control over creation and destruction of streams.

2.1.3.2 Pandora segments

The size of the Pandora segments and the frequency of them inside a stream must be the driving factors behind the design of the network software. From section 2.1.2.2, the two types of segment are quite different: video segments are larger (1860 to 4096 bytes) and less frequent (2 segments at 25Hz or below) than audio segments (68 bytes at 250Hz). Therefore the network software must be designed to cope with both sorts of traffic and must maintain a quality of service on a per stream basis. In addition the Pandora segments contain no checksum or similar validity check, and so, as in the internals of the Pandora box, corruption of the Pandora segments is not permitted. It has been suggested that rate-based data transport protocols like VMTP [11] or NETBLT [12] are promising candidates for multimedia data transport, providing steady reliable transport. However, each Pandora segment is entirely self-sufficient, and needs no other Pandora segment to help describe the data it carries. So reliability of the transport of segments is not a requirement, although it is desirable. Then, the underlying network need only meet the requirement for smooth transport of data.

2.1.3.3 Data transport

Unlike the other high bandwidth subsystems of the Pandora box the network is only connected to the server with a single INMOS OS-link, which affects the performance available in terms of both bandwidth and stream multiplexing. Data for many streams is multiplexed onto the INMOS OS-link for both reception and transmission. Large video segments may take a long time to transmit over this link (up to 4.5 milliseconds, or even 5.3 milliseconds if data is travelling over the link in both directions). This could have a major impact on the audio performance of two networked Pandora boxes, especially if audio is kept waiting for transit down the link by more than one large video segment.

2.1.3.4 Request, responses and events

The most obvious choice of control mechanism for the network card is the request and response system used throughout the rest of the Pandora box. This requires adapting the Pandora libraries to add new processor and process addresses for the new processor and its software, and increasing the requests to allow full control of the network. However, the request and response system places control entirely in the hands of the Pandora box software, which is designed for the reliability of internal Pandora box connections and not the world of lightweight network connections. The latter may be broken due to forces external to the Pandora box, for example catastrophic network failure (i.e. someone unplugging the network connection), whereas the former will run perfectly provided the Pandora box itself is not vandalised. Also in a networking situation a connection may take a long time to set up, especially if compared to an internal connection. If the request/response system is used, a choice would have to be made as to when to send the response indicating a successful connection. If a response indicated a connection would be attempted, then at the failure or success of that connection an asynchronous message would need to be generated by the network which the control software would have to handle. On the other hand, if the network waited until a connection had been established over the network before responding to the request then, as the Pandora system waits for the response after a request, the whole control of the Pandora system would have had to wait for the network to make that connection.

Another problem with the Pandora implementation of the event system using the response channel is that none of the event messages is acted upon. Each subsystem is responsible for managing its own overload conditions, which is not unreasonable for the video and audio subsystems as they have full control over their resources, and any asynchronous messages are just logged by the Pandora daemon on the Unix workstation. However, the network subsystem does not have full control over its resources, as network performance depends on other network users. Overload can occur due to entirely external reasons, to which the only solution is higher level control.

There are three options for coping with these problems: add the handling of unreliable connections and asynchronous responses to the Pandora box control software; force the network to provide unbreakable instantaneous connections; take the network out of the request and response system and so force the Pandora control system to be enhanced or pass responsibility for handling a misbehaving network onto the user. As the CFR is quite unreliable and has many sources of failure it is unreasonable to choose the second option. The first option would require major changes to the control software, which is outside the scope of the chosen research. Choosing the third option turns the networking card into a seperate device with integral device driver, to which simple commands are sent, from which asynchronous events are received, and data transmission and reception are handled without any overheads for the rest of the system. It logically removes the network card from the Pandora box in a more modular approach than the integration of the other parts of the Pandora box. This frees the Pandora control software from any dependence on the particular network to which it is attached, and it makes the network interface usable in applications other than a Pandora box. It also makes the network code more reliable as it is testable on its own without any other Pandora code.

2.1.3.5 Implications of the system

The Pandora box is a large stand-alone system with complex interacting subsystems, driven by many microprocessors. Debugging such a system can be difficult, especially the network subsystem, as it is the furthest subsystem from the host. It does have a spare INMOS OS-link accessible when prototyping, and this can be used for debugging.

Once the Pandora system with seperate network cards were in general use, access to the debugging link was difficult due to the physical casing of the Pandora box. In addition, debugging a collection of physically distributed Pandora boxes, necessary because of their interactions, makes it necessary to have the better alternative of supplying the network software with a network-based debugging interface.

2.2 Constituent parts of Pandora

The subsystems of the Pandora box have different networking requirements, in terms of latency, jitter, reliability, and more subtle features. Each subsection below starts with a description of the hardware of a subsystem, followed by an examination of the implementation of the software driving that hardware. The subsections end with a short discussion of the networking implications of the subsystem.

Note that there have been two major versions of the hardware for the Pandora box (in terms of the subsystems), and these are described below as Pandora 1 and Pandora 1a.

2.2.1 The Pandora interface

A Pandora stream starts at a data source, which produces Pandora segments depending on some source-specific parameters. It then travels through the server card, which routes it to a sink, whose behaviour may depend upon sink-specific parameters. (The source or sink may be the network, which a single Pandora box regards as a start or end point for data.)

The management of streams, creation and destruction of streams, the control of the stream flow and the control of the source- and sink-specific parameters, is performed by the interface subsystem.

2.2.1.1 Hardware

The original plans for communication between the Pandora box and the workstation was by means of a SCSI interface. The Pandora box would provide the multimedia facilities with the control software (the *interface* [73]) running on the workstation. This required one of the subsystems to have a SCSI interface. The data rates of Pandora audio streams are very low compared to video streams, especially considering it is only telephone quality. Also it is very hard to listen to more than one audio source at once, whereas a workstation may be required to display many video images. These thoughts led to the idea that the audio processing card would be the least heavily loaded of the Pandora box, and so would be the best option for the placing of the SCSI interface attached to the workstation.

To make the initial implementation of the Pandora system easier, it was then decided to place the interface software inside the Pandora box, on its end of the SCSI connection. In the end, the SCSI interface was not fully supported; instead an INMOS OS-link connecting the audio processing card to the workstation was used for the control. Also, despite the plans, the Pandora interface software was never moved into the workstation, and remains on the audio card.

So the Pandora interface has no dedicated hardware, either to control or maintain.

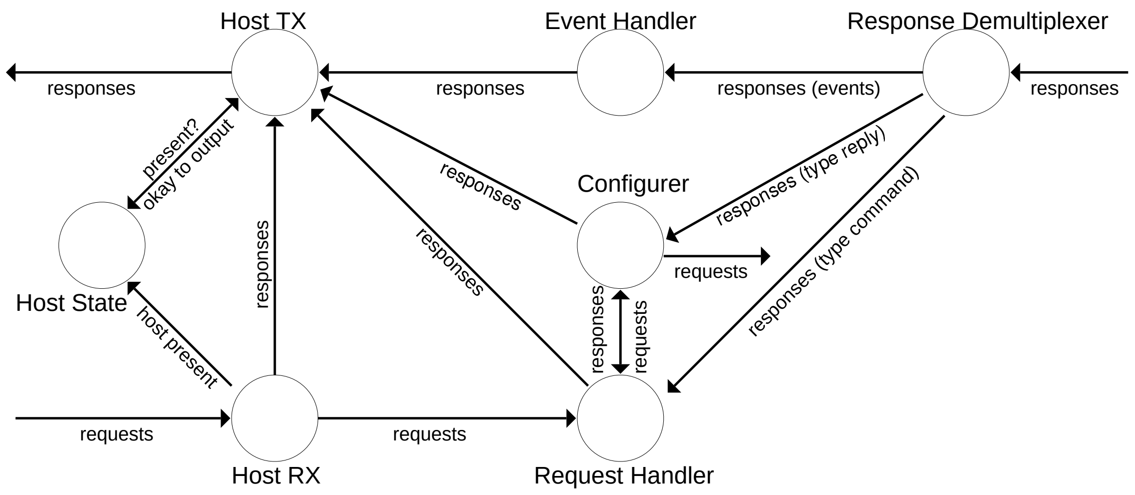


Figure 2.4: Process and Channel Structure of the Interface Software

2.2.1.2 Software

The interface layer in the Pandora box provides the control mechanisms for handling video and audio streams, from sources, through the server and network layers, to sinks. It is connected to the workstation by one INMOS OS-link, over which it multiplexes asynchronous events (such as those generated by overflowing buffers in the audio processing code) and responses to requests from the workstation. As described above in the request-response section the interface code was designed to handle only one request from the workstation at any one time.

The process structure, shown in figure 2.4 divides into three areas:

1. Host Management

   The *host state* process attempts to keep track of the state of the attached workstation. This is so that the *host TX* process, which sends responses to the host over the INMOS OS-link, only sends messages if the host is working. The host is deemed working if the *host RX* process receives a request, and is deemed to be not working if the *host TX* process fails in an attempt to send a response to the host. So the *host RX process* informs the *host state* process whenever it receives a request, then it forwards the request to the *request handler*.

2. Request Management

   In the original design of Pandora, where there was no control path from the network, a request arrived at the *configurer* from the host management processes. This process converts external requests into one or more internal Pandora requests. If the request required replies from the internal Pandora requests then the *configurer* process would block until it had received the replies. The response to the external request can then be sent to the *host TX* process, and the *configurer* can then await the next request.

As described above, the single request/response access through the Pandora daemon to all of the Pandora features was a performance bottleneck for complicated applications. Also, the Pandora box was tied to the Unix workstation as it had to be controlled by the Pandora daemon. With the advent of new networking software there was a chance to improve the request management system to remove these problems by adding network access to the request/response system. To do this the *request handler* was added as a layer on top of the *configurer*. This process performs two tasks. First, it buffers requests for the *configurer*, to allow concurrent requests from applications. Secondly it generates external requests for the *configurer* from responses it receives from the *response demultiplexer*. These responses must have the new, special report type *command*, and may be generated by any process on the response path. The *request handler* intercepts the replies generated by the *configurer* process so that it knows when it can send another request to the *configurer* process. Also, these replies must be sent to the host if they are in reponse to a request from the host, or, using a request to the *configurer* process, to the network if they are in response to a network-sourced request.

3. Response management

   The responses from the lower levels of the Pandora software arrive at the *response demultiplexer* process. This looks at the individual reports which make a response, to find the type of the response. A response with a report type *reply* is sent to the *configurer* process, as it is the reply to a request the *configurer* is waiting for. A response with a report type *command* is sent to the *request handler* to be decoded and inserted into the buffer for the *configurer*. All other responses are asynchronous events, which are buffered in the *event handler*, before being forwarded to the host.

2.2.1.3 Network implications

In the original Pandora implementation, where requests and responses always originated from the host OS-link, the interface software had no bearing on the network. The addition of control access from the network to the interface request/response system does place extra requirements on the network software.

Firstly, the requests and their responses in the Pandora system require a completely reliable networking subsystem. Each request must be checked for corruption, then executed exactly once. The response to the request must then reach the destination uncorrupted.

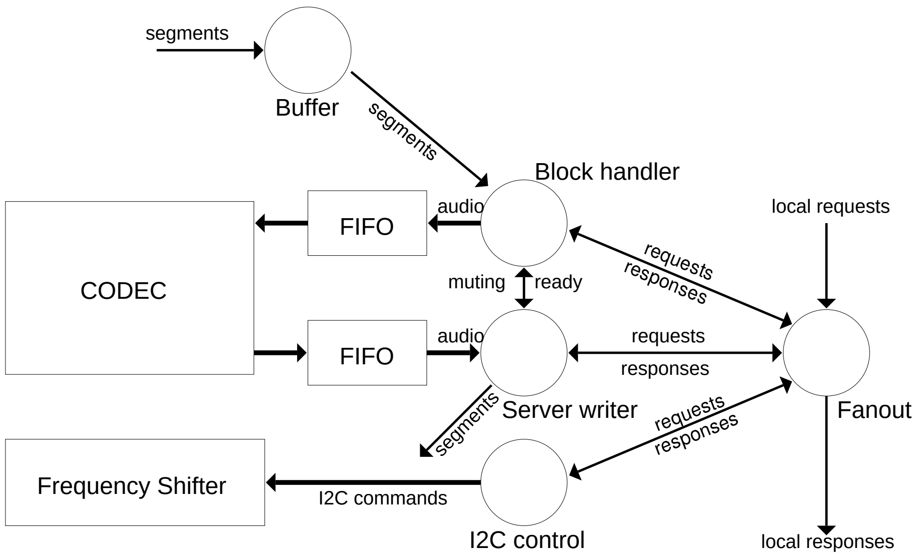


Figure 2.5: Hardware, Process and Channel Structure of the Audio Card

Secondly the network layer must allow connections on a public Sevice Access Port (SAP) for any other network entity which may wish to control the Pandora interface code. There may be many such clients at one time, and so provision for multiple connections per MSNL SAP is necessary.

Thirdly, with many clients sending requests at one time there must be a suitable amount of buffering of requests, with appropriate responses to the clients should the buffering be exceeded.

Fourthly, for the extra network access to be useful, the latency of the transport of the requests and responses must be much lower than that experienced in the Pandora daemon, less than one millisecond.

2.2.2 The Audio card

The audio processing card on a Pandora box provides both audio input and output facilities, in addition to other more esoteric features (depending on the hardware version, see below). It supplies a single audio data stream to the server and mixes many audio data streams for connection to a speaker. (See figure 2.5.)

2.2.2.1 Hardware

The audio processing board contains a T425 transputer [31], DRAM (256 kilobytes on Pandora 1, 1 megabyte on Pandora 1a). Digital to analogue conversion and analogue to digital conversion is handled by a telephone quality (8bit μ-law, 8kHz) CODEC, fed by and feeding 512 deep, 8 bit wide FIFOs.

Pandora 1 audio processing cards contain a SCSI interface (originally planned for communication with the workstation), a full telephone handset handling circuit (handset up and down detection, tone dialling decode), and an Active Badge transceiver [71].

Pandora 1a contains an I^2C interface [52] to manage analogue-to-digital converters, a small EEPROM, and eight digital-to-analogue converters. The analogue outputs from these control a pitch shifter, as well as contrast and black level settings for the video capture card.

The audio processing transputer is connected with its four INMOS OS-links to the host, the server transputer (audio data in both directions), the video capture transputer (requests out) and the video mixer transputer (responses in).

2.2.2.2 Software

The main task of the software on the audio processing card is to supply the server transputer with a continuous stream of audio data from the CODEC (the *server writer* process), and to take a variable number of streams from the server transputer to be mixed together prior to playing out through the CODEC (the *block handler* process). These audio streams travel along the OS-link between the two processors.

There may be jitter in the output streams because of the paths taken from their sources, and the clock rates used by the sources and sink will almost certainly differ slightly. To remove the effects of these infelicities, clawback buffers are implemented in the *block handler* process just before the output streams are mixed, as described in [34]. The audio software also tries to prevent echoes due to sound from the speaker being reflected by the room back to microphone, by reducing the input level (from the microphone) when the output level (to the speaker) is above a certain value, in a three-state process: the first state applies no input level reduction; the second state reduces the input level by half; the third state reduces it to 20%. Moving from the first state to the second, or the second to the third, occurs when the output level exceeds the threshold in a 2 millisecond period, whereas the other direction requires the output level to be below the threshold for 20 milliseconds.

The other extra facilities provided by the audio processing cards are not supported by the Pandora stream paradigm — instead, it is all under the control of the interface using the request-response mechanism. So extraction of the data from the analogue-to-digital converters, or control of the pitch shifter, must be performed with requests to the interface, and no data streams are set up.

2.2.2.3 Network implications

The audio software is, within some limits, easy to satisfy in terms of network performance. The data streams are low bandwidth, and any network jitter can be recovered from by the buffering. However, a slapdash approach is inappropriate as good performance is required. Here is a closer look:

1. Audio bandwidth

   A single Pandora audio stream, 136 kbps, is considered to be a low bandwidth stream, especially when compared with the data rates supported by the ATM networks Pandora was designed for (50 or 100 Mbps). It is also low bandwidth when compared to the 20Mbps INMOS OS-links along which it must travel to the network interface, so this should not be a problem.

2. Network jitter and latency

   Even though the audio bandwidth does not place a heavy load on the network interface, there are other considerations which do. It is very important that the latency introduced by the network and network card is kept to an absolute minimum, and similarly the jitter must be kept as small as the network itself will allow. These are important as humans understand conversation, and general interaction with the world, using their auditory sense more acutely than their visual sense. As an example, with a videophone conversation it is far harder to transfer information between users if the audio connections disappear, than if the video connections disappear. If the streams are only degrading, then if audio segments are lost in the network layer this introduces clicks and gaps in the audio heard by users, whereas lost video segments frequently go unnoticed, and users perceive little degradation.

   To quantify this, the perceived audio performance is affected by the end-to-end delay and any data lost or dropped by the stream. Large jitter (i.e. greater than the downstream buffer size) causes data to be dropped. Total end-to-end delay is the individual transport stage times of the data plus the buffering at each stage. The *server writer* process introduces 4 milliseconds of buffering in assembling a segment for the server transputer. There is extra buffering in the server and network cards (at least one segment per stage). On a standard stream this will account for at least 4 more buffering points, two servers and two networks. Finally, the *block handler* process attempts to remove all upstream jitter with its clawback buffering. The size of this buffer equals the maximum jitter in its stream (with a target minimum of 8 milliseconds). Now, the end-to-end delay of an audio stream should be bounded by two values: a maximum end-to-end delay of 50 milliseconds (100 milliseconds round-trip) for echo to not be a problem, less than 10 milliseconds round-trip for it to be imperceptible; a minimum end-to-end delay of the video end-to-end delay, to maintain lip synchronisation on video conferencing. These two constraints cannot both be met without streaming video directly from camera to display, not buffering frames at any point in its path. As the video frames are buffered in Pandora, one of the constraints will be broken, and as audio is more important than video, the maximum constraints should be met. To keep to this maximum end-to-end delay the latency and the jitter in transporting streams must be kept low (jitter 5 milliseconds and latency 5 milliseconds).

3. Reliability

   The low bandwidth of audio streams implies that providing fairly reliable network transport for them should not be difficult. However, any lost segments will not be recoverable in any form by an audio sink (unlike with video where successive frames may be very similar). So audio streams require a higher degree of reliability than, for example, video streams. But there is no need to implement fully reliable communication, provided the segment loss is kept low. Each loss will be audible, but a single click at perhaps 30 second intervals is not going to worry the user too much. This would be a lost segment rate (with segments of 4 milliseconds duration) of at most 1 in 7500.

4. SCSI interface and extra facilities

   As the SCSI interface is not fully implemented on Pandora 1 and it was not included in Pandora 1a, there are no special requirements placed on the network card to support it.

   The extra facilities provided by the audio processing cards such as I^2C connections are controlled and read by the request-response mechanism, so again the network interface need not supply any feaures to support them.

2.2.3 The Video Capture card

The video capture card digitises and compresses the output from a single camera, providing the server card with multiple Pandora video streams. (See figure 2.6.)

2.2.3.1 Hardware

The video capture card contains a T425 transputer with 1 megabyte of memory on Pandora 1a, and 256 kilobytes of DRAM on Pandora 1. In addition there is 1 megabyte of VRAM used for the capture of compressed images.

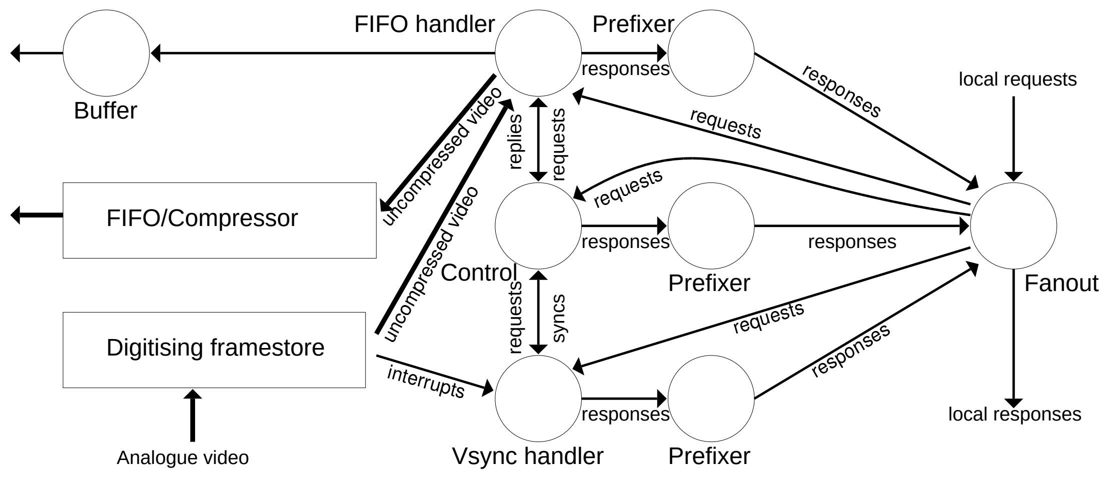


Figure 2.6: Hardware, Process and Channel Structure of the Capture Card

The video signal from the capture is digitised to 256 grey levels (using contrast and black level set by the digital-to-analogue converters in the audio processing card in Pandora 1a), and put into the VRAM accessible from the transputer. The software has to transfer the data from the VRAM to FIFOs for the compression card. The compression card can be switched into a null compression mode, where it just passes the data through, or it performs DPCM compression, line and pixel discarding. The compression card extracts the video information from the FIFOs, and inserts its results into another set of FIFOs, which are readable by the server transputer. The horizontal and vertical synchronisation pulses from the video source are available to interrupt the processor.

The video capture processor is connected with its INMOS OS-links to the audio card (requests in), the mixer card (requests out and responses out) and the server card (control of the FIFO data and responses in).

2.2.3.2 Software

The video capture software is split into three parts: a video synchronisation signal handling process; a FIFO handling process; a control process to glue everything together.

The video synchronisation process maintains a knowledge of the current stage of the video camera's image, in terms of lines and interlace field, by counting horizontal synchronisation pulses and resetting the count on vertical synchronisation pulses. While capturing data for streams the synchronisation process informs the control process at successive 38 horizontal lines (one eighth of a field). It is worth noting that this process is scheduled and run at every horizontal line synchronisation pulse, i.e. approximately every 65 microseconds.

The control process maintains the following information pertaining to streams: capture rate; capture size; scale; area of the camera's view to be captured. It uses this in conjunction with the information from the video synchronisation process to go round the streams one by one, informing the FIFO handling process when a segment of data has been captured into the VRAM. By doing this it maintains a fair distribution of capturing to the different streams even under overload conditions.

The FIFO handling process accepts the prods from the control process on a per segment per stream basis. Using this information it sets the compression card parameters then transfers data from the VRAM into the compression input FIFOs in *slices* (a subdivision of a video segement's data, but still a whole number of video lines). As each slice is inserted into the compression FIFOs the server is informed via the *slice buffer* that the slice has been inserted, so that the server processor can then extract the compressed slice from the output of the compression system.

The capturable sizes are 64 by 56 and 128 by 112 pixels. 256 by 224 pixels is possible, but it produces too much data for the rest of the Pandora system to cope with at video frame rates, so it is not used. With 256 grey levels and 25 frames per second this yields data rates of 0.7 megabits per second and 2.8 megabits per second respsectively. With DPCM compression these figure are reduced by a factor of 2, to 0.35 and 1.4 megabits per second. It should also be noted that half of this frame rate, 12.5 frames per second, is often almost indistinguishable from the full frame rate, especially at small sizes. These data rates may not be considered high in the future, but Pandora was originally a demonstration of, and experiment into, what is achievable on networks of accessible bandwidth of 10 megabits per second.

2.2.3.3 Network implications

Unlike the audio streams, the high bandwidths required by some instances of captured video streams need careful handling in the network system:

1. Data copying

   Copying an audio segment from one memory area to another may be done very quickly because of its size, and so may be performed if necessary for a fast networking algorithm or efficient memory use. However, a segment of video data is up to 8 kilobytes long, so as little copying of the data as possible should be done. For the same reason the networking software must manage the data reception over its OS-link with the server in a memory-access efficient manner.

2. Overload of transmission of video data

The network interface must be able to cope gracefully with overload on transmission. Initially this means maintaining as high a bandwidth as possible without wasting resources on data which is not going to be successfully transmitted. As soon as the overload situation is detected the network software could originate some commands to reduce the bandwidth demand of the stream, by asking the capture system to increase the compression or reduce the picture size. There are two problems involved in following this course: the network software immediately becomes specialised to the capture software; overload in the network may manifest itself in many ways, which would lead to similar network overload conditions producing different effects.

Inside Pandora a principle of local adaptation to exceptional stream conditions applies, so the best solution is to adapt as best as possible to the overload by discarding unmanageable data at the earliest opporunity, and generate overload messages.

3. Network transmission buffering of video data

   Buffering of more than one Pandora segment per stream is necessary for audio data to accomodate network jitter. More than one video segment per stream needs to be buffered as the peak data rate of arrival of video may be higher than the network transmission rate for a single stream. So in the case of video transmission, a decision has to be taken on how much buffering to provide.

   Initial implementations of the network software were written for a network card containing 64 kilobytes of memory for code and data. With the video segments taking up to 4 kilobytes each and 2 segments making up a picture the absolute maximum amount of buffering available for a video stream would have been 8 frames. At a full frame rate of 25 frames per second, this is one third of a second of video buffered at one time. Accounting for code size and multiple streams meant that the real limit was much lower. Unlimited buffering up to memory size was allowed, and no ill effects were seen.

   Further implementations of the network card had 2 megabytes of memory. Initially again the only limit on buffering of data was the amount of memory available. Now when the source rate was higher than the transmit rate (i.e. the network was a bottleneck) the segments backed up in memory, leading at some stages of testing to a five second delay between source and sink Pandora boxes.

   Choosing the amount of buffering can be done either in terms of buffer memory size, number of segments, or temporally. Limiting by memory size affects different types of video in different ways, and does not generalise

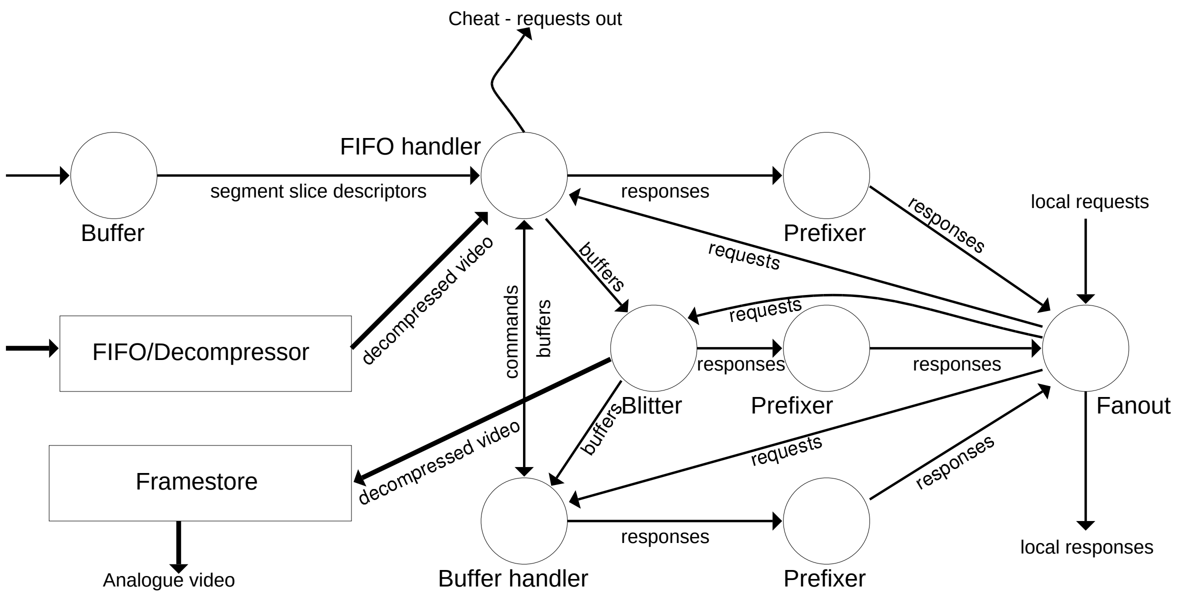


Figure 2.7: Hardware, Process and Channel Structure of the Mixer Card

to other forms of data. Similarly the number of segments which make up a video frame differs according to compression rate. So the third option was chosen, to buffer up to 150 milliseconds of data. This is a suitable length of time as it does not affect inter-stream synchronisation too much, yet it allows for more than the expected amount of jitter in the network. As a principle, buffering only needs to be provided to allow full bandwidth concurrent operation of the hardware. Providing enough buffering to cope with the jitter in the network is therefore sufficient.

2.2.4 The Video Mixer card

The video mixer card in the Pandora system takes the analogue video output from the workstation, and mixes it with the output from a framestore on the video mixer card itself, using a digital mask to determine which video output is visible at each point on the dislay. The processor fills the framestore with video data taken from the output FIFOs of the decompression unit. (See figure 2.7.)

2.2.4.1 Hardware

The video mixer card hardware is very similar to the video capture hardware, with the capture/compression hardware replaced by decompression/display generation hardware. It contains a T425 transputer with 1 megabyte of memory on Pandora 1a, and 256 kilobytes of memory on Pandora 1. In addition there is 1 megabyte of VRAM used for the framestore. The analogue output from the framestore is generated using an 8 bit colour lookup table. The workstation analogue video output is mixed with that of the Pandora framestore using the values in a 1 bit deep pixel mask. A '0' bit in the pixel mask lets the Pandora video show, a '1' bit in the pixel mask shows the workstation video. The vertical synchronisation signal from the workstation analogue video signal is available to interrupt the processor.

The server processor writes compressed video data into the input FIFOs of the decompression card, which decompresses into output FIFOs from where it must be read by the processor, which can then place it into the framestore VRAM.

The OS-links of the video mixer transputer are connected to the video capture processor (requests and responses in), the server processor (requests out, segments in), and the audio processor (responses out).

2.2.4.2 Software

The video mixer is fed *slices* (see section 2.2.3.2 above) of compressed video data by the server processor through the decompression card and associated FIFOs. Control of the decompression unit is in the hands of the server processor. The video mixer processor just pulls data from the FIFOs to buffer memory and copies it to the framestore at the correct place. It uses three main processes for this:

1. FIFO handler process

   The FIFO handler process takes segment slice descriptors from the server card over the link which connects the two transputers (via the buffer), and data from the FIFOs connected to the decompression card. The segment slice descriptors identify which stream they are associated with, and the FIFO handler requests buffer space on a per stream basis for the incoming video data from the buffer handler. Once a complete segment of a stream is ready its buffer information is passed to the blitter process. If a command is received by the FIFO handler to tell it to delete a stream it informs the blitter process, relying on it to free the memory cleanly after it has finished with it.

   In cases of emergency when the decompression hardware needs to be restarted the mixer software must inform the server processor to perform the reinitialisation, as the server has control of the input to the decompression unit. To do this it breaks the top-down approach to requests, and has a request channel going out of it to be multiplexed with other requests being passed on along the request path. This mechanism therefore also relies on the server being further down the request path than the mixer. It works, but it is inelegant.

2. Blitter process

   The blitter process accepts buffers for streams from the FIFO handler, and displays them by copying from the DRAM to the VRAM, clipping the image as necessary. It watches the vertical synchronisation signal from the workstation monitor to wait for the flyback period, so that when this occurs, all the streams for which it has received data from the FIFO handler are updated. This mechanism is used to reduce the tearing effects seen when updating the framestore as the electron beam in the monitor travels through the updating information.

   If the FIFO handler tells the blitter process to deallocate a buffer it removes the corresponding stream from its display list, and then passes the information to the buffer handler process.

   The blitter process is also responsible for maintaining the mask to determine which of the Pandora or workstation output is displayed at each pixel of the display, under control of requests.

3. Buffer handler process

   The buffer handler process maintains a free and a used list. If a request comes in from the FIFO handler process for buffering for a stream, or a change in allocation, it will attempt to do this without any data movement, allocating contiguous memory for the buffer from the free list, and returns this information to the FIFO handler. If this proves impossible it sends out a shuffle request to the FIFO handler in response. The FIFO handler informs the blitter process, which replies to the buffer handler. The buffer handler is then free to shuffle the used and free lists to allocate the memory if it can. It then replies to the FIFO handler, which will restart the blitter process.

2.2.4.3 Network implications

The features of real-time video data transport required by the video mixer clearly relate to those of the video capture. However, the receiving of high bandwidth real-time data does have different implications for network software than transmission. In addition the software for the mixer handles its resources in a manner which should be supported by the network software.

1. Dropping of video segments and frames

   The compression system used by Pandora does not use interframe information, so losing a video frame does not impact on the decompression of other following or preceeding frames, unlike a more complicated compression system like MPEG [17], where the loss of a frame of video will have effects lasting until the next I-frame is sent. This means the transport of Pandora video data need not be reliable, and it is quite acceptable for the network software to drop video frames from high bandwidth streams if it or the network is overloaded, or if higher priority, lower bandwidth streams need to be transmitted. Indeed, there is no inter-segment compression either, so individual video segments may be dropped without the display failing, although this is less pretty to see as it leads to parts of a video window being static whilst others remain active. This is especially noticeable if, say, the first segment of a frame usually gets through, and later segments do not, as the lower regions of a picture remain unchanging while the upper parts change as expected. [67] presents a layered video coding model which would work significantly better under these circumstances, but if segments are not dropped at all the results are always better.

2. Video transmission and reception

   The balance between video transmission and reception at overload must be made. It has been suggested that transmission of real-time data should be given precedence over its reception ([34]). The reasoning behind this is that a user who is overloading his local system should experience degradation of that system before others perceive any performance loss. The user can then take corrective action, without affecting others. There are a few disadvantages to this approach:

   To control the system a user, or agent for the user, must take corrective action. It requires monitoring of local sinks and the network interaction, to produce a reaction of controlling the streams. Implementation of automatic control would require monitoring of data missing from incoming streams, generating a response to control some (maybe different) streams. With receive priority over transmission, the network software will inform higher levels that its transmission queue is overflowing on a per stream basis, and then control of the other streams will have to be affected. This removes the need for complicated measuring of data missing from incoming real-time streams.

   As stated in [34] this principle is reversed for storage systems, where recording is much more important than playback — recording an event is only possible at the instant it happens; playback can occur as many times as desired once a recording has been made. (The flip side to this, however, is the source providing the data for recording wants the outgoing stream to have priority over its incoming streams.)

   Also as stated in [34], there is a principle of command priority over data priority. This can be handled easily at the transmission level, by degrading data transmission before command transmission, yet a degradation of the performance of the receive side is indiscrminate.

In addition, in an ATM switch network it is unfriendly not to receive data once it has reached you through the rest of the network, as it will have taken up bandwidth which could otherwise have been used more effectively.

Finally, if network entities only transmitted and never received, there is no way for automatic control to take over and tell things to throttle back, as they are only shouting not listening.

3. Resource management in the mixer software

   If a video stream is being successfully received by the mixer software then it will have a buffer allocated to it. Buffer memory on the mixer card is not a big resource, and it runs out even with fairly reasonable demands for displayed video streams. So it is important to maintain active streams in preference to others which would be dropped because of lack of buffering resource in the mixer.

2.2.5 The Server card

The server card is the main data switch inside a Pandora box. All video and audio data pass through the server card to and from the network, and even locally from audio to audio card or capture to mixer card.

2.2.5.1 Hardware

The server card contains a T425 transputer with 256 kilobytes of static RAM (for speed). It has access to the compressed data FIFOs on the capture card and the decompression FIFOs on the mixer card. Its four INMOS OS-links are connected to each of the other processors: audio (segment data in both directions); video capture (responses out, slice descriptors in); video mixer (requests in, slice descriptors out); and network (commands, replies and data in both directions).

2.2.5.2 Software

The server software divides into four main parts: a buffer allocator; sources; sinks; a central switch fed by the sources, feeding the sinks (see figure 2.8).

The buffer allocator provides buffer memory for the source handling processes, and accepts buffer freeing messages from every process which may finish with a buffer. To do this it maintains a buffer usage count, set to 1 on initial allocation, and accepts usage increment and decrement messages from processes. A process sends an increment message when it sends the buffer to another process, and decrement messages when it has finished with the buffer. On decrementing the usage count to zero the allocator frees the buffer.

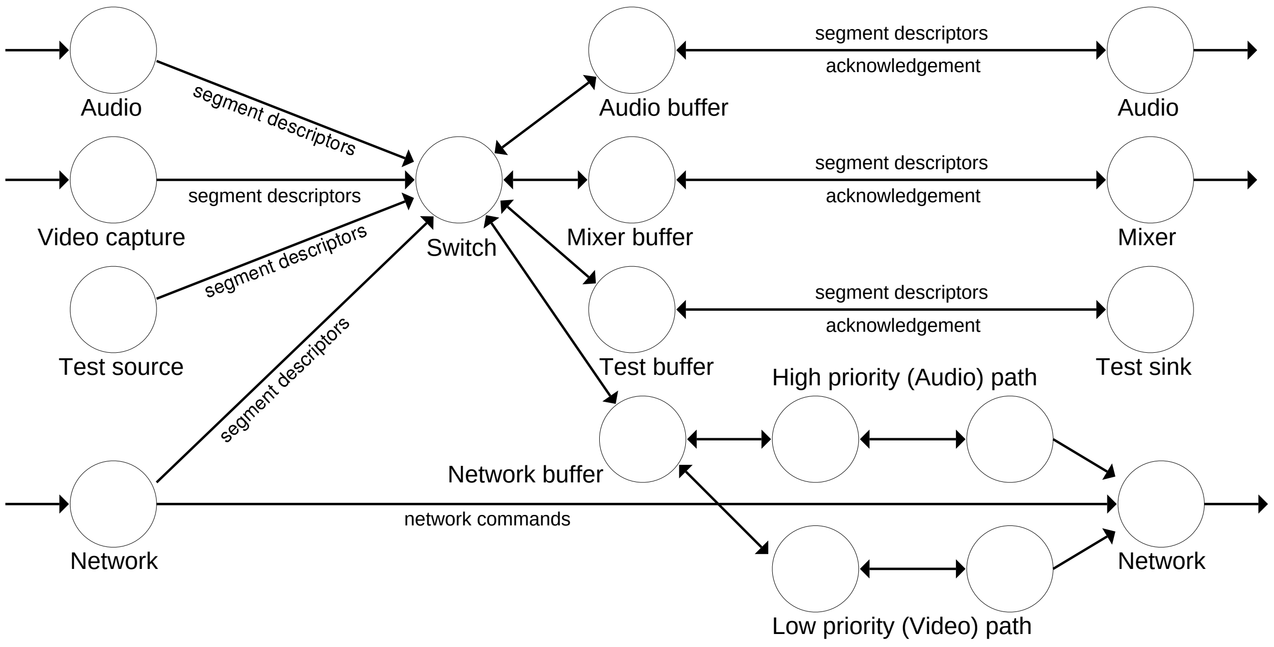


Figure 2.8: Process and Channel Structure of the Server Software

Each of the sources send segment descriptors to the central switch when either data arrives from the data sourcing cards or, in the case of the test source, when generated internally. Buffer memory for the segments must be requested from the allocator process using the above protocols.

The central switch maintains a set of destinations for each stream it may receive. When the switch receives a segment from a source, for every destination of the segment's stream that is ready to accept a segment it increments the usage count of the buffer associated with it, and forwards the data to the destination. Once the switching is complete it tells the allocator to decrement the usage count. This system lets a single source stream be sent to more than one destination, without copying of data or requiring two sources. Indeed, the audio source only provides one stream. For a video conference, the audio stream is split to two or more network destinations, and the video stream is sent to both the local mixer card and network destinations.

Each of the sink outputs (video mixer, audio replay, network out, and an internal test sink) has a handshaking buffer between itself and the switch. These processes use a circular buffer to implement a FIFO. When the switch sends a segment to be placed in the FIFO the buffer replies with a full or not-full signal, to indicate whether it could buffer another segment. If the FIFO is full then when a space becomes available it signals this information the switch. The sink outputs themselves provide a simple handshake to the circular buffers to indicate they are ready to accept a segment. The sink outputs will inform the allocator process when they have finished with a segment descriptor, telling it to decrement the usage count of its associated buffer.

The network output process does not follow this outline precisely. Instead the switch sends its segments to a splitting process, which sends video segments to a low priority buffer and audio segments to a high priority buffer. The network output takes items from the high priority buffer in preference to the low priority segments, which provides both a prioritisation of different streams and differing amounts of buffering for the two sorts of traffic.

The circular buffers for different destinations do not have the same sizes. In fact the audio sink has 12 items; the mixer and test sink process have 4 items each; the network output has two buffers as described above, with the high priority audio buffer having 6 items, the low priority buffer having 4 items. The circular buffers are intended to provide enough buffering to stop data being thrown away when the Pandora box is operating just below overload, and the sizes of the buffers were chosen by experiment to fulfil this criterion.

2.2.5.3 Network implications

The server software provides prioritisation and buffering of streams, and so is very friendly in how it supplies data to the network software. The networking software must just take data from the server at the highest rate it can do so. The output of the networking software, which links to the network source buffer of the server software, is free to send data whenever it wants. The network source buffer always tries to have a segment buffer allocated to it into which it can receive segments of data from the network without having to perform allocation before accepting it. So the total impact the server has on the network in normal operation is not too great.

However, if the network is overloaded and the circular buffers in the server start to fill they may have a large impact on the end-to-end delay of streams. So even under overload the networking software must continue to take data out of the circular buffers and throw it away.

2.3 Network Principles

The above sections detail the hardware and software designs of the Pandora system which affect the network interface. This section brings these features and requirements together to provide a summary of the facilities the network must provide and requirements it must cope with. These form the **Network Principles** for any network support for a multimedia communications system.

2.3.1 Commands, messages and control

As discussed above in section 2.1.3.4 the control of the network engine in the Pandora box should be performed using a system of commands and asynchronous event messages, with the data transfer slotting into the system with low overhead. From section 2.1.3.5, this command/message system needs to be accessible from the Pandora box, from a debugging port on an OS-link, and from the network through a debugging SAP.

To implement a debugging SAP and to implement the network access to the interface request/response system (section 2.2.1.3) on other SAPs, as well as allowing restricted access to the SAPs for Pandora streams, requires support for different types of SAP — from the single-client Pandora SAPs, through a single-client command/message debugging SAP, to multiple-client Pandora interface request/response SAPs.

With the use of the command/message system, and the logical extraction of the network engine from the Pandora box, a mapping is required from the network's connections to Pandora's internal streams (see section 2.1.3.1), both in terms of stream handle and quality of service. If the network software is viewed as a system in its own right, and so may be used in other applications, the veneer between the network and Pandora's internals must belong inside Pandora.

2.3.2 Overload handling

One of the themes of the Pandora software is handling overload situations well. A system may run fine with even a heavy load, but if an event momentarily pushes it into overload and catastrophic failure occurs then the system's usefulness drops. With a networked real-time system the possibilities for such 'event's are innumerable, hence the concentration on overload handling.

Firstly, when overload starts to occur in a portion of the network subsystem the rest of the Pandora system should be informed as soon as possible (see section 2.1.3.1). It is important, however, not to continuously send messages indicating the overload, as generating and handling these messages can easily worsen the situation. A good option is to send messages regularly at intervals which are long enough for the overloading to have a chance of stopping. Using this mechanism the host can reduce its requirements when overload occurs, and use a timeout after the last overload message to increase its requirements again.

When an overload situation has been detected then any data which will be lost due to the overload should be discarded. This means, for example, that when data is transferred from the host for transmission it is discarded immediately, rather than being sent to the transmission process where it will be discarded by an overloaded transmitter. This principle should be extended throughout the system, using the overload messages mentioned above. However, the second principle of local adaptation to overload conditions also applies: in some cases it may not be possible for a data source to back off, or it may be infeasible. This implies that, even under overload, the network should keep sinking data from the server card as fast as possible.

It is usually when under overload that the prioritising of streams becomes most important, and that is discussed in more detail below. However, as Pandora segments are sent as whole blocks, and partial segments are of no use to data sinks, another prioritising issue becomes relevant, that of active streams having higher priority than inactive streams. As an example, if the network card is overloaded on reception, receiving data on three streams, then, if data arrives for a fourth stream, that data ought to be discarded. This helps both stream independence (see section 2.1.3.1) and reduces the effects of the overloading.

One issue that must always be kept in mind during software design for overload conditions is that the complexity of handling overload should not significantly reduce the performance, and therefore the threshold at which overload occurs. It is not much use having a card which can cope with overload well, if it becomes overloaded with only a single video stream, where a simpler implementation of overload handling may allow many more streams.

2.3.3 Quality of service

With the many classes of network connections required by the Pandora system (Pandora interface control, command/message debugging, audio, video real-time streams) different qualities of service must be supported, in terms of reliability, buffering, bandwidth, priority, latency and jitter, and tolerance on these. Each of these is discussed in more detail below. It should be noted, however, that support for different qualities of service is required on a per stream basis, and therefore the command/message system must include control of the quality of service.

2.3.4 Data reliability

The various stream styles inside Pandora have different requirements when it comes to reliability of transport through the network card. For reliability we refer only to providing guarantees that the stream's data arrives at its destination. If it arrives at the destination then it must do so without corruption (see section 2.1.3.2).

Firstly, from section 2.1.3.5, a reasonably reliable transport mechanism is required to debug the network card over the network, although it is not a disaster if some of the commands or messages fail to get through when debugging. However, from section 2.2.1.3, the requests and their responses occurring over the Pandora interface network access require guaranteed reliable transport.

The real-time data streams are different. From section 2.2.2.3 the audio data streams have no apparent degradation if 1 in 7500 segments fail to arrive, and no guarantee of data transport is required. The video data (section 2.2.4.3) may suffer from even more data loss without the system becoming unusable.

2.3.5 Data buffering

Buffering of data is required for both transmission and reception. Data reception buffering should be as large as is needed, assuming the rate at which data is removed from the network card by the host is greater than its arrival rate from the network itself. If this assumption holds then the total amount of receive buffering required can easily be calculated, using the number of incoming streams and the size of their segments.

Buffering for data transmission is a different issue. The Pandora interface network access only requires a single block of data buffered, as each block sent must be acknowledged to ensure reliable transport. To debug the network card over the network there needs to be enough buffering to cope with the messages which may occur and need to be transferred to the remote debugger. Audio streams require enough buffering to avoid data loss due to jitter caused by resource conflicts (e.g. demand for processor cycles during simultaneous transmission and reception) while keeping the buffering small to reduce data transport latencies (see section 2.2.2.3). Allowing for a jitter of 5 milliseconds, audio streams would require 8 milliseconds of buffering. Video streams require buffering to cope with the disparity between the rate that video data arrives from the host and that at which it can be transmitted. From section 2.2.3.3, 150 milliseconds is sufficient.

Only the Pandora interface knows how many Pandora segments a certain time corresponds to for audio and video streams, and so only it can define the number of blocks buffered. So, to cope with this and the other streams' requirements, the amount of buffering allocated to each stream must be settable from a command/message. When an attempt to exceed the buffering allocation is made appropriate messages must be generated, indicating the discard of the data. In the case of transmission this messages cannot be considered as overload messages, as they are directly in response to an explicit request.

2.3.6 Data bandwidth

The various stream types have different bandwidth requirements. The Pandora interface network access and the debugging channels are low bandwidth, well under 100 kbps. The audio streams are not much higher bandwidth (136 kbps, see section 2.2.2.3). The video streams, however, are up to 2.8 Mbps. The network must cope with both the low and high bandwidth streams. This may be handled with two different algorithms, or with a single algorithm which can handle both. It should be noted, however, that the bandwidth of a stream is dynamic. For example, a Pandora video source may have its compression or frame size changed while the stream is active, leading to either low or high bandwidth streams.

In addition to the actual data bandwidth requirements there is the issue of the size of blocks which the data is divided into. For example, a stream with a bandwidth of 8000 bits-per-second may consist of 1000 blocks per second, each of one byte. In the data transmission path there will usually be code executed on a per-block basis, which implies that the low bandwidth stream may require a lot of processor time. The low bandwidth audio streams can consist of 250 blocks-per-second, compared to the high bandwidth video streams which require at most 50 blocks-per-second (2.1.3.2). It is important for the network to cope with both of these loads.

2.3.7 Data priorities

When the network must handle more than one stream at one time there must be some prioritising taking place. As discussed in section 2.2.4.3, the video streams can be of lower priority than, for example, audio streams. It may be deemed that the priority of a stream is inversely proportional to its bandwidth, or its requirement for processor time, or some other constraint. However, this should be left for the host to decide.

Section 2.2.4.3 also discusses the prioritising of transmission and reception of data. This is perhaps more difficult to handle dynamically in software than prioritising streams, and so requires an earlier design decision. The conclusion arrived at in section 2.2.4.3 is that reception of data should have priority over the transmission of data, although load sharing is desirable.

2.3.8 Data latency

The latency of the data transport across the network can have a large effect on the usefulness of the data when it arrives. The latency of debugging commands/messages can be fairly large, in the order of many tens of milliseconds. As discussed in section 2.2.1.3, the Pandora interface network access was, in part, added to improve the performance of the control of the Pandora box, so the latency of the requests and responses must be below one millisecond. Section 2.2.2.3 discusses the latency requirements of the audio streams, arriving at a figure of 5 milliseconds. The video requirements are less stringent, as the main problem is lip synchronisation with the audio streams. In face-to-face conversations people see lips move before the sound arrives, yet after capturing the video and buffering it for display there will be a delay of tens of milliseconds, more than the audio streams' latency. However, it is perceived that a time-lag of 100 milliseconds is still acceptable, especially on the low quality Pandora video streams where lips can be hard to see.

To achieve the low latencies for the audio and interface network access their streams could be allocated high priorities. Even then it is important for the network card to provide a low latency path for the data on those streams. This rules out, for example, unnecessary copying of data on those streams. (See also section 2.2.3.3.)

2.3.9 Data jitter

Any reasonable jitter in the non-real-time streams will not affect their performance. However, jitter in the audio and video streams requires careful handling. For example, the clawback buffering in the audio playback path helps to remove a small amount of jitter (see section 2.2.2.3), but suffers when a large amount of jitter (tens of milliseconds) is inserted into a stream, as there is a long-term effect on the buffering. The video streams suffer a lot less, mainly due to the large inter-frame time (40 or 80 milliseconds, or more). If there is jitter of the order of an inter-frame time then the motion starts to appear jerky.

2.3.10 Data transfer on OS-link

The network card is connected to the server card by a single INMOS OS-link. This can be viewed much as the network is viewed, as a reasonably fast data pipe. However, data is easily multiplexed over the CFR or ATM network, but much less easily over the OS-link. If the server-to-network OS-link is considered to be a much higher bandwidth connection than the network itself then the simplest handling of data transfer, first come first served, may suffice. It is only when the arrival rate of data from the network approaches or exceeds the OS-link speed that other considerations need be made.

One issue is important regardless of the relative speeds of network and OS-link. That is, there should be some attempt to decouple the network from the host to get maximum performance from both, using appropriate buffer processes. The server processes on the Pandora box perform some buffering when communicating with the network, and a similar system is required in the network software.

Chapter 3

Pandora network software implementation

This chapter describes the implementation of the networking software for the Pandora system. There have been four different hardware subsystems that Pandora has used for its network: the first implementation used the server card for the network and supported UDL; the remaining three put the network subsystem onto a seperate card for better performance, and supported MSNL [44]. The hardware of all four is described in the first section. The software for the last three hardware implementations is the basis for all the work in the thesis, and all three implementations have used the same basic structure; the following two sections therefore describe in detail the first of the three implementations, with the differences between the implementations described in the last section.

Throughout the chapter the term 'host' is used to refer to the system which uses the network subsystem: in the case of Pandora this is the server card.

3.1 Hardware descriptions

The Pandora networking subsystem has had four phases:

- Server-based CFR interface

  Initially the server card contained a CFR interface, and network access using UDL (Unison Data Link, [62]) was handled by the server software. The server processor had to handle interrupts from the CFR chip for transmission and reception, and perform segmentation and reassembly of segments to fit the 28-byte data size of the UDL payloads, in addition to its stream switching requirements.

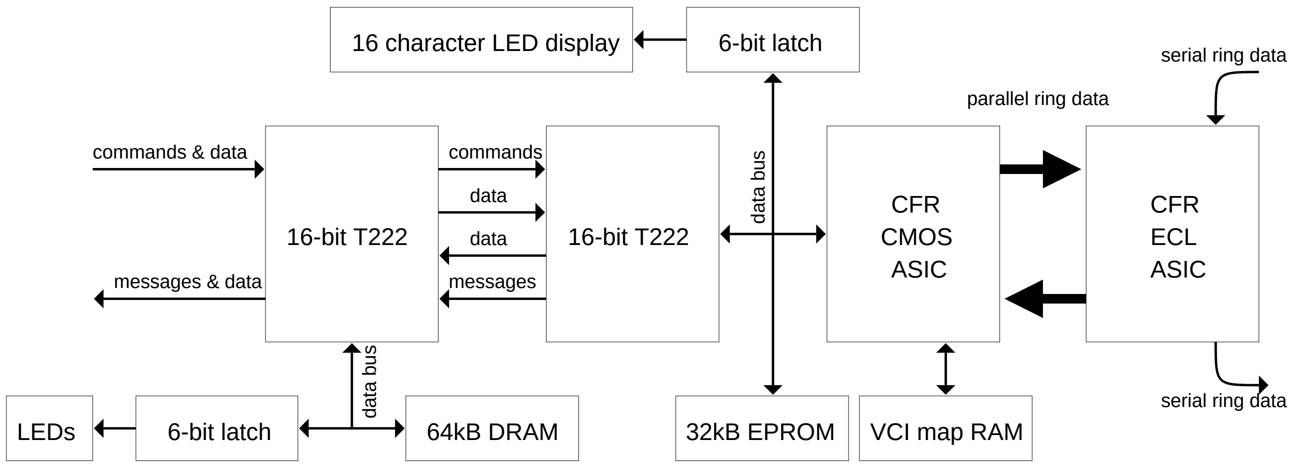


Figure 3.1: T2 CFR network card hardware

- 16-bit CFR network card (the T2 card)

  To improve the network performance of the Pandora system an extra network card was designed containing two T222 transputers [31], one with 64 kilobytes of DRAM, the other with only 4 kilobytes of internal RAM. This latter processor [14] acts as an intelligent interface and buffer to the CFR CMOS ASIC ([54], [2]), see figure 3.1. The two processors are connected together with two INMOS OS-links. With one of the OS-links of the main processor connected to the server transputer there is a spare link for debugging. Also there is a 6 bit latch writable by this processor, of which two outputs drive LEDs. The other transputer has access to a 16 character LED display, for informational purposes or debugging.

- 32-bit CFR network card (the T4 card)

  The third version of the network card hardware replaces the main processor of the previous card with a T425 transputer with more memory, 2 megabytes, while retaining the CFR-controlling T222 transputer (see figure 3.2). This enhancement was mainly to add to the speed of the network interface, which it was felt was still a bottleneck. Fortunately after the upgrade to this board, and appropriately rewritten code, the network interface (i.e. CFR CMOS ASIC) itself seems to be the bottleneck.

- 32-bit ATM network card

  The fourth version of the network card hardware is similar to the previous card with the same processors, but with a 16-bit ORL ATM fast packet switch network interface instead of a CFR interface (see figure 3.3). The 16-bit ATM switch network interface consists of a 16-bit wide FIFO for cells for transmission and a 16-bit wide FIFO for cells for reception. These

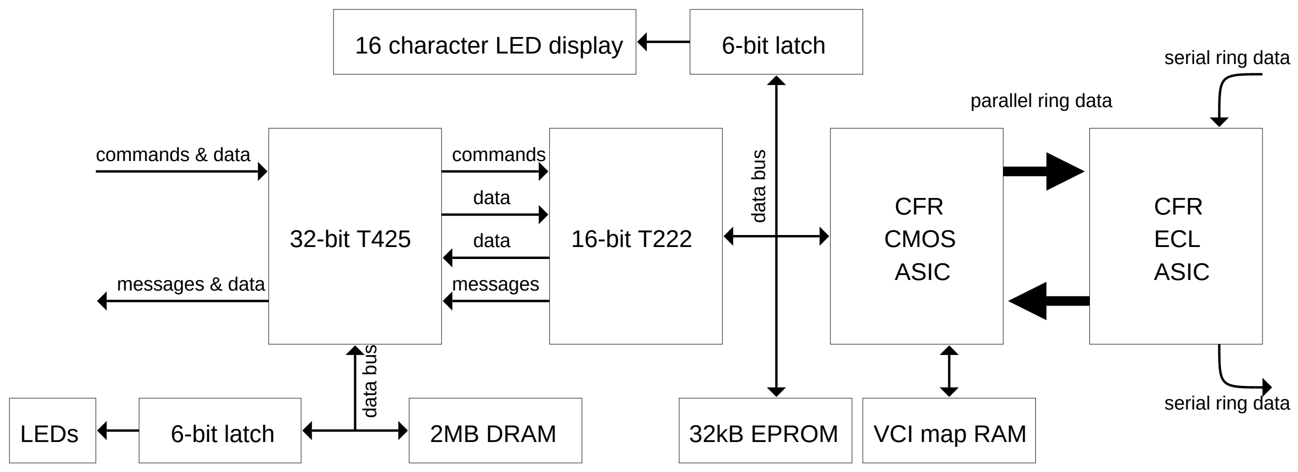


Figure 3.2: T4 CFR network card hardware

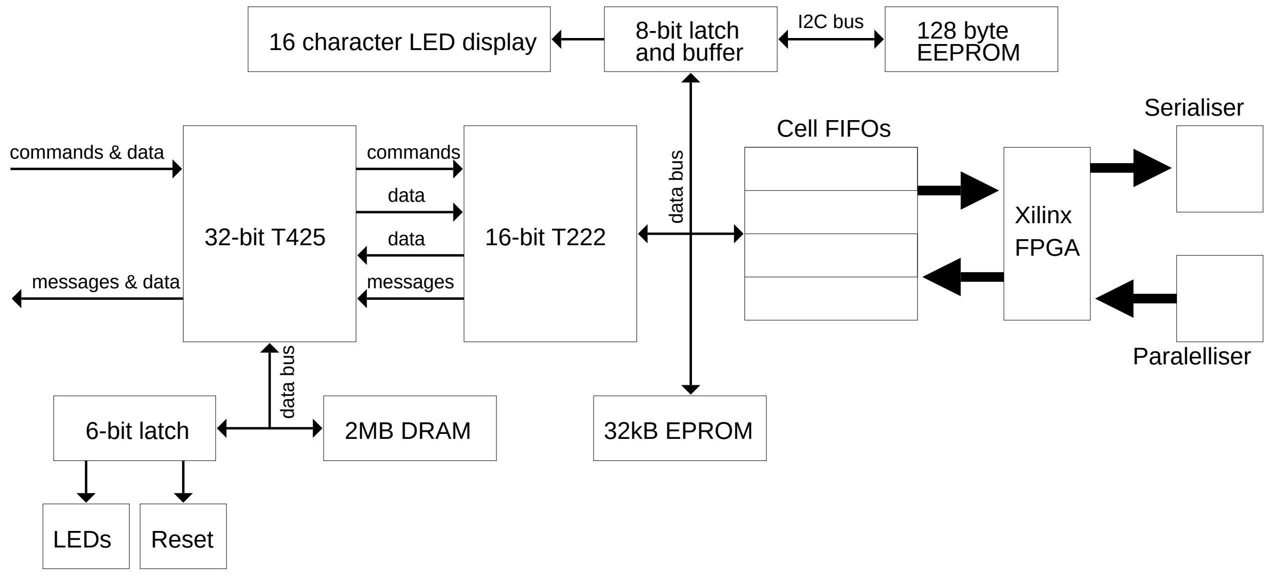


Figure 3.3: T4 ATM switch network card hardware

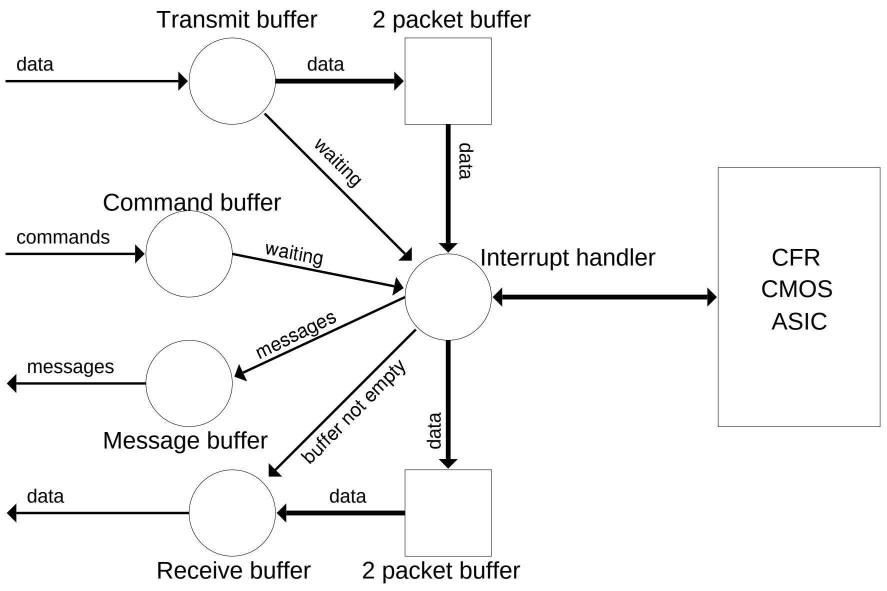


Figure 3.4: CFR slave transputer software structure

FIFOs are connected to a Xilinx FPGA, which is coupled to two AMD Taxi chip parts [3] to serialise and deserialise the cell data. (The Taxi chips were designed for FDDI, but are encompassed by the ATM Forum for 100Mbps multimode fibre [5]. In this application 50Ω coaxial cable is used as an alternative physical layer, and the transmission convergence layer uses additonal hardware handshaking [24].) The T222 transputer has access to the control signals generated by the FPGA, and the FIFO flags, so it can cope with disconnections from the network, a blocked local switch, and other local network failures. In addition, for reliable continuous running of a physically remote ATM network object, the T425 transputer can reset itself and surrounding boards. The T222 transputer can still access the 16 character LED display, as for the previous two boards, but it can also act as the master on an I^2C bus, to which a 128 byte EEPROM is connected.

In the three cases of the seperate network cards, the main processor (T425 with two megabytes or T222 with 64 kilobytes of memory) is termed the master processor, and the other T222 transputer is termed the slave processor. The slave processor has a 32 kilobyte EPROM, from which the network card processors can boot.

3.2 CFR slave transputer software structure

The CFR slave software [13] was written by David Clarke prior to the start of the research for this thesis. It is described in this chapter because of its close relationship to the master transputer software that is the main part of this chapter, and also for comparison with the ATM slave software which is part of the research. The CFR slave software could have been rewritten as part of the research, but after a detailed examination of its structure and implementation with regard to the network requirements of the Pandora system the decision to retain David Clarke's work was made.

The CFR slave software provides limited buffering of CFR packets for both transmission and reception. It consists of five main processes: a transmit buffer, a receive buffer, a command buffer, a message buffer and an interrupt handler. The transmit buffer and interrupt handler share access to a two-packet-long transmit queue. The receive buffer and interrupt handler share access to a two-packet-long receive queue.

Communication to the master transputer occurs over two OS-links. Although the links are bidirectional, because of the communication protocol used by the OS-link hardware with an acknowledge bit pair in the reverse direction per byte transferred [30], the two high bandwidth channels, of transmit and receive data, are not placed onto the same OS-link. Instead, one OS-link is used for the transmit data to the slave and messages from the slave. The other link carries the receive data from the slave and commands to the slave.

3.2.1 Interaction with the master transputer

The transmit buffer waits to receive a packet from the master transputer over an OS-link into one of the two transmit queue entries. It then generates an interrupt to inform the interrupt handler of the valid transmit buffer entry, before looking for another packet from the master transputer, which would be read into the other transmit queue entry. Should this second packet be received before the interrupt handler has finished with the previous packet the transmit buffer will inform the interrupt handler, blocking on that communication until the interrupt handler is ready. At this point the first transmit queue entry will no longer be in use, as the packet data will have been transmitted, so the transmit buffer again looks for a packet from the master transputer. If the master transputer does not keep on sending packets the transmit buffer will restart, waiting for a packet and generating an interrupt. This mechanism reduces the number of interrupts and inter-process communications required over simpler methods, hence improving performance.

The receive buffer is informed by the interrupt handler whenever a packet has been received into the two-packet-long receive queue, and it forwards the packet to the master transputer over an OS-link. The other entry in the receive queue can be filled by the interrupt handler while this is happening, so providing a little decoupling of the reception of packets by the CFR interface from the master transputer.

The command buffer accepts commands from the master transputer over an OS-link, generating an interrupt to inform the interrupt handler whenever this occurs. The commands provide full access to the CFR interface as well as software initialisation and writing to the 16 character LED display.

The message buffer provides a path from the interrupt handler to the master transputer over an OS-link for responses to commands and for messages about asynchronous events, like ring breaks. It provides a single stage of decoupling so the interrupt routine may continue to execute while the master transputer is given a message.

3.2.2 Interrupt handling

The interrupt handler does most of the work. There are six interrupt sources: the transmit buffer; the command buffer; four interrupt pins on the CFR CMOS ASIC (ring broken, packet transmit FIFO empty, packet receive FIFO full, packet TOGged). The interrupt sources are prioritised in software into the following order (highest priority first):

- Ring broken (framing error)

  If the CFR framing structure is not correct the CFR CMOS ASIC reports this by raising the ring broken interrupt. The interrupt handler responds to this by masking out the CFR interrupts to prevent more failures occuring before the master transputer has responded. The master transputer is informed of the failure, and when it has recovered from this it must turn the CFR interrupts back on with the appropriate command to the slave transputer.

- Packet received by CFR (receive FIFO full)

  When a packet is received by the CFR chip from the ring it is placed in the receive FIFO. This FIFO is only one packet deep, so to get the best receive performance from the interface this interrupt must be the highest priority (of the frequent interrupts). When it occurs the packet is copied from the FIFO into the receive queue and the receive buffer process is informed. Should the receive buffer still be transmitting the other receive queue entry to the master transputer, the interrupt handler will block, waiting for the receive buffer to complete the transfer and collect the new packet. In this way no data is thrown away inside the slave transputer on reception: the CFR interface may fill the receive FIFO again, and if another packet arrives on the ring which should be received the CFR interface will mark it as TOGged, forcing retransmission. (TOG stands for Thrown On the Ground.)

- Packet TOGged by the CFR

  When an attempt to transmit a packet on a VCI has failed, perhaps because the receiver's FIFO is full or no other node is listening on the VCI, the packet transmission will be retried a certain number of times. If all the retries fail the CFR CMOS ASIC raises the TOG interrupt and its "transmit fifo empty" interrupt. To correctly inform the master transputer that a transmission has failed the TOG interrupt is handled first, at higher priority than the transmit interrupt. It causes a transmission failure message to be generated to the master transputer, and it sets a flag to suppress the generation of a success message on the following transmit interrupt.

- Packet transmit FIFO empty

  This interrupt is used by the slave transputer to indicate a packet has been sent by the CFR CMOS ASIC and that there is space for the next packet to be transmitted. If a previous TOG interrupt has not occurred the interrupt handler deduces the last packet inserted into the FIFO has been successfully transmitted, and informs the master transputer of that fact. The interrupt handler then checks to see if the transmit queue shared with the transmit buffer process contains another packet for transmission, in which case that packet is inserted into the CFR CMOS ASIC.

  This interrupt is only enabled if a packet has been inserted into the CFR CMOS ASIC's transmit FIFO.

- Transmit request from transmit buffer

  If the transmit buffer receives a packet from the master transputer for transmission on the CFR after an idle period, the transmit buffer generates this interrupt to alert the interrupt handler. The interrupt handler then takes the new packet from the transmit queue and inserts it into the CFR CMOS ASIC, and enables the packet transmit FIFO empty interrupt.

- Command from command buffer

  In a similar manner to transmit buffer interrupts, commands received from the master transputer will cause the command buffer to generate an interrupt. This causes the interrupt handler to request the received command from the command buffer, and then to obey it.

3.2.3 Priorities and overload

The interrupt handler and receive buffer operate at high priority, and thus run in preference to the other processes, and are not preemptable. This helps enforce receive priority over transmissions (see principle 2.3.7).

However, it should be noted that a snapshot of the interrupt sources is taken at the time the interrupt handler is scheduled, and the active interrupt sources in the snapshot are all acted upon before a new snapshot will be taken. So continuous receive interrupts do not stop command interrupts just because the receive interrupt has a higher priority. This helps to fulfil the stream independence principle (see principle 2.3.2).

The CFR will introduce jitter in network transfers, especially if the receiver for a cell has not emptied its receive FIFO from a previous cell, so the fast removal of data from that FIFO helps keep the jitter low (see principle 2.3.9). The simplicity of the software design and prioritising of the receive buffer and interrupt handler help in that respect.

The buffering in the slave software has to be kept small due to memory restrictions, but it is tuned to the smallest required to acheive the CFR bandwidth. This is a balance between data bandwidth and data latency principles (2.3.6 and 2.3.8).

It should also be noted that the CFR provides guaranteed delivery of data which is transmitted, and the slave software never discards data or messages, so data reliability comes for free (principle 2.3.4), and any overload in the network is reflected implicitly for the master transputer to handle, where more knowledge of the data conditions means the complexity of the handling of overload is less and the chances of reporting it correctly to the host are higher (principle 2.3.2).

3.3 Master transputer software structure

This section describes the implementation of the master transputer software for the first CFR network card containing two 16-bit T222 transputers. The software divides neatly into six parts: host interaction; slave control; data transmission; data reception; connection management; debugging facilities. A section for each is split into descriptions of its seperate processes, followed by an examination of how the network principles affected their design. Firstly, however, there are some issues common to all the software design.

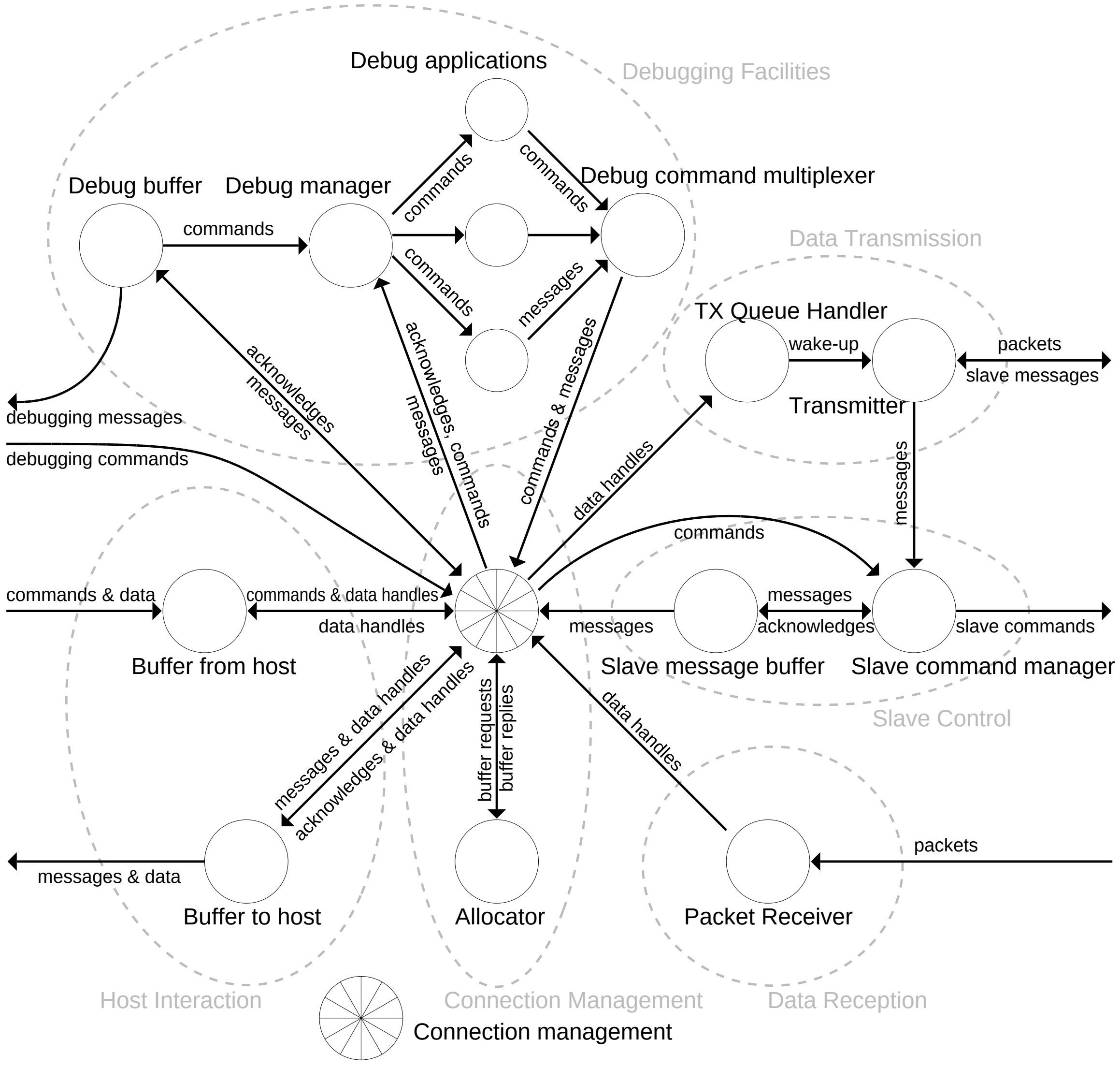


Figure 3.5: Master transputer software structure

3.3.1 Master transputer fundamentals

3.3.1.1 Buffer allocation

The remains of the memory not required by stack space, data space or program code is used as the data buffer. The buffer memory in the last implementation of the T2 card's code is 37 kilobytes long. It is divided into areas 36 bytes long, all initially placed on a free list. These areas are used to contain lists of data packets for transmission or reception. A block of data for transmission or reception is defined by a single header area, which contains fields describing which connection the data belongs to, block size, etc., and a pointer to the packets of data which form the actual data of the block. Each packet of data for a block takes one 36 byte area, and the packets are joined together as a linked list (the 36 bytes is divided into 32 bytes of data, and the linked list pointer).

Allocation of the buffers from the free list, and deallocation to the free list, must be done by high priority processes, so that the transactions are atomic. Those processes which are not run at high priority must use a high priority allocation process to perform the alllocation and deallocation from the global area. This is the case for the *connection management* and host interaction processes.

The buffer allocation strategy was chosen as the available buffer space (e.g. 37 kilobytes) is not large compared to the size of blocks of data required to pass through it (see section 2.1.3.2). An alternative strategy considered was to divide the area into video sized blocks (larger than 4096 bytes), and small audio or command sized blocks (less than 128 bytes), and allocate from these. This has the disadvantage of limiting the buffer space for video data to, for example, eight segments (32 kilobytes). This must include all segments being transmitted (arriving from the host and segmented into the slave), and those being received (reassembled from the network and transmitted to the host). Control of the segments being transmitted could be performed by managing input from the host. However, arrival of many segments from many sources on the network may occur simultaneously, and it is quite reasonable to expect up to eight segments to be being received or delievered to the host at one time. Should nine segments need to be received, one would have to be discarded at its beginning, even though the buffer space may become available as another segment completes its journey to the host. The chosen strategy also provides some independence of transmission and reception of data. The system would have to be much more heavily loaded for the transmission of blocks to fail due to the allocation to received blocks, or vice versa — the apparent buffer space is greater under the chosen scheme, as blocks for transmission may be freed in parts as their packets are successfully transmitted, and the exact amount of data for incoming segments can be allocated as the packets arrive. Another benefit is that each stream can have different buffer sizes allocated to it at no extra cost, so providing that quality of service parameter per stream, as required by principle 2.3.3.

3.3.1.2 Message and command structure

Commands and messages inside the master transputer software, as well as those to and from the host, are always of the form *length in words of command; command type; command arguments...*; this includes data blocks for transmission. The word size used by the commands and messages is 16 bits, as the master transputer is a 16-bit T222 transputer.

Command types include connection maintenance (make and kill), system maintenance (reset), informational (connection status interrogation, transmit/receive status) and data transmit. Message types correspond in general to the command types, as commands will usually generate a message. They include connection maintenance (made, killed, timed out), system maintenance (system reset okay, reset failed, network failed), informational (alive, transmission information, reception information) and data received.

Commands and messages are specified to be of a maximum length of 32 words, except for data handling commands/messages, whose length can be up to the maximum transmittable/receivable by the master transputer software.

Messages can be routed either to the host OS-link or to the debugging OS-link, by using a lookup table indexed by the message number: many messages would be ignored by the host, and are useful for debugging, whereas other messages must be sent to the host for correct operation. Some messages lie in a grey area, where they may be of interest to either system, and so the destination may be changed. This mechanism also allows unrequired messages to be discarded by being sent to a message sink. This all helps to fulfil the principle 2.3.1.

3.3.1.3 Handshaking buffers

The occam message passing system ([26]) supplies unbuffered message passing between processes. This provides a simple way of synchronising processes, which can be useful. However, it is frequently the case that a process needs to be designed so that it never blocks on an output channel, as it may have critical time paths on its input channels. To accomodate this it is a common practice to insert handshaking buffers on these channels [25]. These use an additional back channel to indicate that the handshaking buffer may accept another message. After placing a message into the buffer, sending another is not possible until the buffer has returned with this handshake. This places an additional performance penalty onto the user of the buffer, as another input channel must be monitored, so their use must be carefully thought out.

3.3.2 Host interaction

The processes which interact with the host perform two operations. Firstly they simply provide the decoupling required by the host (in Pandora, the server card): any overload conditions in the master transputer software will not effect the server as badly as it might due to the buffering the presence of the processes provides. Secondly they provide block disassembly and reassembly for data transmission and reception.

To provide buffer allocation (for *buffer from host*) and buffer freeing (for *buffer to host*) there are two suitable processes which are not included in figure 3.5. Both are high priority (so their operations are atomic, see section 3.3.1.1).

3.3.2.1 Buffer from host

Commands from the host are read by the *buffer from host* process. The length field and first two words of the message are read first into local workspace. The command type is then examined to see if the command is for transmitting data, in which case a large command must be expected. If not, the rest of the command is read into a local buffer of fixed size, then forwarded to the *connection management* process. This may cause the buffer process to block, which is undesirable, but the *connection management* process is designed to be ready for commands at all times, so it should not block for long.

A data transmission command from the host requires the allocation of areas from the buffer memory. A header block is allocated, into which the connection to which the buffer belongs is written. Blocks are then allocated, filled with the data by reading directly into them from the host, and then they are linked together. The header block is set to point to this buffer data, and a command is sent to the *connection management* process for it to validate and execute the transmission.

3.3.2.2 Buffer to host

Messages from the *connection management* process which are destined for the host are sent to the other host buffering process, *buffer to host*, which is a handshaking buffer (see section 3.3.1.3). This takes the message and transmits it directly on to the host, unless the message type is *data received*. Then the message includes a data handle and a pointer to the block descriptor for the received data. The buffer generates a message of type *data coming* which it sends to the host, describing the stream and size of the coming data. Then a *data received* message header is generated and sent. Finally the data portion of the message body is sent directly from the list of buffer areas pointed to by the block descriptor.

Once a message has been sent to the host the buffer sends an acknowledgement signal back to the *connection management* process, so that it knows when it can send another message through the host buffer without blocking.

3.3.2.3 Network principles in action

The host interaction processes are basically pipes, and so do not handle any prioritising nor provide complicated management of different streams — this is left for the *connection management* process. However, the buffering processes do support some of the network principles in the following ways:

- Decoupling from server

  The *buffer from host* process sinks the data from the server as quickly as possible. This continues even when the network or network software is overloaded, provided the *connection management* process has the capability to discard the data and messages it produces. In the same manner the network is decoupled from the server, should that be overloaded, by the *buffer to host* process. This helps satisfy the requirements of overload, principle 2.3.2, and decoupling the network from the server, principle 2.3.10.

- Guaranteed command and message delivery

  The handshaking of the *buffer to host* process, and the blocking nature of the *buffer from host* process, provide a guarantee that all commands and messages are either delivered or a suitable message generated to indicate that delivery was not possible. This helps satisfy principle 2.3.4, providing support for reliable data transport.

- Low overhead for a data block

  The handling of commands and messages for data transmission and reception is simple, and has a low overhead per data block, requiring only a fast buffer allocation/free operation and some parameter extraction. This low overhead is required for high bandwidth streams, and is a bonus for low bandwidth streams, as described in principle 2.3.6. The mechanism for data transmission allows an overloaded stream's data to be discarded as it arrives from the server, the earliest possible point, as required by principle 2.3.2.

- Latency proportional to size

  Principle 2.3.8 states that there should be a low latency path for high priority streams. The latency of the host buffer processes depends entirely on the size of the data to be transferred, as it must all be buffered in the transputer's memory before being handled. To provide a low latency path, the blocks used by the host should therefore be small to keep to the principle.

- No data copying

  As the data is written into memory from the link on transmission, and read from memory to the link on reception, in lists suitable for the transmission and reception processes, no data copying is required throughout the rest of the system. This helps reduce the latency of the path (principle 2.3.8) and, when the CPU or memory bus is the bottleneck, increases the bandwidth available (principle 2.3.6).

3.3.3 Slave control

The CFR interface requires a great deal of maintenance if it is to operate correctly. The slave transputer provides a sensible interface to the CFR CMOS ASIC to make this simpler, and the slave command and message processes provide the overall management.

3.3.3.1 Slave command manager

The *slave command manager* takes general commands from the *connection management* process (reset, open VCI, close VCI, check status, for example) and converts them into sequences of commands for the slave transputer. The replies that these commands generate, and any other messages the slave may generate, are also received by this process via the *transmitter* process. Any special messages are converted into real messages, using the full message structure described in section 3.3.1.2, and are forwarded to the *slave message buffer* process.

In the case of a reset command the full sequence of commands described in [13] is performed. This provides a complete check of the local ring hardware, and some checking of the integrity of the ring itself. It takes about thirty seconds to complete.

At intervals of one second the *slave command manager* process also sends commands to the slave transputer to update the 16 character LED display with a string which resides in memory shared by all the processes on the master transputer. This lets any process display exceptional information for debugging purposes, and it allows some informational messages to provide users with confidence that the network is working. This display has also been used for logging results of performance tests.

3.3.3.2 Slave message buffer

The *slave message buffer* decouples the *slave command manager* process from the *connection management* process. It is a handshaking buffer (see section 3.3.1.3, and so buffers a single message from the *slave command manager* process, replying with an acknowledgement once it has successfully forwarded the message. In this way the *slave command manager* process does not block on attempting to send a message to the host, so the *connection management* process can always send commands to it without fear of blocking or even deadlock. The single exception to this is during reset, as it was decided that it should not be interruptable. In this case the *connection management* process must remember when it sent a reset command to the *slave command manager*, and send no more commands until it receives a reset reply in return from the *slave message buffer* process.

3.3.3.3 Network principles in action

The management of the CFR does not effect the network features provided by the network software nor the requirements of the Pandora system. Indeed, the slave control software is basically an attempt to prevent the unreliable nature of the CFR from impacting on the Pandora system, by handling any unusual circumstances locally. This includes maintaining the ring interface, for example by periodically checking that the VCI map RAM is not ‘decaying’, which tends to occur as the DRAM which contains it is not correctly refreshed by the hardware.

3.3.4 Data transmission

One of the most important aspects of the networking software is the management of data transmission. Not only should the maximum total possible bandwidth be achieved, but also the streams transmitted should be interleaved and prioritised for the best Pandora performance. To achieve this the data transmission is split into two processes. Both use and maintain the data structure in figure 3.6, which shows that a list of transmittable streams is kept in order of priority; the list entry for each stream is then a list of block descriptors for transmission on that stream; each block descriptor is also a list, this time of packets for transmission with a header containing reassembly information, as created by the *buffer from host* process.

3.3.4.1 TX Queue handler

When the *connection management* process decides to send a block of data, either given to it by the *buffer from host* process or generated internally, it sends the

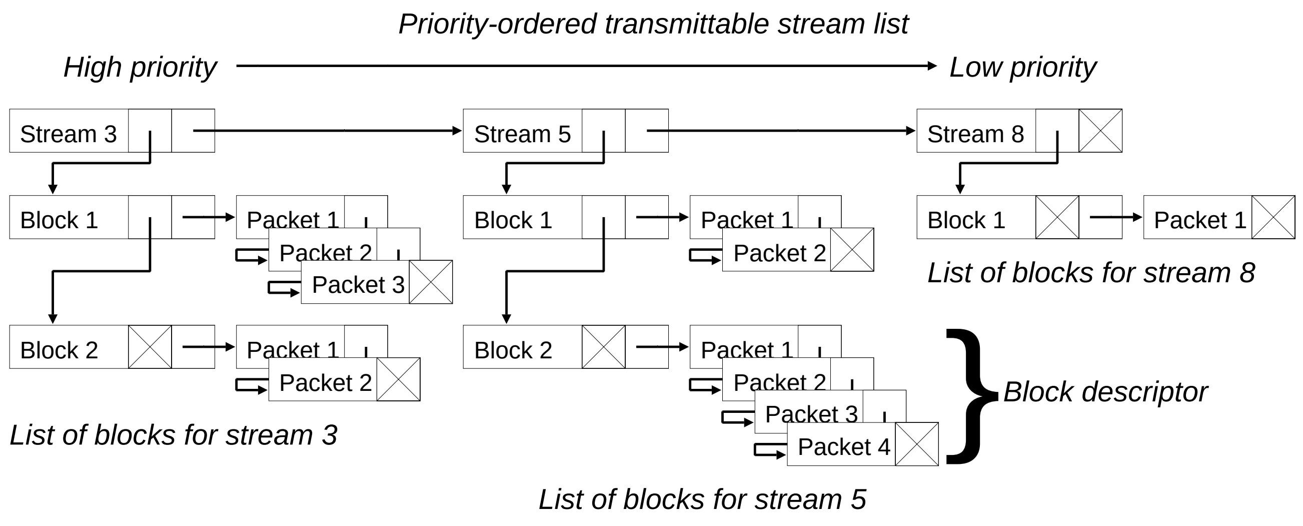


Figure 3.6: List of transmittable streams

block descriptor to the *TX queue handler*. This process then adds it to the appropriate stream's list in the list of transmittable streams, creating a new stream list entry if the stream is not currently transmitting. If the *transmitter* process has marked itself as idle at this point then it will be sent a message to wake it up.

To perform all the list operations atomically, and so that the wake-up message may be generated without possible logic races, the *TX queue handler* process runs at high priority.

3.3.4.2 Transmitter

The *transmitter* process is responsible for taking packets from the block descriptors in the list of blocks in the entries of the transmittable stream list, and sending them to the slave for transmission.

In response to each packet given to the slave a message will be returned by the slave indicating the success or failure of the transmission of that packet. Under MDL ([7] and [8]), where out-of-sequence packets cannot be reassembled correctly, it is important to ensure that a packet from a block has been successfully transmitted before sending the next packet in that block. So in order to keep the CFR busy and hence the slave transputer's transmit FIFO full, the transmitter process may insert up to two packets into the slave from different streams. In cases where there are three or more streams with blocks for transmission the transmitter starts another buffer process, and maintains three packets in the transmission path at once.

If a packet should be successfully transmitted then it is removed from the first block of its stream's list, the next packet's reassembly information is computed, and the new packet is then sent into the transmission path. The old packet's buffer space is then freed. After successfully transmitting a whole block, the next block in that stream's list will be started on. Should there be no more blocks to transmit on that stream, the highest priority stream which is neither transmitting nor backed off will be started on.

If a packet is not transmitted correctly (i.e. it is TOGged) the slave's response will indicate this. The *transmitter* process then marks the stream as backed off. Some time later the stream will be marked again as ready, and transmission will restart at the failed packet. If the packet should be TOGged too many times the whole stream's data will be thrown away and the host informed, using the message channel to the *slave command manager*.

As all the messages from the slave must come to the *transmitter* process in case they refer to the success or failure of transmitting packets, any other messages must be forwarded to the *slave command manager* process. These messages tend to occur only in exceptional circumstances, but even so the *slave command manager* process has to be designed not to block the *transmitter* process if possible.

When idle the *transmitter* process will be waiting for a wake-up message from the *TX queue handler*. This message indicates the transmittable stream list is not empty. Once awake the *transmitter* marks itself as no longer idle, and transmits the blocks for the transmittable streams. Should it ever empty the transmittable stream list it will mark itself as idle, and again wait for a wake-up message.

To provide the best transmission performance possible the *transmitter* process runs at high priority. By doing so it also allows atomic updates to the stream and block lists shared with the *TX queue handler* process, as well as atomic access to the global buffer area list for fast freeing of the transmitted packet data areas.

3.3.4.3 Network principles in action

The data transmission processes provide prioritised transmission of data at the highest possible transmission rate. In their design the following were taken into account:

- Low cost overload handling

  When the network destination for a stream is overloaded, the CFR TOG mechanism provides the *transmitter* process with appropriate feedback, which in turn allows the low cost overload handling that process implements (see principle 2.3.2).

- Discards excess data

Also, when the network destination overloads and provides that feedback, the structure of buffer lists per stream backpressures the *connection management* process so that data which cannot be handled because of the overload will be discarded in the *buffer from host* process; if the destination is sufficiently overloaded so that data times out then the *transmitter* process discards all the buffers associated with that stream. These discard mechanisms help support the overload handling principle 2.3.2.

- Low overhead per block and per stream

  Both the transmission processes do very little extra work per data block and per stream on top of the actual data handling. This is required by principle 2.3.6, to support high bandwidth streams.

- No data lost without a message

  With the CFR TOG mechanism and retransmission implemented in the *transmitter* process, all discarding of data in the transmission path is performed explicitly, and so all data lost can be reported to the host. This helps with the reliability of data transmission, principle 2.3.4.

- Latency proportional to size

  As the data in the transmit path has to be copied from memory over the OS-link to the slave transputer, the latency of this portion of the transmission is proportional to the size of the data. Principle 2.3.8 refers to low latency high priority streams, and this can therefore be acheived with low bandwidth streams.

- No data copying

  As no data copying is performed inside or to the transmission processes, as required by principle 2.3.8, there is also a benefit in higher possible data bandwidth, principle 2.3.6.

- Stream prioritising

  The transmission process maintain the structure in figure 3.6, which provides a priority-ordered list of streams for transmission. As the *transmitter* process transmits blocks from streams at the front of the queue in preference to those further down the queue, streams at the front have a shorter latency (principle 2.3.8) and higher priority (principle 2.3.7). As the stream priority is based on the stream number, which is host-specified, and as the stream priority forms part of the quality of service for the stream, principle 2.3.3 is supported (quality of service specifiable per stream) in priority terms.

- Active queue priority over inactive stream

  However, when a low priority stream is active and succeeding in transmission, the *transmitter* process does not preempt the stream when higher priority traffic arrives. This maintains the principle of active streams having a higher priority than inactive streams, described by principle 2.3.2.

- Streams multiplexed over OS-link

  The transfer of data to the slave transputer over the OS-link by the *transmitter* process multiplexes up to three streams. This helps to reduce data jitter (principle 2.3.9) and aids prioritising streams (principle 2.3.7), and at overload of one stream it helps stream independence (principle 2.3.2).

3.3.5 Data reception

Fast reception of data is important as although the CFR provides the TOG mechanism, which informs transmitters that the data they sent has not been correctly received (a very useful backpressuring mechanism), high bandwidth transfers requires this not to be used. So, the reception of data is managed in the networking software by a single high priority process.

3.3.5.1 Packet receiver

This process takes packets from the slave transputer, which buffers up to two packets itself. These packets must be reassembled into blocks, whose block descriptors must be passed to the *connection management* process.

The *packet receiver* process blocks waiting for a packet from the slave transputer. This packet is read directly into a buffer area which has been preallocated. The header information is examined to determine on which stream the packet was received. The header is then checked to see if it matches the header of the next packet expected on that stream. If so, the packet is linked into the block for that stream, and a new expected header is calculated. If the packet is not expected, either because the stream itself was not actively receiving, or because it is the start of a new block, or for some other reason, some exceptional code is run. In the first case a new block descriptor is allocated, the packet validated (it must be the start of a block), the packet is linked into the block descriptor, and the new expected header for the stream is calculated. In the second case the old block descriptor of the stream is replaced with a new one pointing just at the new packet. The packets linked into the old block are freed. The third case indicates the packet has arrived unexpectedly, which may occur due to the VCI map RAM failing, or because it is an MSNL PDU. An MSNL PDU is sent to the *connection* *manager*, whereas a completely unexpected packet forces the *connection manager* to check the CFR hardware via the *slave command manager.*

There may be cases where some packets from a block are received on a stream, then the transmitter stops for some reason (reset, hardware failure, or the connection being closed). To allow for this a timer is watched to ensure that any buffer areas allocated on a stream on which this occurs may be freed after a suitable time period. In this case the packet receiver process reports a *data timeout* message to the *connection management* process. This timeout is expected to be required very rarely, and is set at one second.

The *packet receiver* process runs at high priority to ensure the fastest possible response to packets being sent by the slave, and to provide fast access to allocation and deallocation of the global buffer area.

3.3.5.2 Network principles in action

The *packet receiver* process is driven by the slave transputer, which is in turn driven by the network itself, and it drives the *connection management* process with reassembled data. As such it is similar to the host interaction process, just a data pipe: it has no control over the order in which it receives data, so prioritising the data reception. Also it cannot recover from, or handle, the network itself being overloaded, as there is no available indication of this.

The network principles which were involved in the design, however, are:

- Low overhead per block and per stream

  The handling of packets, blocks and streams is highly optimised inside the *pakcet receiver* process, and results in a low cost in handling the blocks and streams (even at overload, principle 2.3.2 as well as a high receive bandwidth, principle 2.3.6.

- No data lost without a message

  Any data discarded in the network subsystem occurs either in the transmitter or at the *packet receiver* process, as the CFR uses its TOG mechanism to prevent receiver FIFO overflow. So, on the receiving end of a connection, all data discarded must be explicitly discarded, and messages indicating this data discard can be generated, helping in the stream reliability if required (principle 2.3.4).

- Latency proportional to size

  As with the previous OS-link transfers, the overhead of handling the data is mainly in the OS-link to memory transfer of the received data. As whole blocks must be reassembled before transfer to the host, the latency of a stream is proportional to the block size. To achieve low latency streams (see principle 2.3.8), small blocks should be used.

- Unlimited receive data buffering

  As required by principle 2.3.5, receive buffering is not explicitly limited, as the transfer of data to the host is faster than the network and therefore the receive buffering will balance automatically.

- No data copying

  The *packet receiver* process received data from the slave transputer over the OS-link into memory, and it is not copied before being transferred to the host by the *buffer to host* process. Not requiring a data copy reduces latency (principle 2.3.8) and improves bandwidth (principle 2.3.6).

- Streams multiplexed over network

  The *packet receiver* process reassembles many streams at once, thus allowing streams to be multiplexed over the network. Compared to restricting the reassembly processes this reduces jitter in streams, principle 2.3.9. At overload it also provides a uniform degradation of the network receiver.

3.3.6 Connection management

The sections above describe the low level processes in the networking software: host management; slave management; data transmission and reception. To provide a complete MSNL implementation requires a higher-level process. Also a switch is needed to transfer data between the host's processes and the slave's processes. This functionality is provided by the *connection management* process.

This process functions primarily as a switch, to transfer data from the host buffering to the transmission system and from the packet receiver to the host buffering. In addition it must decode commands from the host, converting them into MSNL actions, connection management actions, or commands for the *slave command manager* process. Most importantly it must accept data or commands from any of the processes which might want to communicate with it at all times — it is not allowed to send requests and block waiting for responses, or block on outputting data — otherwise the whole network software could grind to a halt. In this sense it is entirely event driven, the events being messages from its input processes. Note that in the whole structure this is the only point at which decisions are made based on one of many inputs, all other processes are pipes, and the process only operates on blocks, not packets. This is because the 'ALT' instruction required to wait for many inputs is slow to set up [29].

The following sections describe the implementation of the various functions of the *connection management* process, describing the events which can occur and the actions taken in response.

3.3.6.1 Protocol implementation

MSNL provides facilities for the creation and destruction of lightweight, bidirectional virtual circuits between two network interfaces connected by a variety of networks. In its simplest form the protocol implementation handles the following events:

- Host requests creation of a connection to another network interface

  When this event occurs the *connection management* process must check that the connection specified in the request is not already active — it is not permitted for a single connection to connect to two places. After passing this test the connection is initiated by setting up the internal connection data structures, creating an MSNL PDU for transmission, and passing this to the *TX queue handler*. This should ensure the MSNL PDU is broadcast on the local network. A timer is also set up so that another identical PDU may be sent, if no connection has been formed, after a suitable interval. This continues until a retry count is exceeded or the connection is formed.

- Connection creation reply received from packet receiver

  A connection creation reply may be received by the network in response to a creation request, in which case the *packet receiver* will forward the packet to the *connection management* process. This checks that the connection creation request was indeed sent, then establishes the connection in its internal data structures. The host is informed through a message to the *buffer to host* process. In addition the virtual circuit identifier (VCI), which was sent in the creation request, is opened by sending a command to the *slave command manager* to ask the CFR CMOS ASIC to set the correct VCI map RAM entry.

- Connection creation timed out

  If a connection creation request fails to elicit a response after a certain number of retries the creation of the connection is stopped, and the host informed via the host buffer.

- Connection create request received from packet receiver

  As an alternative to initiating a connection, another network interface may request a connection with the network software. This will mean receiving a connection create request PDU, which the *packet receiver* forwards to the *connection management* process. This process can then check its data structures to see if it should allow the connection to be made. If so, a connection create reply will be generated, a VCI chosen and the internal data structure updated. The host will be informed a connection has been made, and the *slave command manager* will be sent a command to ask the CFR CMOS ASIC to set the VCI map RAM entry.

- Host requests destruction of a connection

  When a stream has finished needing a connection the host will request the destruction of that connection. This forces a single connection destruction request PDU to be sent, via the *TX queue handler*, on the network; the internal data structures to be updated; a command to the *slave command manager* to ask the CFR CMOS ASIC chip to reset the VCI map RAM entry for the VCI allocated to the stream being destroyed.

- Connection destruction request received from packet receiver

  The final connection maintenance event is a request received from the network for an established connection to be destroyed. This request need only include the VCI belonging to the connection, and so when this PDU is received by the *connection management* process from the *packet receiver* it must search its data structures for a corresponding connection. Should one be found, the connection is removed by updating the internal data structures; informing the host the connection has been killed; closing the VCI by sending a command to the *slave command manager* to ask the CFR CMOS ASIC to reset the VCI map RAM entry.

3.3.6.2 Command interpretation

Command interpretation is in general simple. Some commands are handled by the protocol implementation. Others may request or set internal parameters, such as connection types or retry strategies. Yet others may not be understood by the *connection management* process, and these are forwarded to the *debug manager* in case it can handle them.

Commands may arrive from the host buffer processes, from the external OS-link reserved for debugging, or from the *debug command multiplexer*. This latter route allows the implementation of a network-based debugging system, as well as higher-level processes accessing the command system of the connection management process.

3.3.6.3 Data switching

Data switching is in general also simple. Data may arrive from the host buffer for transmission, in the form of a pointer to a block descriptor. This block descriptor will contain the stream on which the data should be sent. This maps directly onto an MSNL connection, so the connection structures must be checked to see if the connection has been fully created, and therefore whether the transmission should take place. If so, the block descriptor is updated with the destination VCI and block reassembly number, then it is forwarded to the *TX queue handler* process. If the transmission is not to take place the data must be discarded, so the local allocator process is asked to free the buffer areas, and th appropriate messages generated.

The other form of data switching is from the *packet receive* process, which may hand completed block descriptors to the *connection management* process. These are then sent to the host buffer, which will forward them to the host.

3.3.6.4 Message switching

So as not to complicate the above sections the message switching scheme described in section 3.3.1.2 has been omitted. Every message which is described as being sent to the host buffer in fact goes through the message switch. This allows routing of messages either to the *buffer to host* process, to the *debug buffer* process or to the *debug manager*, or to be discarded. The *debug buffer* and *buffer to host* are handshaking buffers (see section 3.3.1.3), and so for these destinations the connection management process maintains FIFO buffers of messages. As these FIFOs must be of finite length it is possible that they will fill up, with messages having to be discarded, When this occurs an appropriate message is forcibly inserted into the FIFO to indicate the overflow to the host or debugger.

3.3.6.5 Network principles in action

The connection management process provides the necessary features to supply the following Pandora network requirements:

- Data reliability

  To implement reliable data transfer (principle 2.3.4) for low bandwidth streams the message switching system can be utilised. Any data received on a stream can be forwarded to the debug manager, which can forward it to a reliability checker, which in turn could generate an acknowledgement and send on the data to the host by going back through the connection management process.

Similarly connections can be continually checked for integrity by having keep-alive packets generated by a higher-level process. These packets can be transmitted with a command to the connection management process.

- Receive priority over transmit

  The transputer has a construct, ‘PRI ALT’, which allows prioritised handling of message channels. In the *connection management* process the message channels from the *packet receiver* and *buffer to host* processes have top priority: this provides a form of receive streams prioritised over transmit streams, as required by principle 2.3.7.

- Inform host at overload

  All messages are switched inside the *connection management* process, and so it has full knowledge of the overload states. The process can the inform the host frequently, but not continuously, of the overload, as required by principle 2.3.2.

- Message overloading has no significant handling

  When the message system overloads, for example when the host itself overloads, the message switching in the *connection management* process inserts the message overload messages into the path to the host without much processing required. This helps keep the extra processing required at overload low, principle 2.3.2.

- Quality of service per stream

  The *connection management* process provides the host with full accessiblity to over 1000 streams, each with its own transmit buffer limit (T4 software only, see below) and other QOS parameters (principle 2.3.3). The priority is determined by the stream number, so with suitable choice of stream numbers the host can prioritise audio streams over video streams, for example (principle 2.3.7).

- Low per block overhead

  The handling of blocks for transmission and reception requires some simple decision making before switching the block handles to the appropriate processes. This is a small processing overhead, reducing latency (principle 2.3.8) and improving bandwidth (principle 2.3.6).

- SAP handling

  In order to implement remote debugging of the master transputer network software, or to support the remote Pandora interface access, the *connection management* process provides the facilities for handling its service access points in a wide variety of ways, allowing many clients to attach to the same SAP, or to only allow connection to a SAP if the host has explicitly requested access, as is the case for Pandora streams' connections (principle 2.3.1). The message switching provides the ability to route message to the debugger or to the host, required again by principle 2.3.1.

3.3.7 Debugging facilities

The debugging facilities provided by the initial version of the master transputer software were very limited, although the software was designed to allow easy expansion. The debugging subsystem consists of the *debug buffer*, the *debug manager*, the *debug command multiplexer*, and the debugging applications.

3.3.7.1 Debug buffer

The *debug buffer* process takes messages from the *connection management* process and forwards them to either the external OS-link reserved for debugging, if a local debugging system is present, or to the *debug command multiplexer* process if network-based debugging is in progress, or it discards them if no debugging is taking place.

For network-based debugging, messages forwarded to the *debug command multiplexer* are converted into commands to send the message contents on the debugging connection. Thus any debugging command which generates a message, or any event which creates a message routed to the debugger, will have that message sent as a block of data on the debugging connection.

3.3.7.2 Debug manager

The *debug manager* process supplies the expandability. It takes any command not known by the *connection management* process and passes it to the appropriate debugging application. The decision about which process to pass a particular command to is taken on the command type, and so each debugging process can support a number of command types, but no two processes can support the same one.

3.3.7.3 Debug command multiplexer

The *debug command multiplexer* provides the route by which the debugging processes can request commands to be performed. It takes input from all the debugging applications, and forwards them to the *connection management* process. An alternative would be to have all the inputs going instead directly to the *connection* *management* process, but this has the drawback that it would slow that process down (increasing the number of inputs to a process decreases its performance).

3.3.7.4 Debugging applications

The debugging applications provide the debugging functionality on top of the connection maintenance and slave transputer maintenance provided by the *connection management* process. For the initial software this includes processes for measuring the performance of connections, and utilisation of the buffer allocation system.

3.3.7.5 Network principles in action

The debugging system in the master transputer software was designed with two purposes in mind. Firstly, it allows the remote maintenance of a Pandora system's network interface, with access to the connection maintenance facilities and rudimentary connection statistics. Secondly it provides the abililty to measure the performance of the networking software under real loads, and to report that performance to a remote system, so avoiding using the Pandora system's response path and picking the messages out of a log file generated by the Pandora daemon.

In this way it fulfils only the requirement for a remote network-based debugging system. However, the expandability is there to provide reliable low bandwidth connections for the Pandora interface, network-based access (principles 2.3.1 and 2.3.4).

3.4 Additions and changes in later implementations

Since the initial version of the MSNL-based software was written there have been two new hardware versions of the network card (see section 3.1), each requiring new software. In addition, extra facilities have been added to the card in terms of debugging applications.

This section is split into three parts. Firstly the changes required by the shift to the T4 card (T425 master transputer with 2 megabytes of memory) are described. Then the remaining two parts describe the changes required to support the ORL ATM switch network, in both the slave transputer and the master transputer.

3.4.1 T4 Card

Upgrading to the 32-bit T425 driven card provided more freedom in the management of the buffering of data, with 2 megabytes of DRAM.

Instead of the buffer allocation system described above, all the software was rewritten to use 8 kilobyte buffer areas. So instead of reading in data from the host into packet-sized buffers and linking these together, the whole block of data from the host is read into a continguous piece of memory. This places the burden of segmentation on the *transmitter* process. Similarly, instead of linking packet-sized buffers together, the *packet receiver* process reads the data from the slave transputer into the correct position in a buffer allocated to a stream. These changes improve the performance of the software dramatically: the allocation and deallocation of buffer areas occurs on a block basis rather than a packet basis, and so requires less processing power; host interactions read and write from areas of memory, rather than breaking down the interactions into packet-sized operations. The system also removes the requirements for the high priority buffer allocation processes attached in the T2 card's software to the host interaction processes — this allocation can be performed by the *connection management* process.

It was at this stage that the buffer limiting requirement had to be implemented (principle 2.3.5). This is based in the *connection management* process, and restricts the number of buffers which may be waiting to be transmitted on a single stream. A stream can be receiving data from the host faster than it can be transmitted through the CFR interface, and so, without a buffering limit, a large amount of data may be buffered amounting to seconds of real-time data. So, when new data is received from the host for transmission over a connection on the CFR, the number of blocks already buffered on that connection for transmission is tested to see if it exceeds the limit set for that connection. If the limit is exceeded a message is generated indicating that data has been refused. This buffer limiting supports the Pandora network principle 2.3.5 on a per-stream basis, principle 2.3.3. (The T2 software did not require this functionality as there was not enough buffer memory available in the hardware.)

In addition to the changes to larger buffers, more debugging applications were added. These include a reset management application, which allows a Pandora box to boot over the network, so not requiring a host workstation at all, and more performance measuring applications.

3.4.2 ATM slave

As can be seen by comparing the two figures 3.2 and 3.3, the hardware that the slave transputer has to manage is entirely different for the ATM switch network than for than the CFR. This requires completely different slave transputer

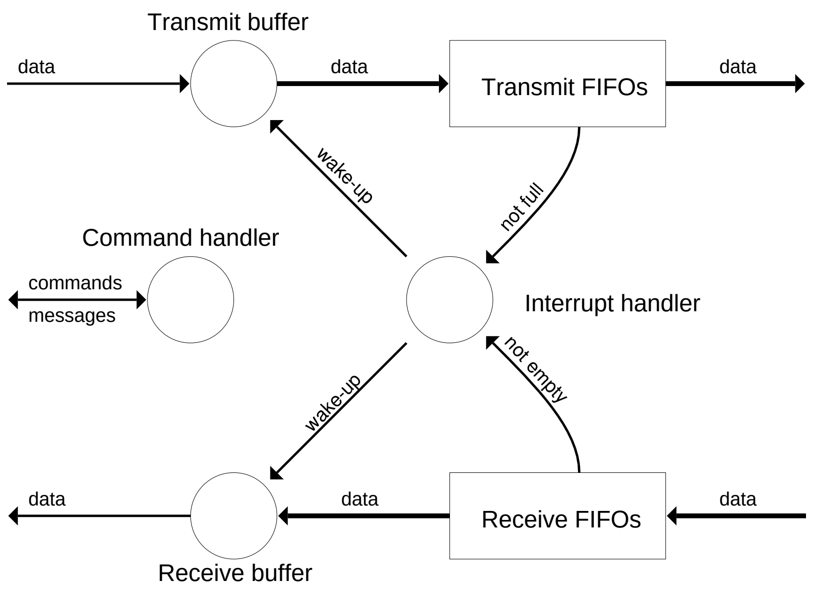


Figure 3.7: ATM slave transputer software structure

software.

The ATM slave transputer software, as for the CFR slave software, divides into four main processes (see figure 3.7): the *transmit buffer*; the *receive buffer*; the *interrupt handler*; the *command buffer*.

3.4.2.1 Transmit buffer

The ATM switch network interface uses FIFOs for cells for transmission and reception, each 64 ATM cells deep, unlike the CFR which provides only a single packet of buffering in each direction. This relieves the slave software of a major load: the complex management of the transmit FIFO. Instead, the *transmit buffer* must just read data from the master transputer into memory, then copy the data into the FIFO if there is room. If the transmit FIFO is full, the *transmit buffer* process marks itself as blocked, enables the TX FIFO not full interrupt, and waits for a message from the *interrupt handler*, much as the CFR slave does.

The copying of data from memory to the FIFOs is unfortunately necessary. It would be better if the data could be read directly from the master transputer into the FIFO. However, the OS-link engines in the transputer access memory using byte-wide accesses, whereas the FIFOs must be accessed 16-bits at a time.

3.4.2.2 Receive buffer

The *receive buffer* process normally runs constantly, reading cells from the received cell FIFO into memory and transmitting them to the host. The copying must be done for the same reason as outlined above.

If the receive FIFO should become empty the receive buffer process will mark itself as idle, enable the RX FIFO not empty interrupt, and wait for a message from the *interrupt handler.*

3.4.2.3 Interrupt handler

The *interrupt handler* waits for an interrupt to occur. When this happens it checks the source of the interrupt, which will be one of: TX FIFO not full; RX FIFO not empty; physical ATM connection broken. If the *transmit buffer* process is marked as blocked and the TX FIFO is not full it will be sent a wake-up message. If the *receive buffer* process is marked as idle and the RX FIFO is not empty then it will be sent a wake-up message. If the physical ATM connection has been broken the only action required is to reset the state of the ATM connection, to acknowledge the interrupt.

3.4.2.4 Command handler

The *command handler* accepts commands from the master transputer, obeys them, then generates a reply. Possible commands include reset, enable interface, and read or write the I^2C interface. A reset requires the FPGA to be reset by writing to a latch, holding the processor busy for some time, then releasing the FPGA. Unlike the CFR reset requirements, this takes only a few milliseconds. The other complex command, reading or writing on the I^2C bus uses the microsecond clock of the T222 transputer to time the setting, unsetting and reading of two latch bits, according to the I^2C specification [52]. Any data read is returned in a message to the master transputer, in addition to the standard acknowledgement reply.

3.4.2.5 Priorities and master/slave interaction

All four processes run at high priority, as they must run exclusively. However, none of the processes requires processor-intensive operations and so contention for the CPU does not occur. The relative priorities of receiving and transmitting data is not an issue, as without CPU contention the two processes occur independently of each other. Each will utilise the maximum bandwidth of the OS-links to the host that is possible given the hardware constraints.

3.4.2.6 Comparison with CFR slave

The ATM hardware requires much less management than the CFR interface. Indeed, the only reason for keeping the slave processor on the ATM network interface card was for the ease of implementing the software, by porting the CFR card software. The large FIFOs for transmission and reception place less time constraints on the slave software compared to the CFR. The ATM interface also requires little maintenance, unlike the CFR which has many failure modes. In addition there is no VCI map RAM in the ATM interface, as the network is formed from many single point-to-point connections rather than a multiple-access ring. Every cell received by the ATM interface must be destined for the ATM interface, whereas the CFR CMOS ASIC requires the VCI map RAM to determine whether a packet should be received.

3.4.3 ATM master

The ATM master transputer software is very similar to the T425 master transputer software, and actually uses the same debugging and host interaction processes. The *connection management* process differs only in the way it creates MSNL PDUs, which have a different format under FDL [6] on the 48-byte cell ATM switch network to MDL on the 32-byte packet CFR, and in the small amount of command decoding for managing the slave transputer (no VCI map RAM checking commands are required, for example).

The *packet receiver* process is different, as it must take the 48-byte cells from the ATM slave transputer for reassembly. In all other cases it too is the same.

The largest difference is in the *transmitter* process. This process no longer has to handle responses for each cell transmitted, and indeed has no idea whether a cell has been succesfully received by its destination. There is also no backpressuring mechanism provided by the network. The process simply sends the first buffer on every actively transmitting stream in order of priority, cell by cell to the slave transputer.

Some of the support for the network principles disappears as the ATM switch network does not have an analogue of the CFR TOG mechanism. Many of the reliability guarantees disappear, and overload in the network itself is much harder to detect. However, the ATM switch network also breaks the bandwidth limitations of the CFR, and so the transmit bandwidth to the network is approximately that from the host, leading to CPU overload being the major transmit failure, detected when the transmit buffer limiting comes into action; receive bandwidth from the network is similarly handled, but with the message system providing the buffer limiting. Overload from the network to the slave transputer, or cell loss inside the ATM network, can cause data to be discarded by the receiver, but usually some indication will be gathered when some cells from a block are received.

Finally, the *slave command manager* need not perform any of the complex tasks required by the CFR slave transputer, and only acts as a command sink, with the exception of reset commands which are forwarded to the slave transputer.

Chapter 4

High precision clock distribution

This chapter describes the background to the clock distribution implemented as part of the research of this thesis. It consists of a section describing the nature of the timestamping and synchronisation provided in the Pandora system, with its associated disadvantages; proposals for possible solutions to improve on the original Pandora system; an examination of the proposals; a look at the requirements necessary for precise, accurate clock distribution; a summary of the technology enabling the fulfillment of these requirements.

4.1 Pandora timestamping and synchronisation

The Pandora system provides capture and replay of many real-time video and audio streams. One of the major problems associated with handling this real-time data is the synchronisation of replay of related streams. The Pandora system uses an old approach to handling synchronisation — that is, it trusts synchronisation will just happen. The following sections describe in detail how this system works, and in what circumstances it fails to work.

4.1.1 Pandora timestamping

All the continuous media streams in Pandora are timestamped by the data sources with a 32-bit microsecond timestamp. On a standard Pandora box these timestamps are local to the data source which created the data, and as they are created by low priority transputer processes, their granularity is 64 microseconds. The only synchronisation point is at reset, when the Pandora system's transputers reset their clocks to 0. Indeed, the timestamps given to the data are times relative to the time of the last reset of their source and not relative to the time the stream was started.

The timestamps are not used by the Pandora boxes themselves at all. The data is timestamped for the sake of completeness. Instead, the Pandora systems rely on low jitter in streams and low inter-stream latency differences to provide synchronisation of real-time video and audio streams, and the timestamps are just ignored. This places the burden of synchronisation on the systems which must transport the data from source to sink, of which the hardest to manage is the network.

The network that Pandora was designed to use, the CFR, can cope quite well with the requirements of low jitter and low inter-stream latency with the network software implementation described in chapter 3. Indeed, Pandora boxes work very well together, and synchronization between live video and audio sources is well within user acceptable limits.

4.1.2 Synchronisation problems

Having said that Pandora boxes perform well when networked in synchronisation terms, there can be problems caused by the network itself, or by using a Pandora box to comunicate with a non-Pandora system:

- Videomail repository

  One of the problems which arose with running the Pandora system in conjunction with a videomail repository was synchronisation of the replay of the audio and video streams which make up a piece of videomail. Unlike a Pandora box, which replays data when it receives it, the repository must maintain a timetable for the replay of data it records. As the timestamps on the audio and video are generated by their respective sources, and no synchronisation of the sources inside a Pandora box is performed, the timestamps of contemporary audio and video segments need not be, even approximately, equal, so the timestamps themselves cannot be used as the full solution.

  The actual solution chosen by the author of the videomail repository software was to calculate the offset between the timestamps of the audio and video segments at the start of recording, and to assume that the drift in the clocks of the data sources can be neglected over the time span of a videomail message. The initial offset between the timestamps of the different steams being recorded is calculated by waiting until both streams are active and sending data, then to evaluate the difference between the timestamps of the next segments of data received on both streams. The replay of data is performed by trying to maintain the offset between the timestamps of the two streams at all times.

This mechanism works well with Pandora boxes connected over a single CFR network, where the startup of streams' transmission of data is immediate and not subject to initial variations. However, when Pandora boxes on different CFR networks, interconnected with the Cambridge Backbone Network (CBN) [20] as a backbone network, were used, it was found that the synchronisation of streams became erratic. This problem is due to the jitter introduced during the start-up phase of data transmission by the bridges between the networks. For a short time after a connection has been made between the Pandora box and the videomail repository on different networks, the intervening bridges do not handle the data sent on the connection consistently. This is either due to jitter introduced by the upper protocol layers managing the new connection, or perhaps due to the order in which a connection is initialised by the bridges. Whatever the case, the Pandora box sending the video and audio data would end up retrying the first few buffers for transmission. With the bridges not being ready to accept the data, the retries could take many milliseconds, forcing the audio and video buffers to fill. When one of these buffers fails in its retry attempts, all the data is discarded, and new data read in. If at this stage the bridge starts to behave normally, and both audio and video streams transmit successfully, there can be an additional offset between the timestamps of the audio and video segments transmitted. If, for example, the audio data is discarded due to its retry strategy failing, but the video data (being lower priority) gets through, the first video data to arrive at the repository is much older than the first audio data. By virtue of the mechanism chosen by the repository for calculating the inter-stream timestamp offsets, the value obtained is wrong by this extra age.

As a workaround, the repository now waits until both streams are active, waits another second, and then examines the two timestamps of the next video and audio segments to arrive.

- Bridges

  Two forms of bridge between the networks were used in the internetworking environment during this research. The first form was block-forwarding, the second cell-forwarding. Cell-forwarding bridges transmit cells on the destination network as soon as possible after they are received on the source network. These bridges have a small effect on the transit time for single cells, which is not noticeable in general in live or recorded streams.

  Block-forwarding bridges reassemble cells received on the source network into blocks, then segment the blocks and transmit the new cells on the destination network. This form of bridge is generally used between networks with different cell sizes (the ATM switch network has 48-byte cells, the CFR and CBN have 32-byte packets), or when different segmentation processes are required on the different networks (MDL/FDL or AAL5, for example). These bridges introduce extra delay in the transit time of a segment of data, proportional to the number of cells in the block. So video data has a much longer transit time than audio data. This is true of Pandora streams anyway, but the additional delay caused by the bridges in practice leads to disconcerting synchronisation lapses between live video and audio streams, which is also recorded by the videomail repository.

4.2 Improving synchronisation

The beauty of the Pandora system design is its simplicity, and a solution to the problems of synchronisation should attempt to retain the simplicity. Three possible solutions are presented here in the following sections: a timestamp database; synchronisation agents and managers; clock synchronisation.

4.2.1 A timestamp database

Synchronisation of a number of streams requires a mapping between the individual streams' timestamps. A possible solution is to generate a mapping from the local data sources to a fixed reference clock, and this can be performed by requiring the data sources to inform a timestamp database at regular intervals what its local time is. If all data sources report to the same timestamp database then timestamps from streams from a source can be mapped onto timestamps from streams from a different source.

This removes any problems involved in the synchronising of recorded streams, but does not help much with the playing of live data. An enhancement to the system would allow data sinks to request the timestamp relationships of the data sources in real-time, and so a mapping from their own clocks to the data source timestamps could be made, and the streams therefore synchronised.

4.2.2 Synchronisation agents and managers

The problems with synchronising streams only occur if different streams have different transit times from source to sink. If these transit times can be calculated, the difference in delays can be removed before the data is replayed or recorded, achieving synchronisation. This can be performed using synchronisation agents as in [59]. All the data which needs to be synchronised is sent from the sources to a local synchronisation agent. The jitter in the transit time between the source and the agent must be kept to a minimum, ideally zero. The synchronisation agent can then either combine the data into a single transmission, or record the timestamp offsets and transmit these and the data, to a destination synchronisation agent. This agent forwards the data to the data sinks with the appropriate delays between segments, using the knowledge of the timestamps given to it by the source synchronisation agent, this time with the jitter in distribution from the agent to the data sink being a minimum. A very similar device is the synchronisation manager ([55]), where data sources and data sinks provide stream information to the manager, which collates the information before providing feedback to the sources and sinks to maintain synchronisation. Others, e.g. [66] and [56], have investigated and implemented similar systems.

4.2.3 Clock synchronisation

A common mechanism used for timestamping data, for example file acceses, over a network filing system is by synchronising the clocks of the file servers ([37], [36]). This scheme can be extended to the timestamping of real-time data, for recording, live replay and replay of recordings.

The only requirement is that all data sources and data sinks synchronise their clocks, using a clock distribution mechanism. Data sinks can then replay data a fixed time offset from the data timestamps. Synchronisation occurs because the clocks of the data sources were synchronised, so the data is timestamped with the same global clock, and because the clocks of the data sinks are synchronised.

4.3 Examination of the possible solutions

The three proposed systems differ in their complexity of implementation, their impact on the data paths, and their effectiveness:

4.3.1 Timestamp database

The timestamp database requires a large database of all the data sources and data sinks to be kept, with a history of the timestamp offsets being kept so repositories can correctly map their recorded timestamps of different streams together. The data paths are unaffected, although connections from each data source and data sink to the timestamp database must be maintained. For reliable service, multiple copies of the database should be kept in case one fails, in which case the system becomes more complex and more connections are required, impacting on the data performance. The effectiveness of the timestamp database also depends on the jitter in sending the timestamp messages to the database from the sources, and from the database to the sinks. Since this jitter is the source of the synchronisation problem, the effectiveness of the solution must be in doubt.

4.3.2 Synchronisation agents and managers

The synchronisation agents are perhaps the most Pandora-like solution to the synchronisation problem, as they would be a single device managing external variations locally. Even so, they would be complicated at the replay end, as they must manage the scheduling of data transmissions to the sinks for replay purposes. In addition, all data which requires synchronisation must travel through the same synchronisation agents, and so the network bandwidth available to the synchronisation agents will have a large impact on the performance of the streams. Finally, provided the synchronisation agents are local to the data sources and sinks, and hence jitter from sources to agents and from agents to sinks is kept small, they should be effective in synchronising streams. Synchronisation managers are less effective in a multi-stream system where streams may be required to go to multiple destinations: combining feedback from all the sources and sinks grows in complexity as the sources and destinations increase in number. They also do not fit in well with the Pandora style of simplicity and local handling of streams.

4.3.3 Clock synchronisation

Distributing a clock of a suitable accuracy to each of the data sources and sinks may require individual connections from the sources and sinks to a clock distribution service. In this case the jitter in the communication of the clock from the distribution server to a client must be removed, or the effectiveness of the distribution will be reduced. Fortunately, when the local clock is synchronised to the global clock, the data paths are unaffected by the clock distribution method, and inter-stream synchronisation can occur fairly painlessly if the data sinks can schedule the replay of their incoming data according to the local clock.

Clock synchronisation has been examined many times before, with master-slave synchronisation (e.g. [51]), fully distributed synchronisation (e.g. [58], [57]), and emphasis on fault tolerance ([60]). For the experimental Pandora system on interconnected CFRs any fault tolerance is a luxury, but to support the precision of the Pandora timestamps (one microsecond) precise global clock distribution is required. This implies the clock source should have a precision and accuracy of one microsecond, and need not be elected, but can be fixed. Different CFRs can have seperate clock sources synchronised by some other mechanism to achieve microsecond accuracy. (For inter-stream synchronisation the full accuracy is not required, but it an be used effectively for performance evaluation of the system.)

The global clock distribution system can run hierarchically, with the fixed reference clock synchronising a few clients, which in turn act as servers for more clients, down to the data source and sinks.In the Pandora system an obvious clock synchronisation point is the network interface. This can then serve the data sources and sinks inside the Pandora box.

4.4 Requirements for precise distributed clocks

There are four requirements for accurately distributing a precise clock: a precise time source; a consistent local clock; an accurate distribution mechanism; a good logical clock model.

4.4.1 Precise time source

The precision of a distributed clock depends primarily on the precision of the time sources from which it is derived. Greater precision than that of the time source cannot be achieved, although if the distribution mechanism throws away some of the precision given by the time source, that precision is also lost to the distributed clock.

4.4.2 Consistent local clock

Once a precise clock value is accurately known by a system, that system must be able to maintain the accuracy of its knowledge of the distributed clock using its own local clock. If this local clock has a steady drift then this can be accounted for by the logical clock model. If the local clock is inconsistent, and drifts randomly over short intervals of time (less than the time intervals between distribution messages from the clock server, for example), this cannot be removed by the clock model, and so it will affect the accuracy of the local knowledge of the global clock.

4.4.3 Accurate distribution mechanism

In order to distribute the global clock to a client to a certain accuracy, the transit time of messages from the server to the client must either be known accurately, be accurately calculable, or be statistically such that the logical clock can remove any unknown in the transit time. If this constraint is not met, the logical clock model cannot remove the inaccuracy of the transit time of messages from the server to the client from the local knowledge of the global clock.

The transit time is defined to be the time between a time server or client requesting a transmission, and it arriving at the local-clock-reading process at its destination.

4.4.4 Logical clock model

The logical clock model has three inputs: a local clock; messages from the server indicating the global clock value; estimates of the transit time of the messages from the server to the client. With these the logical clock model must provide a representation of the global clock to the highest accuracy attainable given the accuracy of the local clock, the transit times from the server to the client, and using a model of the statistical variation of the transit times.

4.5 Technology available for precise distributed clocks

Until recently the availability of precise time sources and accurate distribution mechansims required by precise distributed clocks has been limited. With the advent of the Navstar Global Positioning System (GPS) [48]and high speed ATM networks these requirements may be more easily fulfilled. The following sections describe the way in which all four of the requirements can be fulfilled with the technology available to Pandora systems.

4.5.1 Navstar GPS — a precise time source

The Navstar GPS is a satellite-based radio-positioning, navigation and time distribution system. Navstar GPS receivers provide a high accuracy time anywhere in the world, in a small unit (10cm by 10cm, or smaller) for a few hundred British pounds, provided a suitable position for its small aerial is accessible. The installation at Olivetti Research Laboratories, Cambridge, has the aerial placed on the roof with the receiver inside the building.

4.5.1.1 Overview of Navstar GPS

The Navstar Global Posistioning System consists of three parts. The first is a set of satellites in approximately circular orbits. Originally twenty-one satellites were planned for the system, using three satellites in each of six orbital planes with three spare satellites. Presently there are twenty-five satellites, with four satellites in each of the orbital planes. Each satellite transmits on two radio frequencies (1575.42MHz and 1227.60MHz), modulating the signal with a ranging code. Superimposed on these signals is data relevant to the satellite, including clock information and its ephemeris. Each satellite contains an atomic clock (some use caesium, others rubidium), and receives data from the ground-based control system to provide a correct ephemeris.

The second part of the system is the ground-based control system. This consists of five monitor stations and ground antennas throughout the world. The monitor stations use standard satallite receiver units to deduce range information from the satellites, and this data is collated at the master control station (in Colorado Springs, USA). The master control station can estimate a new ephemeris and clock parameters for each satellite, and this is transmitted to the satellites with the ground antennas.

The final part of the system is the satellite receiver unit. This must receive data from more than one satellite, preferably simultaneously, in order to fix the position of the receiver's aerial.

From the information given by a single satellite a user can, if the user's position is known, evaluate the UTC time to a high accuracy. From the information given by four satellites a user can determine his position and the UTC time. The positional accuracy is 100m for 95% of measurements, but improves if the user takes an average of a large number of measurements.

Many Navstar GPS receiver units track more than four satellites at once, and choose the four strongest or best satellite signals. Some units have twelve channels to track simultaneously twelve satellites, although it is very rare for twelve to be visible. Other units use a smaller number of channels with some dedicated to particularly strong satellites, with the other channels tracking more than one satellite using time division multiplexing.

4.5.1.2 Precise timing calculation using Navstar GPS

A precise UTC-synchronised clock can be accurately derived from the signals transmitted by the Navstar GPS satellites. The method goes as follows:

The Navstar satellites transmit by modulating a radio frequency with a ranging code. They use two radio frequencies, one which uses P-code (Precise code), and the other uses C/A-code (Coarse/Acquisition code). At present P-code receiving units may be bought, but are expensive. Also AntiSpoofing may be implemented by the US Department of Defense in the future, which encrypts the P-code signals into Y-code, making current P-code receivers unusable.

The C/A-code is a 1.023 Mbps stream is a modulation of a code G(i,t), a 1023 bit Gold code, where i is the satellite vehicle number. G(i,t) is generated from two linear functions, G1(t) and G2(t). These are both generated using 10-stage shift registers. G1(t) is defined by the polynomial $1+x^3+x^{10}$; G2(t) is defined by taking the pattern generated by $1+x^2+x^3+x^6+x^8+x^9+x^{10}$; G(i,t) is then the exclusive-or of G1(t) and G2'(i,t). G2'(i,t) is generated by exclusive-oring two different taps of the G2(t) shift register, dependent on the satellite number. As G1 and G2 both repeat after 1023 cycles, so does G. The data to be transmitted modulates the G code at 50Hz. If a data 1 bit is to be transmitted then the inverse G code is transmitted for that 50th of a second, else the normal G code is transmitted.

The receiver can lock onto the sequence for a specific satellite vehicle by genearating the appropriate Gold code G(i,t), and multiplying the data in the received band by that signal. If there is a correlation (positive or negative) after a data-bit time then the receiver has locked onto that satellite. If there is no correlation the receiver must try again after slipping by 1 Gold code bit time. In this way it can take 1023 cycles of integration, each integration taking 1023 gold-bit times. The satellite receiver must do this for a few frequencies around the satellites transmitting frequency, as the received signal will have a Doppler shift, of up to 4 kHz.

The data transmitted by the satellite using the modulated Gold code contains:

- Satelllites' health

  Information on the health of all satellites.

- Antispoof flags for each satellite

  Antispoof flags indicate which satellites are using Y-code and not P-code, and so cannot be used for precise positioning by a commercial Navstar receiver unit.

- The satellite almanac

  This is reduced precison ephemerides for all the satellites. It is used to help the Navstar receiving unit search for visible satellites. The health of the satellites is used in the search to mask out satellites which are currently invalid or not in orbit.

- Accurate satellite ephemeris

  The accurate ephemeris for the satellite transmitting, given to the satellite by the master control station.

- Ionospheric data

  The ionospheric data provides information for the ionospheric model corrections in the receiver positioning algorithm.

- Week number and time of week in seconds

  The week number is the number of weeks since 6th January 1980. This is in a sense in units of GPS weeks, where each GPS week consists of $60 * 60 * 24 * 7$ seconds. The time of the week in seconds is then always between 0 and $60*60*24*7-1$. GPS time does not allow for leap seconds

— it increases steadily at one second per second. The offset between GPS time and UTC accounts for leap seconds in UTC.

- UTC data

  The UTC data provide information for converting GPS time to UTC time.

- A 22-ASCII character message.

  The 22-character ASCII messages are usually junk. However, examples which have been used and logged are: "AIR FORCE 2SOPS TEST " and "2 SOPS FALCON AFB TEST".

The time data distributed by a satellite only has the precision of a second. To generate accurate timing and positioning, the relative code phases of more than one satellite are calculated. The signal transmitted by a satellite is synchronised to its internal atomic clock, starting a gold code sequence precisely at the start of a second. The code phases at the receiver of signals from different satellites will be different, as the distances and hence delays between the receiver and the satellites will be different. The relative code phases then provide accurate information as to the relative distances of the satellites. With four satellites being received the relative code phases give enough information for the position of the receiver to be triangulated. Knowing the position of the receiver and a fairly accurate knowledge of the time gives a very precise estimate of the distances to the satellites, which can be used to improve the accuracy of the knowledge of the time to greater than the time taken by one Gold code bit, i.e. under a microsecond.

4.5.2 Consistent local clocks

A consistent local clock is a hardware counter which increments approximately at a known rate, where the rate does not vary over short time periods (i.e. a few seconds), which can be read by the microprocessor in a system. If the frequency is fairly constant, the logical clock model may calculate its value accurately. The counter must be quickly accessible by processes running on the microprocessor when distribution server messages arrive, so operating system support for the local clock may also be required. These two areas are discussed in more detail in the following sections:

4.5.2.1 Hardware support

Most microprocessors which may be used in networked systems do not contain any readable clocks at all. Those microprocessors which do not may have extra clock circuitry added in the surrounding system to provide timestamping and timeout management facilities.

In the case of the Pandora box, each card contains a transputer, which contains a microsecond clock linked to the processor clock signal, which in turn is derived from a quartz crystal oscillator IQEXO-3C. This provides a fairly constant rate if the temperature of the oscillator does not vary much, and even if it does the maximum variation is 100 parts per million ([32]). Due to the thermal mass of the oscillator its temperature will not change much over a few seconds, so the frequency fulfils the stability criterion.

The new multimedia system being designed at the Olivetti Research Laboratory in Cambridge uses boards based on the ARM microprocessor, which does not contain a clock. However, the boards do supply a UART which can provide millisecond counters, which may be used for a clock. This limits the precision of the local clock on these boards to a millisecond at best. Hence there is no point distributing a clock to an Atmos board to greater precision than a millisecond. If the boards were to be upgraded to include a microsecond clock, then the clock distribution to these boards should be to that level of precision and accuracy, if possible.

4.5.2.2 Operating system support

When a message arrives from the clock distribution server at the client the transit time of that message must be determined, and the local clock time of the arrival must be recorded. (The transit time should equal the time between the clock distribution server sending the message and the local clock being read.) Either the operating system must provide this functionality itself, or it must allow a suitable process to run sufficiently fast, to perform this operation accurately

The transputers in the Pandora box do not run an operating system: the transputers include hardware scheduling which is used to manage the running of the Pandora processes. The scheduling provides facilities for waking up processes when messages arrive. The process switching time is below a microsecond (if a high priority process is interrupting a low priority process, or if the processor was previously idle — high priority processes are not interruptable), and so it fulfils all the requirements described above, provided high priority processes are wisely used by the programmer.

Other operating systems, however, tend to have longer timescales for process switching, so interrupt handling would be the only path to take. Unfortunately, if this path is followed by too many clients of the operating system then the interrupt decoding mechanism adds undesirable delay prior to the reading of the local clock, hence decreasing the accuracy of the local knowledge of the global clock.

4.5.3 Accurate distribution mechanisms

As described above, the transit time of messages from a clock distribution server to a client needs to be calculated to an accuracy at least as high as the precision of the local clock, if that precision is to be fully utilised. This requires support from the physical network layer, either explicitly or by the nature of the network's design. The different support supplied by different networks is described in the sections below. The network types broadly split into two with respect to the accuracy of transit time calculation between those networks where the forward and return paths in a connection have the same bandwidths and distances, and those which do not. In the former (e.g. Ethernet), network transit times can be calculated by sending messages on both paths and averaging. In the latter (e.g. ring networks), the network transit time of data in the different directions may not be related, hence limiting the accuracy achievable, and requiring a more intelligent approach.

4.5.3.1 Ethernet

The transit times for packets on an Ethernet can be calculated accurately in three stages. The first is calculating the time between the transmission being requested and the transmitter succesfully starting the transmission. The data then takes a fixed time to arrive at the destination. The third stage is calculating the time between the packet arriving at the destination and the packet being presented to the operating system.

The first calculation can be performed by the network interface at the transmitter using its local clock, reading the value at the time of the request and the time of the transmission. The third calculation can be performed by the network interface at the receiver using its local clock. The second calculation can be performed beforehand, as it corresponds to the network transit time for a block of data of the size of the message from the soure to the destination. Due to the physical nature of the Ethernet, the network transit time is the same in both directions. So sending a message from source to destination and back again takes two network transit times, plus values deducible from the local clocks of the two network interfaces. Deducing and removing these values and dividing by two gives an accurate estimate of the network transit time for the message.

If an Ethernet is split into multiple segments, with bridges in between, the symmetrical transmission nature may not be retained, so reducing the accuracy of distribution. This accuracy can be recovered if the bridge is intelligent with regard to the time messages — it can inform both client and server of the delay in a particular message due to the bridge, if it too contains a local clock.

Most Ethernet interfaces do not provide the accurate time information required for the first or third calculation stages, hence limiting the accuracy of the whole calculation, and the accuracy of the distributed clock. The result should still be accurate to under a millisecond, which is more than good enough for Pandora. Unfortunately the other properties of the Ethernet (bandwidth and media access) do not support the multimedia requirements of Pandora, and so an Ethernet network card for Pandora will never exist.

4.5.3.2 ATM ring networks (CFR and CBN)

For highest accuracy, any time messages whose transit times need to be calculated should be a single 32-byte packet long. The transit time calculation for the ring networks can then also be split into three stages: time to transmission of the packet; time of transit of the packet on the ring; time between packet reception and it being presented to the operating system. The CMOS CFR chip provides single packet FIFOs for transmission and reception, and signals to its controller when the FIFOs become empty or full respectively. If the local clock is read when these events occur, the first and third stages of the calculation can be made accurate. The CBN uses much deeper FIFOs (2048 packets), and so more care must be taken. One possibility is to attempt to ensure that all other traffic is idle; assume insertion time into the FIFO equals the start of network transit; read the time when the receive FIFO becomes non-empty. This solution introduces some inaccuracy, but it can be guaranteed to be within certain bounds.

In both cases, the second stage, the ring transit time, is harder to calculate. This is due to the different paths taken by traffic going in opposite directions beween the two stations. If a packet is sent from a source to destination, then from destination to source, the packet will have spent one ring revolution of time on the ring. If the transit time is taken to be half of this, the accuracy of the calculation is one half of a ring revolution time. This accuracy may be improved if the ordering of stations on a ring can be deduced, as the ring revolution time is effected by the buffering of ring data inside the stations. Unfortunately the ordering of stations on a ring cannot be deduced just by using the ring itself.

On any CFR the ring revolution time is under 100 microseconds, and so the accuracy of a distributed clock available to CFR-based Pandora boxes should be better than 50 microseconds.

4.5.3.3 ATM packet switch networks

As for the ATM ring networks, it is best to use single cells to help accurate transit time calculation of transmission of a message in an ATM switch network. As with the CBN, FIFOs for transmission and reception may be many cells deep, so a similar solution may be adopted.

However, the second stage of the calculation, that of network transit time, is not as easily performed with a switch fabric as either of the network types described above. Firstly, the forward and return paths from server to client may not be the same, so using the echoing method described for Ethernet may not be applicable. Even for networks for which both paths are the same length, the network transit time for packets sent at different moments will be effected by the switching times for those packets, which in turn is effected by the loading of the switches at those moments. Hence the network transit time for two packets along the same path may be different, and not calculable by the receiver or transmitter.

The switching elements could provide help, timestamping with a local clock on both reception and transmission of cells that they switch. This may lower the overall performance of the switch, and so is undesirable. The only alternative is to apply some statistical knowledge to the handling of network transit times.

Depending on the statistical nature of the transit times of cells over the ORL ATM switch network under normal networking loadings, Pandora boxes connected to such a network may be able to have distributed clocks more accurate than those connected to the CFR.

4.5.4 Logical clock models

Logical clock models are the basis of the software in any distributed clock system. Only a small amount of research was performed into logical clock models, as highly sophisticated models were deemed unnecessary if the transit time of clock distribution messages could be accurately determined, and therefore outside the scope of the research.

In this research logical clock models are considered to consist of four parts:

- A clock message filter

  In an ideal system the transit time of messages from the server to the client in a distributed clock system would be determinable to an accuracy equal to the precision of the distributed clock. However, in real systems there can be errors in the transit time calculations, or in reading the local clock values corresponding to the arrival of the messages from the server. In these cases a large error could affect the stability of the logical clock model. So messages with unexpectedly large transit times or unlikely local clock arrival values may be filtered out. This filter depends on the message distribution system, and in some logical clock models may never need to filter messages as the clock correction unit may perform weighting or filtering of the messages according to their distribution times.

- An offset detector

The offset detector takes the incoming clock messages and compares the global time derived from that message using the transit time, local clock arrival values, and the local-to-global clock conversion parameters of the model, with the global time of its transmission given in the message. The offset between the locally believed transmission time and that given in the message is fed into the clock correction unit.

- Clock correction unit

  The clock correction unit adjusts the local-to-global clock conversion parameters for the model according to offsets it receives from the offset detector. It may also make use of the message distribution time, to weight the correction according to the believed accuracy of the message.

- Clock reading

  The logical clock model must also provide a mechanism for reading its derived global clock. This will involve using the local clock value and the clock conversion parameters. In returning the global time it should also be able to describe the precision and accuracy of the time.

In the course of the research a few clock models were experimented with, and they are described below. For an examination of the performance of the clock models, see section 5.3.

4.5.4.1 Initial experiments

Initial experiments used a simple clock model. A difference in microseconds between the local clock value and the believed global clock was stored in a shared 32-bit word. This was updated according to frequency and phase error values derived from the clock correction unit.

No clock message filter was used; the clock correction unit used a long-term average to calculate the frequency error of the local clock and the phase error was set to the offset detected between the clocks. On the receipt of a message, the clock correction unit calculated the number of microseconds by which the difference value would have to be adjusted in a single second, and from this deduced the interval between single microsecond adjustments for the difference value.

4.5.4.2 NTP Fuzzball logical clock

The above clock is not very stable, and can be easily improved on. The Fuzzball [45] logical clock, as used in the Network Time Protocol ([46], [47]), was investigated next. This maintains a frequency error, phase error, compliance, and difference between the local clock and the believed global clock. The compliance is a measure of how close the derived clock is to the global clock, and is determined by averaging the offsets received from the offset detector over a period of time. The frequency error is adjusted using the compliance as a scaling factor — if the clocks are not synchronised (absolute value of compliance large), the frequency error will be adjusted more than if the clocks are believed to be close to synchronisation (compliance near zero). The phase error at the time a message is received is set to the offset detected.

At regular intervals the difference between the local clock and the believed global clock is adjusted according to the frequency error and the phase error. At each of these stages the phase error is reduced, so that it asympotitically recovers the last offset detected.

4.5.4.3 Enhanced fuzzball logical clock

To improve the capture range and frequency accuracy of the Fuzzball logical clock (limited because of the 32-bit precision used in the implementation) it was enhanced. The frequency error was adjusted to use as many of the 32-bits as it could, using a divisor like an exponent to provide a pseudo-floating point number. The real frequency error is the frequency error divided by the divisor. The divisor is decreased only if the frequency error would overflow its 32-bit word. It is doubled, and the frequency error halved, only when the compliance remains small for a period of tens of seconds, and then only if the frequency error will not overflow. The clock difference and phase error are stored to a precision of 1024ths of a microsecond. In some internal calculations (for the frequency error adjustments) 64-bit integer arithmetic is used.

To ensure a monotonically increasing clock the clock difference adjustment processes of the above two clock models were combined, so that at suitable intervals the clock difference would be adjusted by one microsecond (as in the initial implementation), and at regular intervals a new adjustment would be calculated and the phase error reduced (as in the Fuzzball logical clock).

Another enhancement was added to improve the start-up time of the model. This uses the global and local time differences between the first two messages received from the server to calculate a rough value for the frequency error of the local clock.

Chapter 5

Experimentation for global clocks

This chapter describes the work performed to evaluate network, processor and logical clock performance for time distribution, using both the Cambridge Fast Ring networks and the ORL ATM switch network. The research described provided support for the belief that the accurate distribution of a precise clock is possible with the tecohnlogy available in a Pandora system, and an idea as to how a complete implementation could perform.

The chapter divides into three sections, each describing the work carried out, the results gathered, and the analysis of the results, for: calculation of message transit times over the ORL ATM switch network; calculation of message transit times over the CFR; possible logical clock models. (The ATM switch network is described before the CFR as gathering information from the CFR had to be performed using the ATM switch network, with derivatives of the ATM software.)

5.1 ATM switch network transit times

This section describes the work performed to analyse the echoing of cells between two Pandora network cards using the ORL ATM switch network. Performance problems related to the capabilities of the network cards in this scenario are very similar to those that would be present in a distributed clock system for Pandora, and so any artifacts introduced by the nature of the hardware do not degrade the validity of the results. This is not an attempt to measure the real performance of the ATM switch network, only the echo behaviour of two Pandora ATM network cards in conjunction with the network.

The experimental system divides into three parts: a data source, an echo server and a results gathering node. The data source and echo server are both Pandora ATM network cards with specially written software. The results gathering node is a DECstation 5000/240 with a YES (TurboChannel ATM network interface) board and the MSNL protocol implemented in its kernel for accessing the ATM network.

As described in section 4.5.3.3 the most accurate transit time calculations will probably take place with messages consisting of single cells. Therefore the echo server described in this chapter echoes cells as they are received, without reassembling them into blocks.

5.1.1 The echo timing system

The echo timing system uses the MSNL protocol to set up virtual circuits describing a route from a data source to an echo server, back to the data source, and from there to a results gathering node. This is performed with three MSNL connections, one for each hop. As MSNL connections are bidirectional, the same results could have been achieved with two connections, but using the extra connection leads to a simple design for the software for the Pandora network cards. Indeed, the data source and echo server can use identical code, and this is described in the following section. After that is a description of the software required for the results gathering system.

5.1.1.1 Data source and echo server software

The task of the data source is to transmit cells to the network on a virtual circuit, receive echoed cells on another virtual circuit, and retransmit these to the results gathering node on another virtual circuit. The task of the echo server is to receive cells from the network on its echo virtual circuit and retransmit them on another virtual circuit. To manage the virtual circuits both must also handle PDU cells received on the MSNL promiscuous port for connection management. These may be connection make requests, replies or termination requests — any further MSNL PDU's are ignored.

The echo server maintains two publicly accessible MSNL SAPs, to each of which a single MSNL connection may be made, used for the data virtual circuits. Also another MSNL SAP is supported, through which the echo server can initiate connections.

The data source expands on this by having a third MSNL SAP used for a virtual circuit for the data it generates.

As a cell is forwarded by the data source or echo server it is timestamped with the value of the local clock. When a cell arrives at the results gathering node it contains the following local clock timestamps: the originating timestamp from the data source; the retransmission timestamp of the echo server; the retransmission timestamp of the data source. Using the first and last timestamps, the echo delays may be calculated in terms of the local clock ticks of the data source. Without loss of accuracy these can be taken to be in microseconds (the clocks on the transputers, using the oscillators on the Pandora 1a boards, drift by up to 250 ppm, which is one microsecond per 4 milliseconds, much longer than an echo time).

The MSNL SAPs, supplied by the data source and echo server are accessed using the following port numbers:

- Port 0.0.0.2

  Connecting to port 0.0.0.2 sets up the virtual circuit for the incoming data stream. The data will be retransmitted on the virtual circuit created using port 0.0.0.4 below, if that has been set up.

- Port 0.0.0.3

  Port 0.0.0.3 is the SAP for extra connection management and packet sourcing. The two command types are differentiated using the first word of data in the cells received on the virtual circuit formed using this port.

  Connection management cells received here are retransmitted as MSNL connection make request cells for the card's port 0.0.0.4. For example, a controlling program can connect to port 0.0.0.3 of the echo server and send an appropriate cell on the virtual circuit causing a request from the echo server to form a connection between its port 0.0.0.4 and another MSNL node.

  Packet sourcing cells contain a word specifying the number of ATM cells in the block to be sourced. The block is built in the data FIFOs of the network card, each cell containing the local microsecond clock time of its generation. The cells are then made available to the ATM hardware by inserting the correct number of words into the cell tag FIFO (one word per cell). In this manner the data transmission rate is limited by the ATM network hardware itself, and not the rate at which cells can be inserted into the FIFOs by the Pandora network card.

- Port 0.0.0.4

  Port 0.0.0.4 is controlled by cells sent on the MSNL connection made to port 0.0.0.3. This is the outgoing data half of the echo circuit, as when an MSNL connection reply PDU is received for this port the echo server records the VCI for the transmission side of the connection and uses this as the outgoing VCI for echoed cells.

- Port 0.0.0.5

Connecting to port 0.0.0.5 sets up the connection to the results gathering system from the data source, the second outgoing connection for the data source.

The simple management of connections leads to small code size. It also makes the management of individual ATM cells very efficient — once received, they are quickly sorted according to received VCI. Cells to be forwarded (hence echoed) are then timestamped, given new VCI's, and inserted into the transmission queues. It must be noted that even though this code has been highly optimised it still takes 13 to 14 microseconds to receive, change, and retransmit an echoed cell. Of this time, 3.8 microseconds is taken in reading the cell into memory, 3.8 microseconds writing it out again (these are memory speed limited), with the rest spent managing the cell.

5.1.1.2 Results gathering system

The results gathering system is a DecStation 5000/240 fitted with a YES board (a TurboChannel ATM card), running OSF/1, with the specific software written in C. It uses MSNL to form connections with the data source and echo server to create the data path, then it commands the data source to generate data. It uses the Unix "select" call to wait for 20 milliseconds between receiving a reassembled block of cells from the data source and sending out the next data generation command. It can determine if blocks are not received even though a prod was sent. It will repeat prodding the data source until it has received a specified number of blocks. The echo delays, calculated from the differences in data source timestamps, of each cell in each block are recorded in a textual file as the data is received.

To generate the connections for the data source and echo server, the result gathering system runs as follows:

1. Connect data source to echo server, the outward path

   First it forms a control connection to port 0.0.0.3 of the data source, sending a cell to request a connection on the data source's port 0.0.0.4 to the echo server's port 0.0.0.2. It then terminates this control connection.

2. Connect echo server to data source, the echo path

   Then it connects to port 0.0.0.3 of the echo server, sending a cell to request a connection on the echo server's port 0.0.0.4 to the data source's port 0.0.0.2. It then terminates this control connection.

3. Connect data source to results gathering software, the results path

The results gatherer then forms a connection to port 0.0.0.5 of the data source, setting up the final leg for the data.

4. Connect results gatherer to data source, the data control path

   In order to control the sourcing of data the results gatherer connects to port 0.0.0.3 of the data source again. This time it uses the connection to send 'source packet' commands at regular intervals.

Once the network connections have been made the results gatherer repeats sending the 'source packet' commands to initiate cells to the data source. At most 64 cells per block may be generated per command, and as there are 52 data bytes per cell and 50 data blocks generated per second, data rates of up to 1330 kilobits per second may be produced, although the transit times of single cell blocks are the main area of interest.

The design of the software for the data source and echo server is such that multiple hops can be used in the echoing of cells, i.e. the data source sends cells to A, which forwards them to B, which passes them to C, before finally being sent back to the data source and from there to the results gathering system. This feature provides no extra useful information unless the network is highly loaded at some points. Then the data path could be made to go through a highly loaded point more than once to see what effect the high load has on the data. However, as can be seen from the results, the loads in the system were never high enough to generate any real effect on the data, so results using multiple hops are not given here.

5.1.1.3 Artifacts of the system

The software running on the Pandora network cards and the ATM switches connecting them interact in a way that may not be immediately obvious. This section describes the rates at which cells are sourced, carried, switched and echoed, and the artifacts that these different rates might produce.

- Cell sourcing rate

  The cells of a block are generated by the data source and forced into the network interface faster than they can be transmitted by the network, because the insertion of a cell is performed by inserting a single word into a FIFO with the data having already been loaded into the seperate data FIFO. Inserting the single word takes about 200 nanoseconds, and data transmission occurs at the rate of one cell per 4 microseconds.

- Cell transport rate

The cell data is serialised and transmitted along coaxial cable at a data rate of 100 Mbps. For a 53-byte cell (with two bytes of overhead) this corresponds to 4.4 microseconds taken between ATM ports. With additional FIFO fall-through delays, the cell transport time for the first cell in a block is close to 6 microseconds (see oscilloscope results below, section 5.1.2.3).

- Cell switching rate

  The switching speed for cells is described in more detail below (section 5.1.2.7). It should be noted for now that the switch uses FIFO-to-FIFO copies to transfer cells between ports, at about 4 microseconds per cell.

- Cell echoing rate

  The echoing node must receive all the data of a cell into memory before retransmitting the cell. Reading and writing the data for a cell in a Pandora ATM network card takes almost 4 microseconds each, making the absolute minimum echoing time 8 microseconds (slower than the switch and cell transport).

  The echo server and data source must handle cells arriving on different VCI's in different ways — the cells may be PDU cells, cells on a command channel, or data cells. So some time is taken by the software deciding what to do with every cell based on its VCI.

  Also, there are other interrupt sources which must be managed by the software on a Pandora ATM network card, relating to the maintenance of the network hardware. Even when the received cell interrupts and the echo VCI are given the highest priority in the software, the minimum total delay in echoing packets 13 microseconds (again, see section 5.1.2.3 below).

The net effect of the above rates is that the cell echoing is the bottleneck. This implies that data is shipped from data source to echo server at the highest rate the intervening network can manage (the cell transport rate, one cell per 4.4 microseconds). The cells will then tend to be buffered at the echo server waiting to be handled, before being sent back at the rate of one per 13 microseconds. So the expected interarrival time of cells at the data source is 13 microseconds, unless the cell switches in between fail to respond as predicted.

Now, as the cells are transmitted at full network speed, and echoed cells travel along the reverse route through the same switches as transmitted cells, there should be an interaction between the high cell rate and the returning cells. This effect should only appear on suitably large blocks of cells.

5.1.2 Results

The results were generated on the ORL ATM switch network distributed throughout the Olivetti Research Laboratory building in Cambridge. This section describes all the results collected when timing the ATM switch network echoing packets. The routes and loadings of the network are described first, then some oscilloscope measurements taken during the testing, then the results gathered using the software described above are presented.

5.1.2.1 Network routes used

Three different network routes were used to collect results. The first, route A (figure 5.2), has the smallest possible network between the data source and echo server, just a single switch. This route gives a lower bound on the time for echoing any blocks in any system, and also allows simple analysis of a switch. The second route, route B (figure 5.3), is of medium length, simulating the longest route in the ORL ATM network which does not require the data to pass through the central hub. Only 4-port switches are involved, and so interactions between these may be examined. The third route, route C (figure 5.4), is the longest route available on the ORL network, going from one end of the building, through the central hub, then through the next longest spur. This route provides some idea of the behaviour of the echoing of data over long routes, and provides the largest scope for interference from other data streams passing through the switches used in the route.

5.1.2.2 Network loadings used

The results were gathered outside office hours to allow the examination of an almost totally quiet network, as well as under a relatively high loading (certainly much higher than is currently used in office hours). Unfortunately this loading is very much less than the network is capable of, as the high data-rate producers and consumers for the network are not yet available.

With a quiet network the only traffic is very low bandwidth network management cells. The relatively high loadings of the network were designed to simulate, as well as possible, real-time multimedia traffic. With the state of the ORL multimedia systems at the time the results were gathered this means video streams consisting of 16 bit colour video from cameras at a frame size of 88 by 64 and rate of 25 frames per second to DECstation's equipped with YES boards, and stereo audio streams using two 16 bit samples at 32 kHz between networked audio boxes. These correspond to data rates of 2.25Mbps for the video (bursts of data every 40 milliseconds) and 1 Mbps for the audio (continuous data). Video streams were set up from *pheasant* and *dabchick* to *truffle*, and from *pheasant*

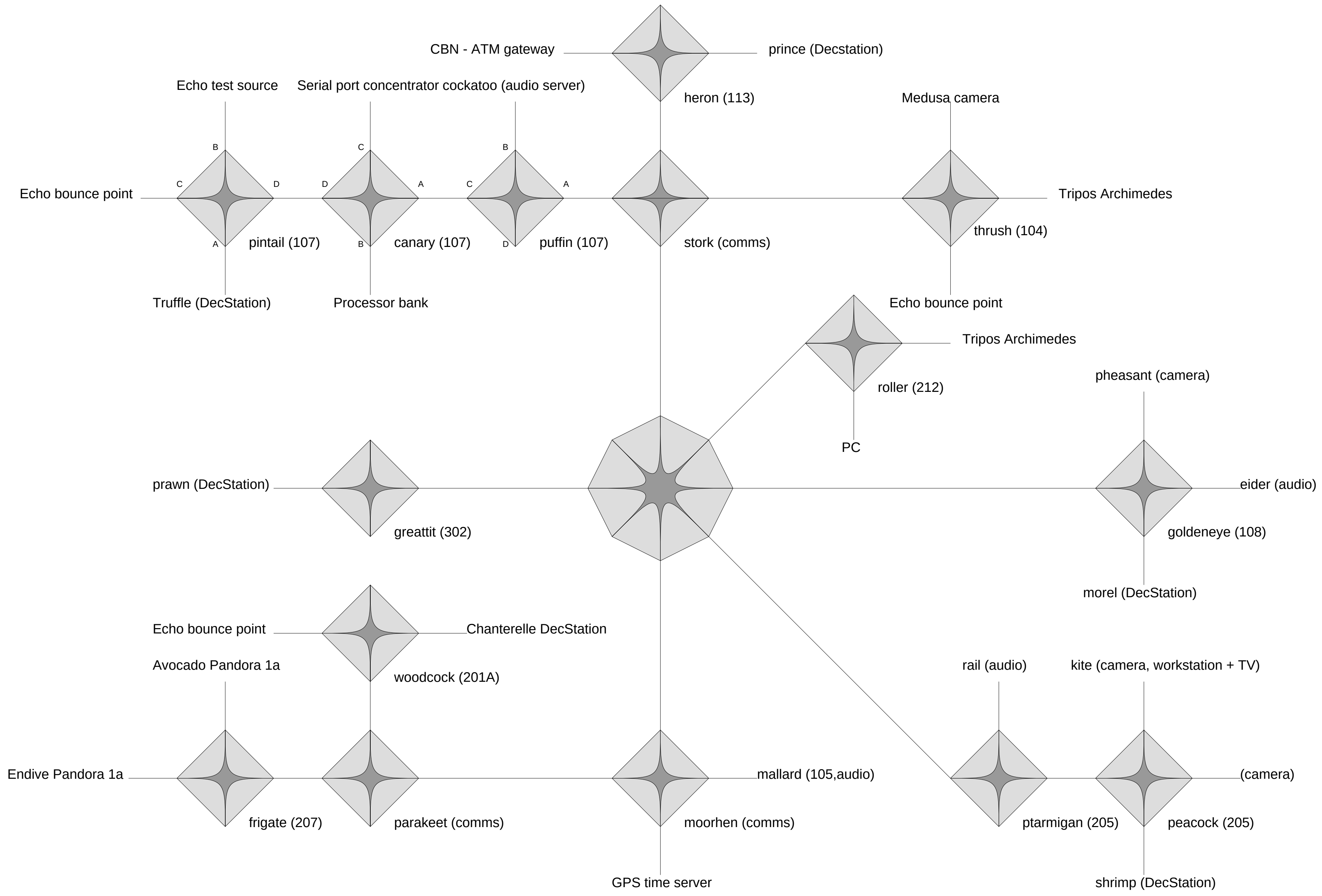


Figure 5.1: ATM Switch Network Topology for Experiments

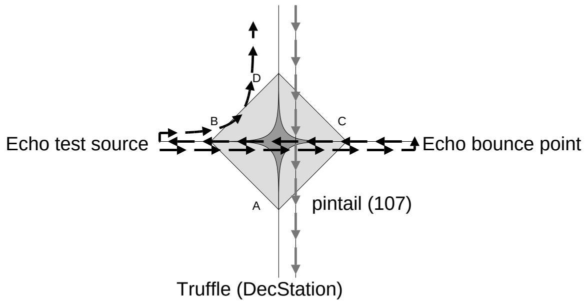


Figure 5.2: Route A

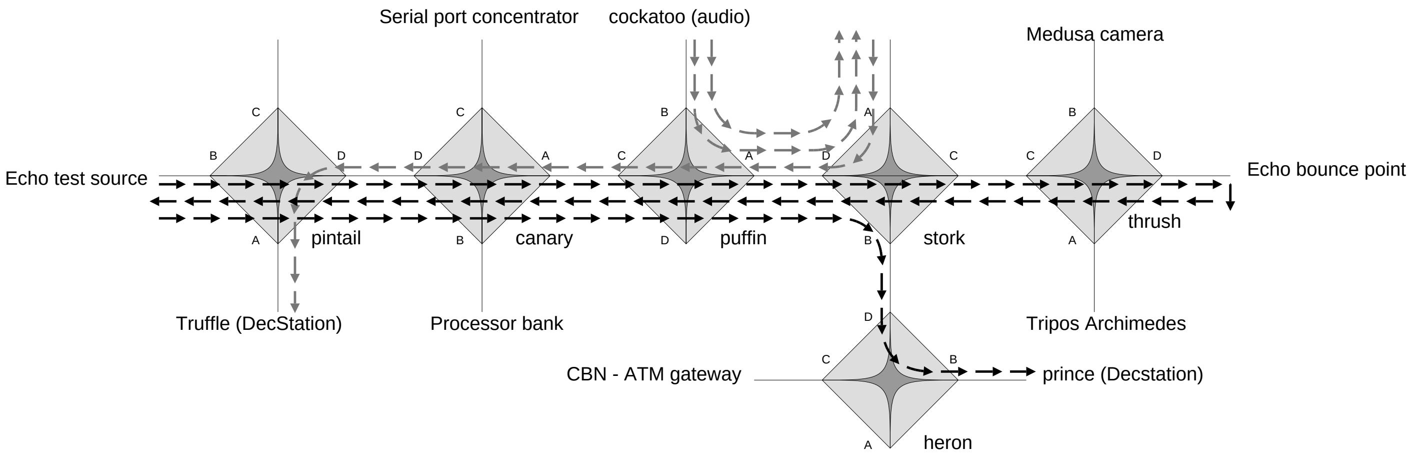


Figure 5.3: Route B

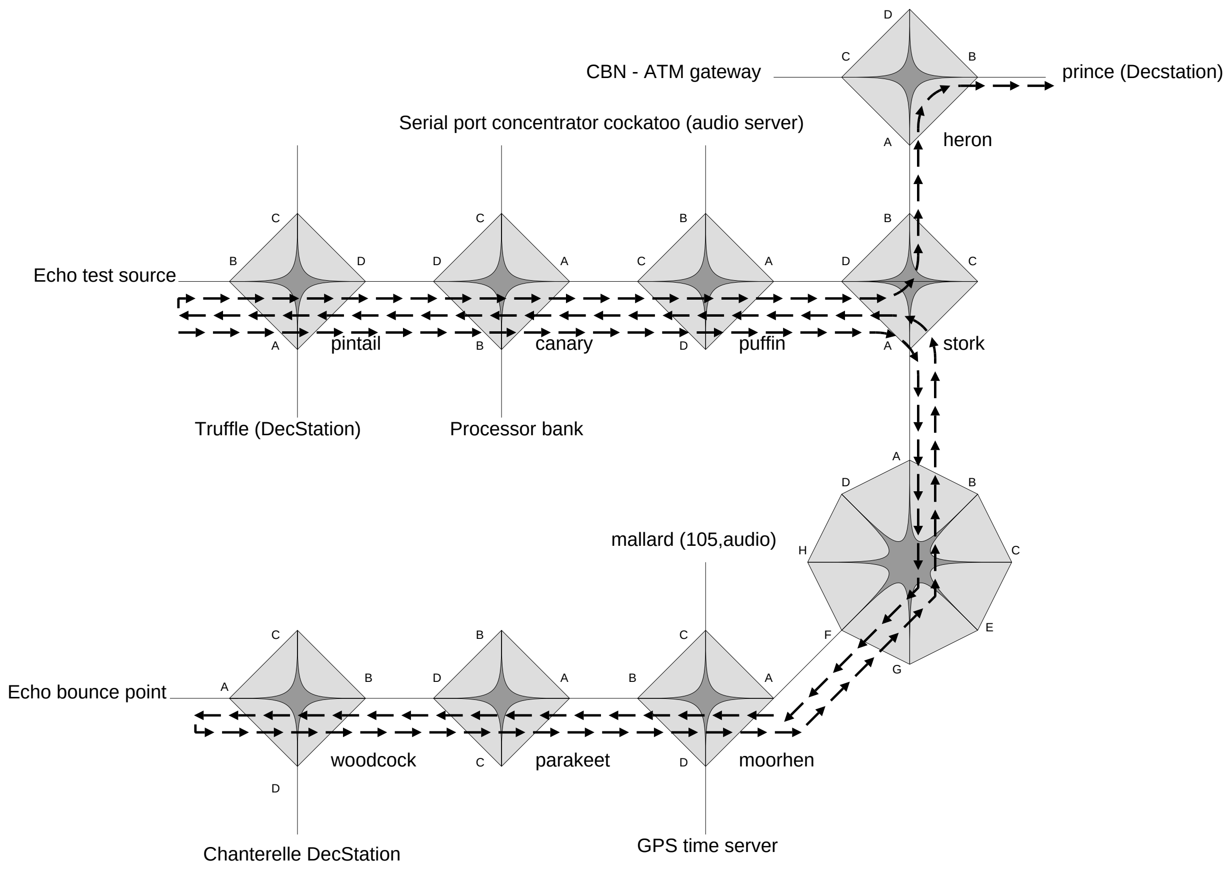


Figure 5.4: Route C

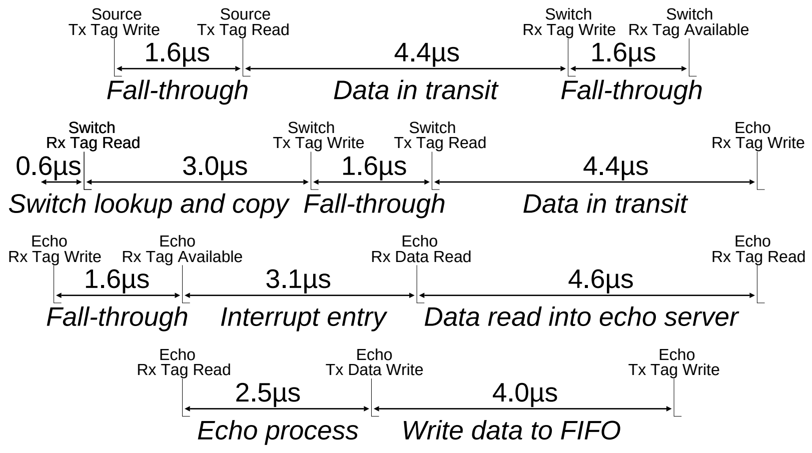


Figure 5.5: Time diagram for a single ATM cell

and *dabchick* to *chanterelle*. Audio streams were set up bidirectionally between *grouse* and *mallard*, and *grouse* and *rail*. These streams were chosen mainly because, of the possible routes, the paths of the streams overlapped to the largest extent available with the chosen routes without the experimental audio and video stream software crashing.

5.1.2.3 Oscilloscope measurements

In addition to running the echo timing system in software, some measurements were taken using a digitizing storage oscilloscope to investigate the performance of the Pandora ATM network cards and the 4 port switches.

The oscilloscope was set to trigger on the data source writing the first word of data into the transmit data FIFO. The data source was running single cell blocks through a single 4 port switch. The timings read from the oscilloscope are not entirely reliable as they were read from different single occurences of cells. However, the timings are fairly repeatable.

Once the data has been written into the data FIFO the tag word is written. This is the first point on the time diagram in figure 5.5. The tag FIFO has a fall-through time of approximately 1.6 microseconds [53], and after which the cell is transmitted onto the ATM cable. The cell is received into the data FIFOs on the switch, and when the transfer is complete the receive tag FIFO on the switch is written by the ATM interface. After another fall-through time the cell becomes available to the switch, after which the switch detects the cell, reads the tag FIFO, transfers the cell data to another port, then writes the transmit tag FIFO for the new port.

As before, the tag has to fall through the FIFO, after which the cell travels to the echo server's FIFO, causing its receive tag FIFO to be written when all the data has been transferred. The echo server enters its interrupt routine, reads the data from the data FIFO and reads the tag FIFO. After a decision process the cell is written to its transmit FIFO, followed by the tag, and the cell has restarted its journey in the reverse direction.

As can be seen from figure 5.5 this cycle takes approximately 33 microseonds. Important features are the switching time (approximately 3.6 microseconds), cell transit time between any two ATM interfaces (4.2 microseconds, or 7.6 microseconds if the FIFOs are both empty), and the echo server echo time (14.2 microseconds).

5.1.2.4 Results for echoing single cell blocks

The simplest test system usable with the echo server consists of the data source, echo server and 4 port switch, as shown in figure 5.2. The results of timing the echoing of blocks containing a single cell are shown in figure 5.6. The first graph contains the full results for the test system without any load. It exhibits a single spike distributed around 67 microseconds. The slight variation in the time between 66 and 68 microseconds is due both to the variation in the timing of the data source reading its local microsecond clock, and the operation of the 4 port switch, which uses a polling loop of approximately one microsecond duration and hence introduces up to one microsecond of delay in the path of a cell. An examination of an enlargement of a portion of the graph shows a small distribution of cells up to 75 microseconds, a spike at 92 microseconds, and a few cells distributed up to around 180 microseconds. The small distribution between 69 and 75 microseconds may be explained by the switch checking for internally produced cells, a test which occurs every 1000 idle polls. Other infrequent events also occur in the switch, which maintains a simple operating system on top of its switching duties, and so occasionally a cell will arrive during the handling of a clock interrupt or some other processing. The switch only switches cells while it is idling — every other process has higher priority. The small spike at 92 microseconds corresponds to 25 microseconds after the main spike. The reason for this is unknown.

The other two graphs in figure 5.6 show the same routes and block size, except the results were gathered while the test system's switch was loaded. The same peak appears at 67 microseconds, but there is a noticeable spread of echo times at 69 to 71 microseconds (5.2% of cells), and again between 72 and 75 microseconds (1.9% of cells). These are caused by the echo cells being forced to wait for the switch to handle one of the video or audio cells prior to the echo cell.

Figure 5.7 shows the results of the timings for loaded and unloaded network using route B (figure 5.3). The main spike for the unloaded network occurs at

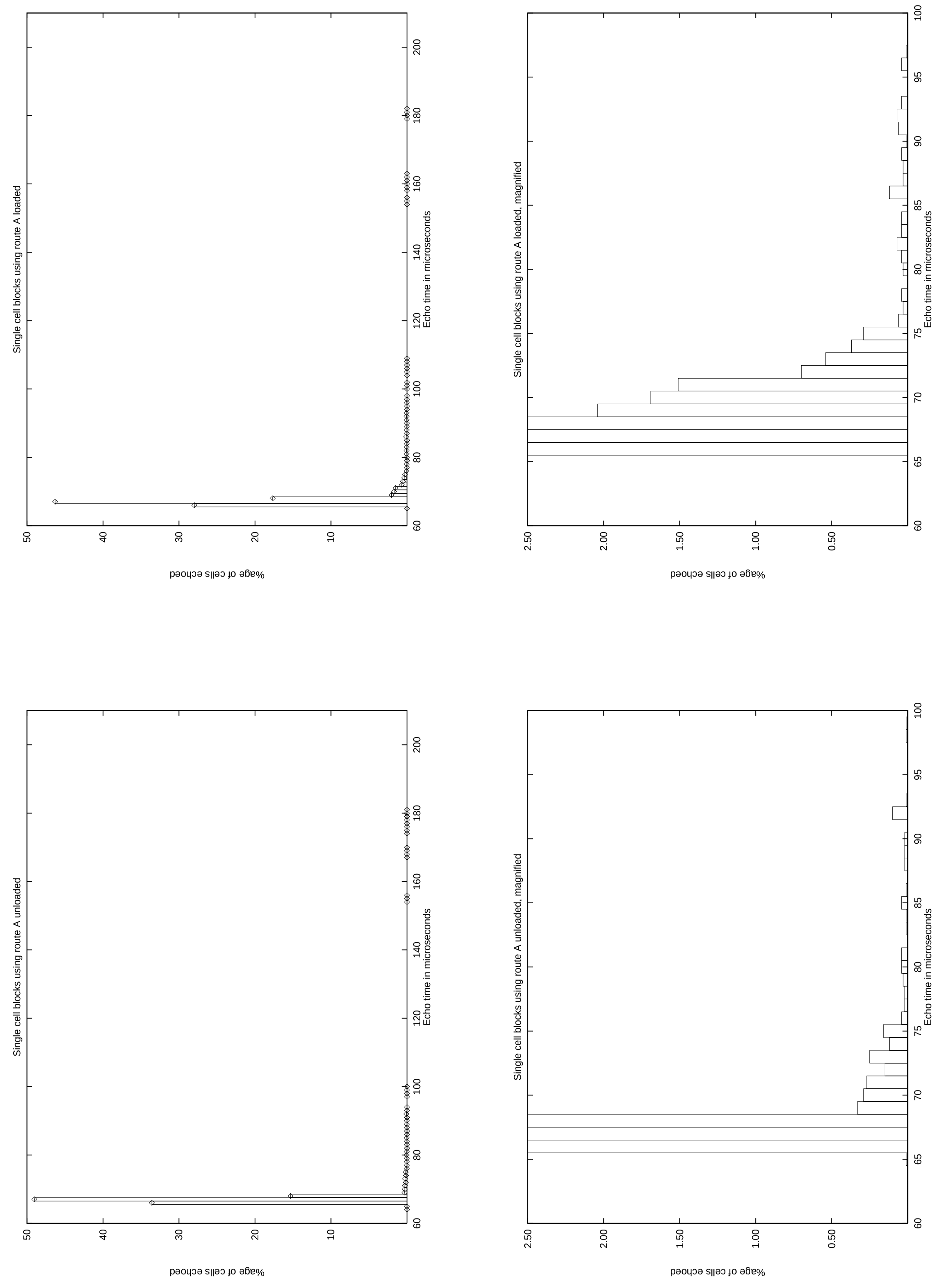


Figure 5.6: Echo times for a single cell, route A

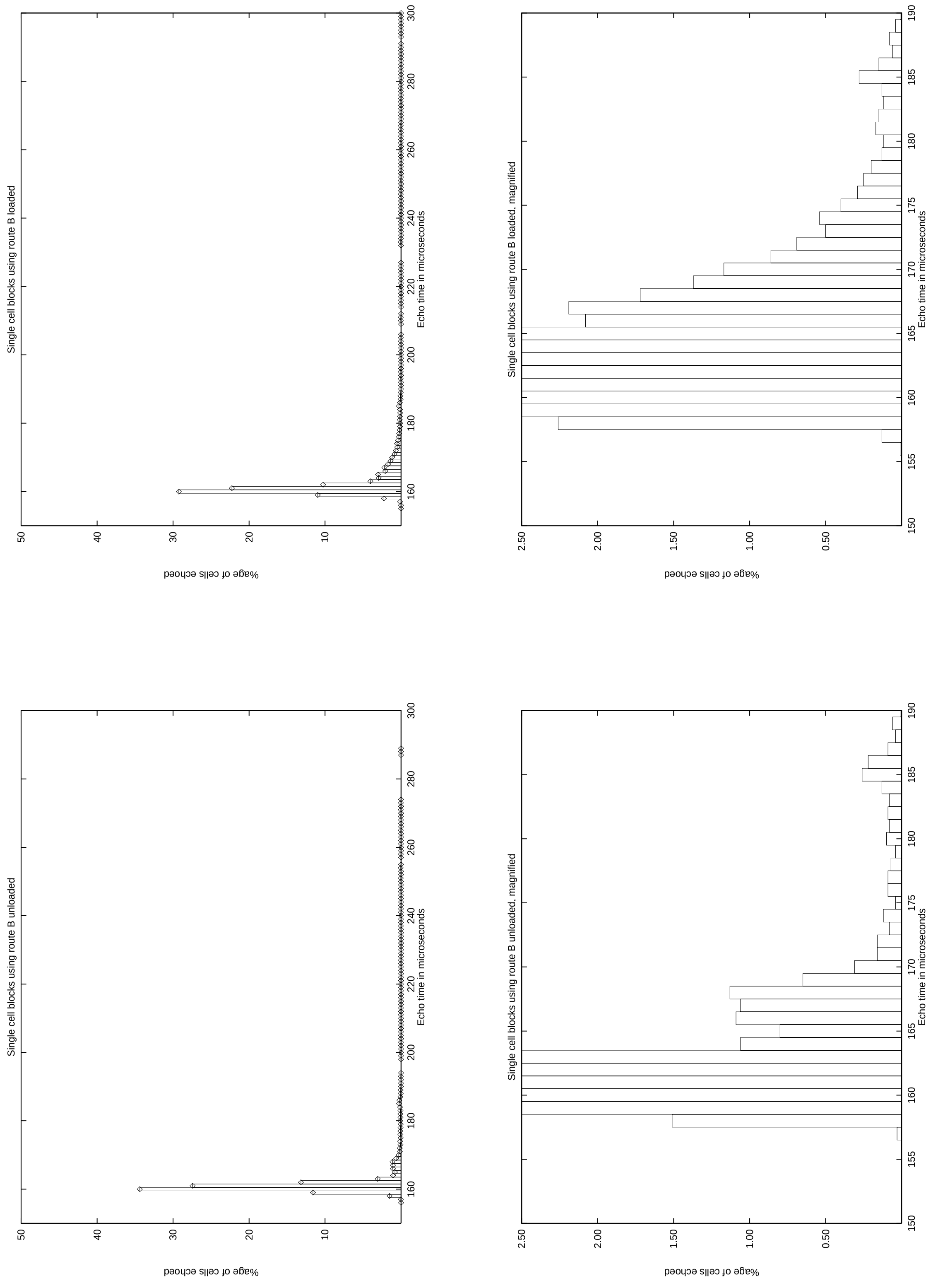


Figure 5.7: Echo times for a single cell, route B

160 microseconds, with a spread from 159 to 163 microseconds. A step similar to that from route A between 69 and 75 microseconds can be seen in the enlarged graph between 164 and 169 microseconds, followed by a tailing off. Note the small increase around 185 microseconds, 25 microseconds after the main spike. This corresponds to the 92 microsecond spike on the smaller route. The results for the loaded network, shown in the other two graphs in the figure, shows a much smoother tailing off in times, due to the extra cells the switches must cope with.

It is interesting to note that the time diagram (figure 5.5) shows that a switch will introduce an expected delay of $1.6 + 4.4 + 1.6 + 0.6 + 3.0 = 11.2$ microseconds in each direction and that route A has one switch whereas route B has five switches (four more switches). Therefore the difference in the peaks between route A and route B should be eight times 11.2 microseconds, as each switch is traversed twice, and therefore route B's peak should be from 89.6 microseconds after route A's peak, i.e. 159 to 161 microseconds.

Figure 5.8 shows the echo timing results for the longest path available, route C (figure 5.4). The unloaded system produces a distribution around a spike at 225 microseconds, spread from 223 to 228 microseconds, with a less well-defined step than previous unloaded systems from 229 to 235 microseconds. Also, larger proportions of cells are taking longer. Another point of note is the distribution of cell echo times around 250 microseconds: again this is 25 microseconds after the main distribution, and although it is starting to become hidden by the smoothing effect, definitely still there. The graphs for the loaded system again show a smoother tailing off, and are otherwise unremarkable.

As between route A and route B, the comparison between route A's and route C's peaks is interesting. Route C has seven more switches than route A in its path, and therefore can be expected to take fourteen times 11.2 microseconds longer than route A, i.e. its peak should be at $156.8 + 69$ to 71 microseconds, or 226 to 228. The peak is actually a little before, which can be attributed to the different architecture of the central hub switch, or, just as reasonably, inaccuracies in the timings of figure 5.5.

5.1.2.5 64-cell blocks

Before the echo results were available it was assumed that the single cell blocks would have the most consistent transit times. However, looking at the graphs of the results of using the short route A as the echo path (figure 5.9), which give the distribution of echo times for the first cell in a block of 64 cells, the distribution is a lot sharper than for the single cell blocks case shown above, with over 90% of the echoed cells taking 483 microseconds in both unloaded and loaded systems.

The distributions of echo times for longer routes (figures 5.10 and 5.11) are closer to those expected, with the times are distributed a lot less cleanly. Looking at the

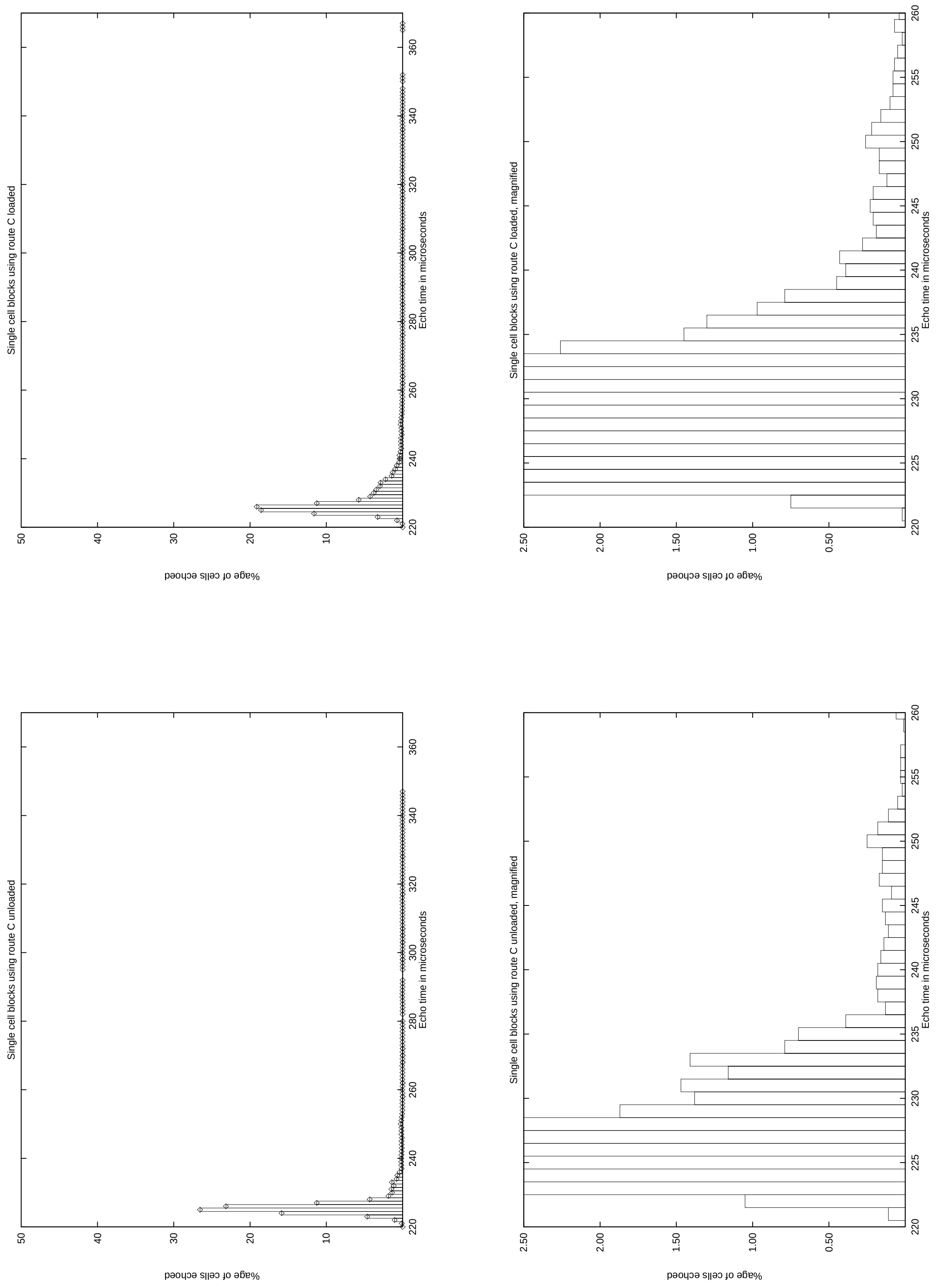


Figure 5.8: Echo times for a single cell, route C

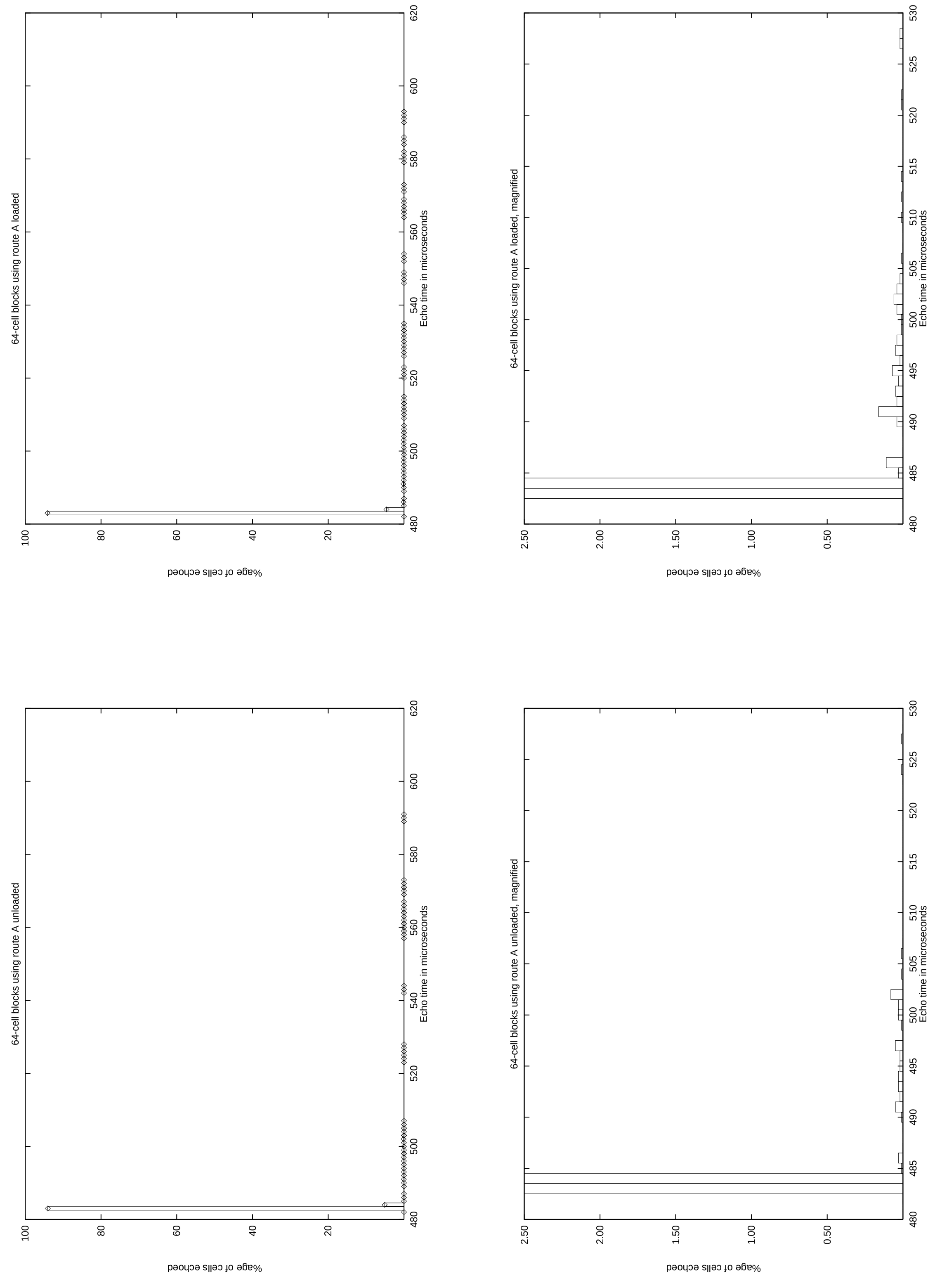


Figure 5.9: Echo times for first cell of a 64-cell block, route A

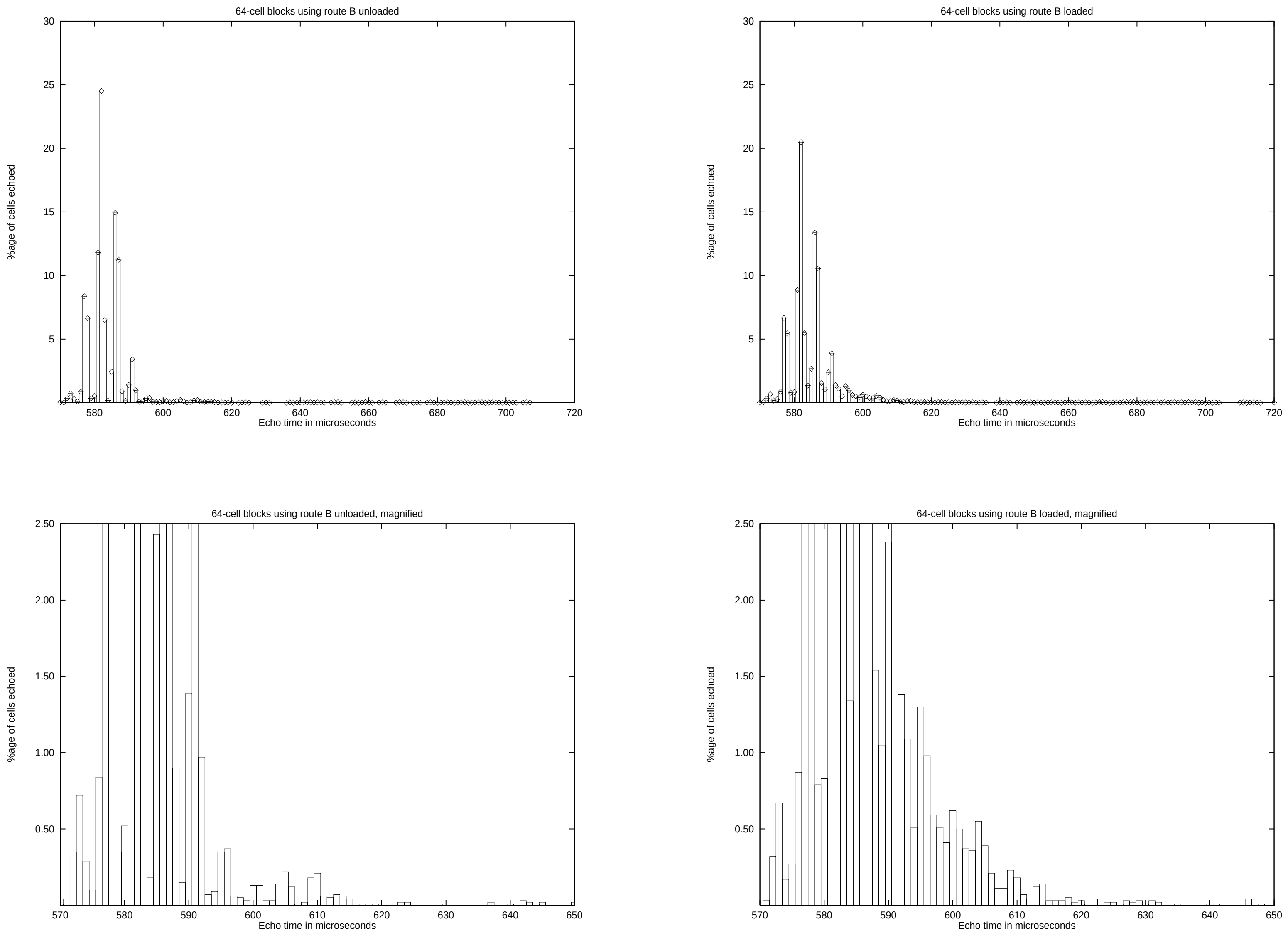


Figure 5.10: Echo times for first cell of a 64-cell block, route B

first graph in figure 5.10 there is a distribution centred on 582 microseconds, but instead of the smooth results with single cell blocks the distribution is a group of spikes, each about 4.5 microseconds apart, with the last one occurring at about 610 microseconds. These may correspond to the echoed cells waiting at switches along the return path for other cells in its block, at each switch this waiting time being the switching time of the other cell: in some sense, this is a beating effect between the forward and return paths. With the echo cells travelling back through 5 switches on route B it might be expected that there could only be at most 5 delays of the switching times, whereas the results in the graph show at least 9 delays. This can be explained by the unfair nature of the switch behaviour, as described in section 5.1.2.7 below.

The other two graphs in figure 5.10, with the results for the loaded system using route B, the patterning is still obvious, although it starts to get smoothed out due to the other traffic passing through the switches along the path.

Now a look at the graphs in figure 5.11 is informative. The unloaded graphs appear to show the 4.5 microsecond spike interval occurring throughout the magnified graph, that is 15 spikes, although the number of cells taking 706 microseconds is probably too small to be significant. With much longer runs the structure may become clearer. The overall distribution is also becoming wider.

5.1.2.6 Analysis of results

The sharp distribution for the 64-cell blocks over route A perhaps leads to the conclusion that a large block may be suitable for the distribution of time messages, as the transit time of the messages over such a route could be calculated very accurately. The single cell blocks, however, have only a slightly wider distribution of times for the same route, both loaded and unloaded. The clock message filter in the logical clock model can remove any messages in either case which have transit times outside the 90th percentile of the transit time distribution, which will limit the inaccuracy to a microsecond in both cases.

Over the longer routes it is clear from inspection of the related graphs that the single cell blocks have a much better distribution of echo times for accurate calculation of a transit time. More than that, the distributions suggest an algorithm for the clock message filter in the logical clock model, for filtering out clock messages with inaccurate transit times. The single cell blocks have a minimum transit time, which most accurately defines the transit time in one of the two directions it has travelled. Should a cell take longer than the minimum transit time it has either been delayed on one or both legs of its echo journey, although which leg cannot be immediately told. Even on the longest route under load, 90% of the cells take between 221 and 234 microseconds, and 50% between 221 and 226 microseconds. So, from the results obtained here, if a suitable cut-off

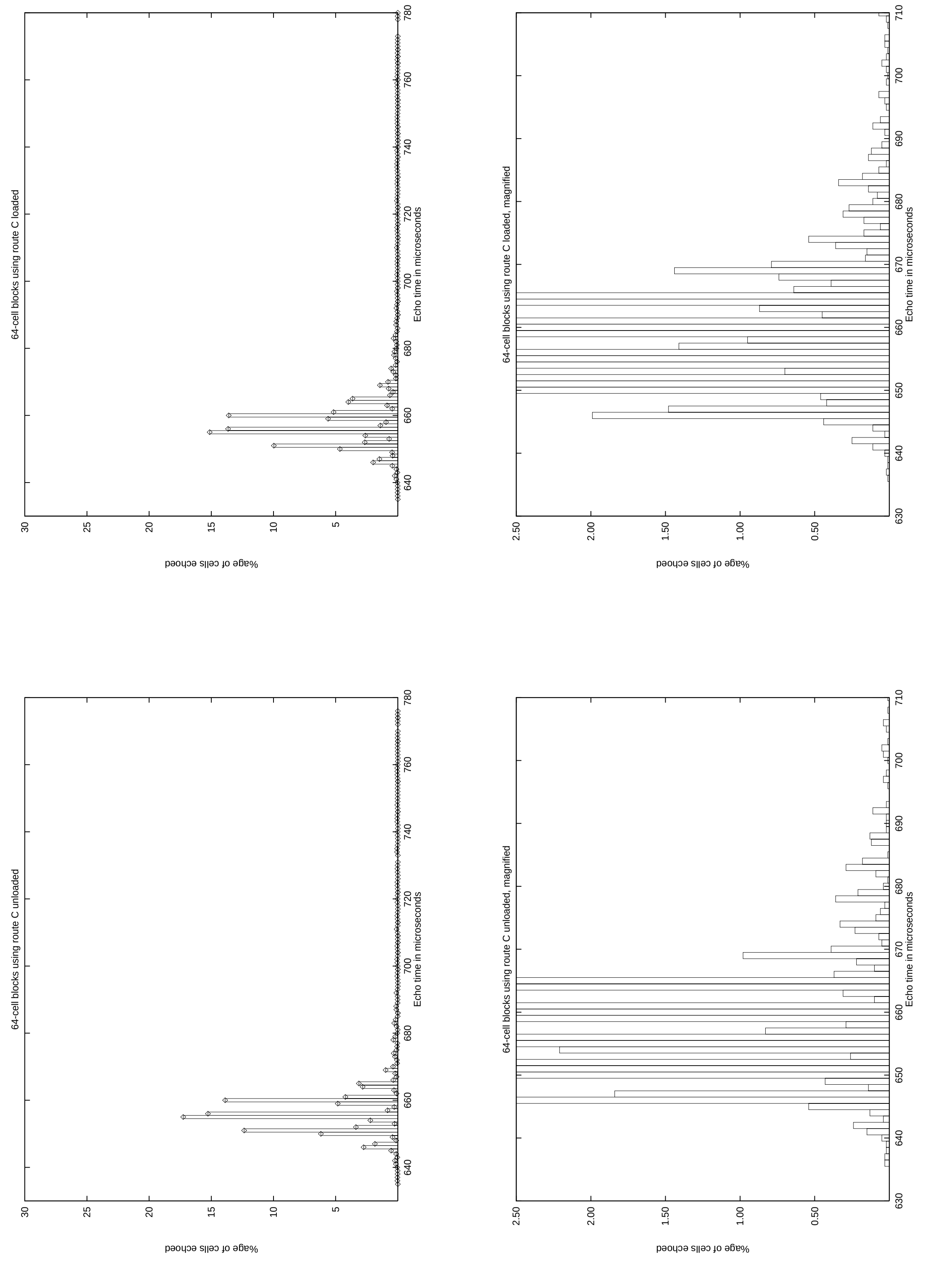


Figure 5.11: Echo times for first cell of a 64-cell block, route C

point is chosen then the inaccuracy in the transit time calculation can be kept very small.

There are two features of the graphs so far unexplained: the 25 microsecond peak for single cell blocks after the main peak in the distribution, and the extremely sharp distribution for the first cell of 64-cell blocks over the short route. The first of these currently has no known explanation. The second is less interesting, as it is probably just an artifact of the implementation: perhaps the actual time is 483.6 ± 0.6 microseconds, with the error distributed normally, whereas for single cell blocks the time is 70.2 ± 0.6 microseconds, therefore providing actual values of 69 and 70 microseconds.

5.1.2.7 Operation of the switches

It should be noted that the switch operation is not fair. The ports are slightly preferentially treated under medium loads, although fairness is reinstated when the switch is under full load.

The switch runs a polling loop for switching cells. At the start of the polling loop, the ports which require servicing are recorded, and then they are serviced in a fixed order. When the recorded ports have all been serviced the polling loop restarts, recording a new set of ports which require servicing. The unfairness manifests itself in the following way.

Say that an ATM cell arrives on port A of the switch. The polling loop then records port A as the 'ports' to service. While it is servicing the cell on port A a cell arrives on port B. After this another cell arrives on port A. When the processor finishes servicing the first cell it reinterrogates the ports which require servicing. These are both ports A and B, and they will be handled in that order. So, the second cell which arrived on port A, later than the cell which arrived on port B, is forwarded first. If the ports are reversed, the cell arriving at the new port A would be serviced first. Hence port A is preferentially treated.

Under full load (FIFOs constantly not empty) the switch will service ports in a true round-robin fashion. It is only under medium loads, where its FIFOs are operating almost empty, that this behaviour appears.

The first cell in a 64-cell block will take an indeterminate time, up to 1 microsecond, before the polling loop of the first switch will detect its presence in its receive FIFO. Thereafter the behaviour of that switch is deterministic (except for clock interrupts), as the receive FIFO is full every second time round the polling loop. With the short route, the first cell arrives immediately at the echo server, which takes a fixed time to insert the cell into the network, and so into the receive FIFO of the return port of the switch. As the switch behaviour is deterministic in this case, it will take a fixed time to react and forward the cell to the data source, hence the very tight distribution of times for the first cell of a 64-cell block.

As described above, the number of smaller spikes in the distribution for the echo time of the first cell from a 64-cell block over many switches may be accounted for by the interference between cells travelling along the forward and return paths. The maximum expected spikes could be the number of switches which the first cell encouters along the return path. However, the unfair nature of the switch probably has an effect, sometimes introducing two wait times for a cell at a single switch. There should never be three wait times, so the maximum number of spikes should be at most two times the number of switches the cell passes through.

5.2 Cambridge Fast Ring network transit times

This section describes the work performed to analyse the echoing of cells and blocks between two Pandora network cards using the CFR, with the aim of evaluating the accuracy to which the transit times of those cells and blocks can be calculated. Performance problems related to the capabilities of the network cards used in this scenario are very similar to those that would be present in a distributed clock system for Pandora. As for the ATM switch network evaluation, blocks of cells were echoed from and to a data source by an echo server, with the cells being timestamped with local clocks at various points along their travel.

5.2.1 The echo timing system

It should be noted that it was envisaged that, if more than one CFR were to be used in the Pandora system, then each of the rings would have its own time server. The service provided by the time server would then only have to operate on a local ring, and so the echo timing system only evaluates a single CFR.

The echo timing system for the CFR is similar to that used for the ATM switch network at its upper levels, but the lower levels are much different due to the nature of access to the CFR. The cell transmission mechanism had to be changed and extra complications due to the management of the ring had to be added. Another problem related to the CFR is the lack of any internetwork facilities for connection to Unix hosts for collecting the results in a suitable way. As a solution to this problem, a Pandora ATM network card is used to interact with a Pandora CFR network card running as the data source. The echo server is another Pandora CFR card running on its own.

5.2.1.1 Data source and echo server software

The data source and echo server CFR card's run the same CFR software. This is structurally similar to the ATM version of the echo timing system, and uses the MSNL protocol in the same way to allow the data sourcing and cell forwarding virtual circuits to be made.

- Port 0.0.0.2

  Requesting a connection to port 0.0.0.2 sets up the virtual circuit for incoming data which will be forwarded onto the CFR. Normally a connection to this port is made from port 0.0.0.4 of another network card (data source or echo server).

- Port 0.0.0.3

  Port 0.0.0.3 is the port used for extra connection management inside the echo server (unlike the ATM software, the data source on the CFR uses its ATM connectivity instead). When a cell is received on this connection by the echo server, the echo server broadcasts the cell on the CFR after having inserted its own MSNL address as the source of the cell. This allows the ATM node connected to port 0.0.0.3 of the echo server to force the echo server to make a connection on its port 0.0.0.4 to another network card on the CFR, normally the data source.

- Port 0.0.0.4

  Port 0.0.0.4 is controlled by cells sent to an MSNL connection made to port 0.0.0.3. The connections made on this port form the outgoing half of the echoed data path. When one of these connections is formed upon reception of an MSNL connection reply PDU, the VCI from the reply is stored, and is inserted into all cells retransmitted as echos or generated by the data source.

The lower levels of the software are based on an interrupt handler and a circular buffer of packets which have to be handled. Packets are added to the circular buffer either by being received from the CFR, or in the case of the data source, from the ATM network card. The interrupt handler is a high priority process, which must handle four possible interrupt sources from the CFR:

- Ring broken

  The CFR CMOS chip requires servicing when the slot structure of the CFR breaks. This may be due to stations being added to or removed from the network, or because the monitor station is unhappy with the framing. The network handler must forget the packet it had in transit on the ring and acknowledge the ring broken state with the CFR CMOS chip before any more transmissions may take place.

- Packet received

  The packet received signal from the CFR CMOS chip indicates that the chip has received a packet into its internal FIFO. The network handler reads the packet from the CFR CMOS chip into its circular buffer, then awaits the next interrupt source. This process takes about 31 microseconds, from the interrupt firing to the finish of the packet received handling.

  If a packet is not in transit on the CFR then the first packet in the circular buffer will then be serviced.

- Packet transmitted

  The packet transmitted signal occurs when the CFR CMOS chip receives back the packet it inserted onto the CFR for transmission. This informs the network handler that it can transmit another packet by inserting the data into the CFR CMOS chip, so it forces the first packet in the circular buffer to be handled.

- Packet TOGged

  The packet togged signal occurs when the CFR CMOS chip tries to transmit a packet of times, failing each time for one of many possible reasons. For the echo server and data source these are very rare events, and are ignored, just causing the first packet in the circular buffer to be handled.

The packets are timestamped at various stages along their path on the CFR. The first 16 bits of the cell contain a count of the number of times the packet has been timestamped. The cell is first timestamped at its first transmission by the data source. Three clock values are inserted at this point, the time of the last successful CFR packet transmission, the time of the interrupt which prompted the packet transmission, and the time the packet is inserted into the CFR transmission FIFO. When the packet arrives at the echo server it is timestamped the time the interrupt was handled. Then, when the packet is transmitted back to the data source the three clock values are then inserted by the echo server. Finally, on arrival at the data source the packet is given its received time timestamp.

5.2.1.2 Results gathering system

As in the ATM switch network echo timing, the results gathering system is a DecStation 5000/240 fitted with a YES board (a TurboChannel ATM card), running OSF/1, with the specific software written in C. It uses MSNL to form a control connection with the data source's ATM network card. Using this connection it controls the data source and echo server to create the data path, and command the data source to generate data. The order of events is:

1. Connect results gatherer to data source, control path

   The results gathering software running on the DECstation first connects to the ATM port of the data source. This connection is also the return path for the echoed data from the CFR onto the ATM network.

2. Connect data source to echo server, outward data path

   The data source is told by the results gathering system, using the ATM connection, to make a connection on its port 0.0.0.4 to the echo server's port 0.0.0.2 over the CFR. This is the outgoing data connection.

3. Connect echo server to data source, return data path

   The data source is then told by the results gathering system to make a connection to port 0.0.0.3 of the echo server. Using this connection it forces the echo server to connect its port 0.0.0.4 to the data source's port 0.0.0.2 (this is the return path for the echoed data) following which the control connection to the echo server is terminated.

Once the above has been performed, the results gathering software requests the data source to initiate a data block. This is transmitted on its outward data path connection. The data is echoed by the echo server, packet by packet, and returns to the data source along the return data path. The data source passes the packets from its CFR network card to its ATM network card, from where it reaches the results gathering system.

It can be seen from the outline above that the software allows for more than two hops in the data path from the outgoing side of the data source to its incoming side. Instead of making the echo server *echo A* connect its port 0.0.0.4 to the data source's port 0.0.0.2 it may connect to a second echo server's port 0.0.0.2. This second echo server (*echo B*) can then be forced with its port 0.0.0.3 to connect its port 0.0.0.4 to the data source's port 0.0.0.2. Then the data path is data source, echo A, echo B, data source.

One problem negotiated by the results gathering system is the different cell sizes used by the ATM network (48 data bytes) and the CFR (32 data bytes). The packets on the CFR use MDL for reassembly into blocks. When forwarded onto the ATM network by the data source the 32 bytes of packet data are placed in the first 48 bytes of the ATM cell payload and the reassembly information is changed to SAR1a. The blocks of data received by the results gathering system can then be split up into 48 byte cells, the first 32 bytes of each of which actually travelled on the CFR. The timestamps stored by the data source and echo server can therefore be extracted from the first 32 bytes out of every 48 bytes of the blocks received by the results system.

5.2.2 Results

The loading of the CFR has little effect on the actual performance of two stations which are not themselves loaded, unless the network load is excessive. The particular CFR on which the Pandora systems are connected cannot become very heavily loaded as the Pandora systems are limited in network bandwidth to about 3 Mbps aggregate bandwidth (transmit and receive), so requiring 32 stations to fill a 50 Mbps CFR, which is more than exist. So all the results presented were performed on an unloaded CFR.

5.2.2.1 Packet timestamps

When a packet arrives as part of a block at the results gathering system it contains eight important time values:

1. Time last successful transmission was notified by CFR to data source
2. Time of interrupt prompting packet transmission by data source
3. Time of insertion of packet into CFR transmission FIFO by data source
4. Time of interrupt for reception of the packet by echo server
5. Time last successful transmission was notified by CFR to echo server
6. Time of interrupt prompting packet transmission by echo server
7. Time of insertion of packet into CFR transmission FIFO by echo server
8. Time of interrupt for reception of the packet by data source

These results are presented here in three forms. Firstly, the operation of both the data source and echo server is displayed graphically, by sorting the times of their events (time values 1, 2, 3 and 8 for the data source, 4, 5, 6 and 7 for the echo server) for a block of cells and plotting the events against the time of the events. Secondly the difference between time value 1 and the previous packet's time value 3 indicates the time the data source thought the previous packet was on the network, and similarly the difference between time value 5 of a packet and the previous packet's time value 7 is the time the echo server thought the previous packet was on the network. Thirdly the difference between time value

1 of a packet and time value 4 of the previous packet gives the clock difference between the data source and the echo server plus the network transit time from echo server to data source, and similarly for time value 5 of a packet and the previous packet's time value 8.

5.2.2.2 Event timelines

The timelines shown in figure 5.12 give the ordering of events inside the data source and echo server respectively. This allows analysis of the software and a comparison of the two cards' clock values. The timelines describe the echoing of a single block consisting of 31 CFR packets.

The first event on the data source's timeline is an interrupt at clock value 27432, which forces the data source to check its circular buffer for packets to handle. It finds a packet to be transmitted, inserting the packet into the CFR transmit FIFO at 27449. The data source then goes idle, waiting for the interupt it receives at 27502. This interrupt is due to the CFR chip having marked the first cell as transmitted, as can be seen by the next event being "CFR TXed". This permits the data source to transmit the next packet in its circular buffer, which occurs at 27519. At clock value 27547 it received the first cell back from the echo server, and so it continues.

The first event on the echo server's timeline is an interrupt due to the CFR chip receiving a packet (the interrupt and "CFR RXed" events are marked as simultaneous), occurring at clock value 62238. This packet is transferred to the echo server's circular buffer, which can then be serviced and the packet returned by transmitting it, which happens at 62267. Meanwhile the data source has transmitted the second packet ($27519 - 27449 = 70$ microseconds after the first packet), which is received by the echo server at 62300 (62 microseconds after it received the first packet). The next event for the echo server is the acknowledgement from the CFR chip that its first packet has been transmitted, occurring at 62322. This permits the second packet to be transmitted, occurring at 62334. And so operation continues.

5.2.2.3 Packet transit times

As suggested above, the packet transmission times (i.e. time from insertion into CFR transmit FIFO to the successful transmission notification being received) can be calculated from the clock values in the echoed packets. The distribution of calculated times is given for both the data source and echo server separately in figure 5.13.

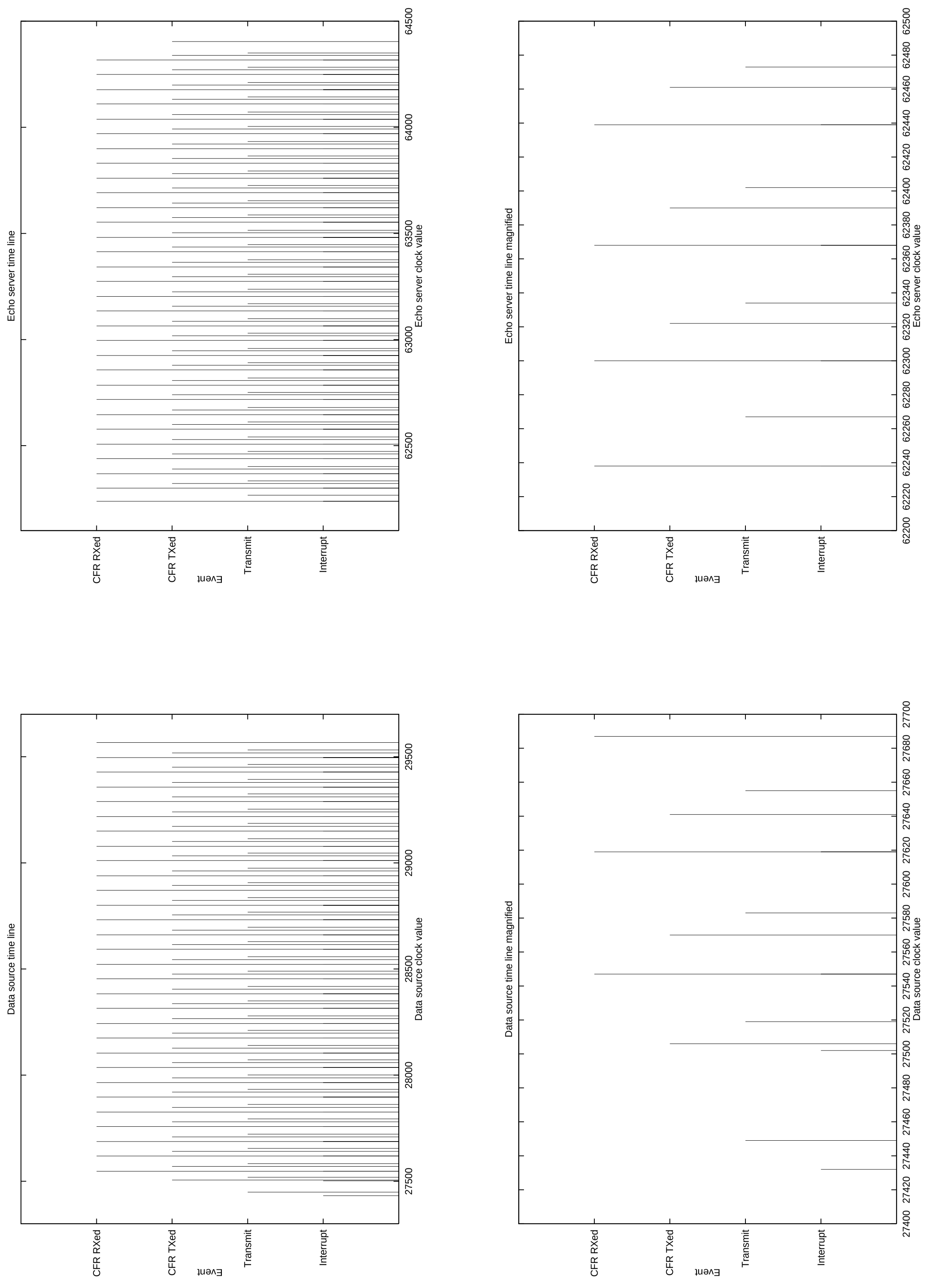


Figure 5.12: Timelines for data source and echo server

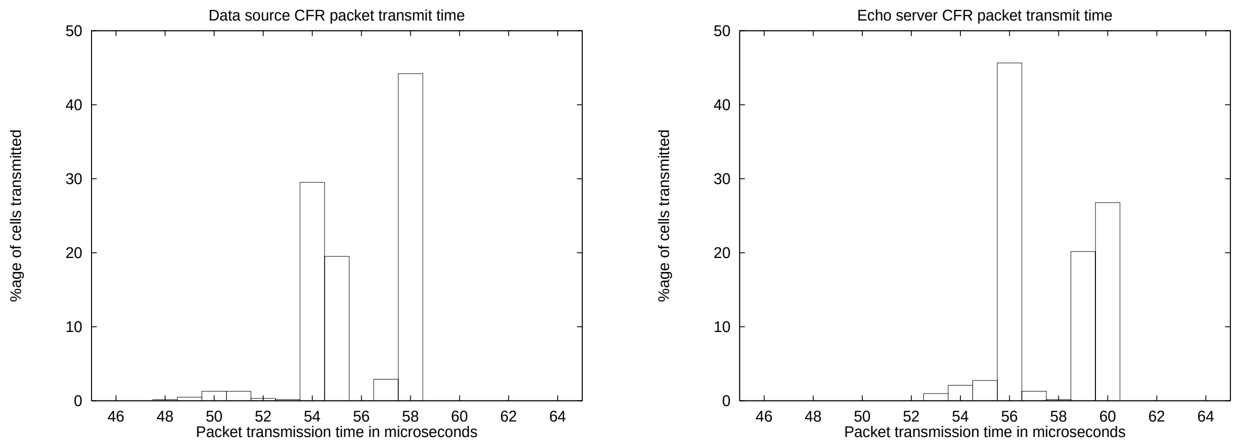


Figure 5.13: Packet transmission times

5.2.2.4 Clock differences

The third form of presentation suggested above is shown in figure 5.14. This consists of two traces, the upper one generated from the difference between the fifth clock value of a packet and the eighth clock value of the previous packet, the lower one generated from the difference between the fourth clock value of a packet and the first clock value of the next packet.

Each trace consists of groups of points, one group corresponding to a block of 31 CFR packets. Each point in a group gives the result from the contents of a single cell. The horizontal axis of the graphs corresponds to the results gatherer's time since the first cell of the first block was transmitted.

5.2.3 Analysis of results

This section describes the analysis of the results that were given above. Firstly it covers the software and hardware features of the systems which are relevant to the results. Then it covers the calculation of the CFR ring rotation time and relative clock drifts, with a summary of the effects that these have on the accuracy of the calculation of packet transit times.

5.2.3.1 Features of the software and hardware

An examination of the timeline for the data source shows that, between the first packet being received by the data source from the echo server and the next packet being transmitted no interrupt occurs — that is, at the end of handling the reception of the packet the CFR will already have signalled the successful transmission of the previous packet. This implies that there is some inaccuracy

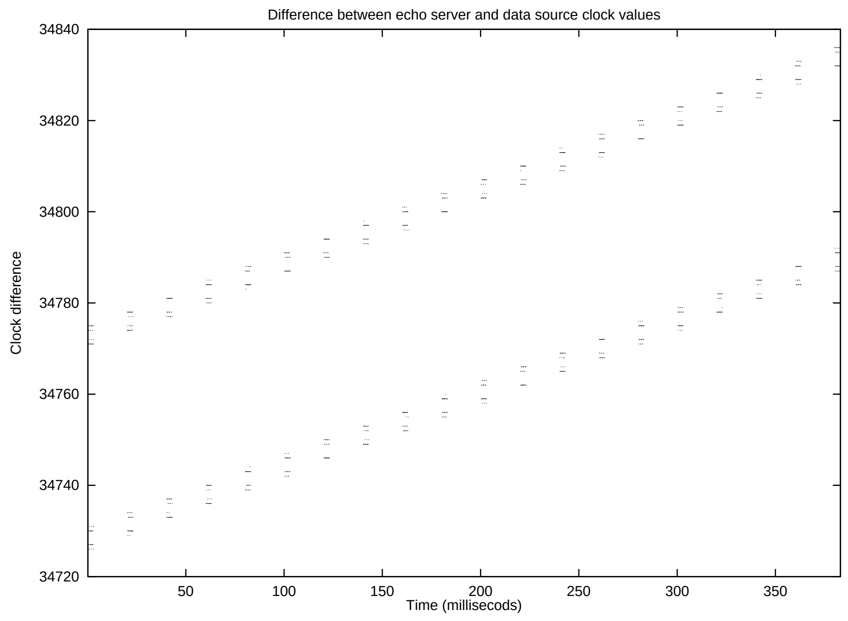


Figure 5.14: Difference in clock values

in the measurements for the clock values of the successful transmissions. This is also the reason why, in the event timelines description above, the data source transferred cells to the CFR chip 70 microseconds apart yet they were received by the echo server 62 microseconds apart.

A feature of the software is the length of time it takes to copy packets into the CFR transmission FIFO after they have been timestamped. This makes the timestamp in the packet out of date by 6 microseconds.

5.2.3.2 Packet transit times

The most important feature of the packet transit time distributions shown in 5.13 is their width, which corresponds to that expected from the network. The width is about 8 microseconds, which corresponds well to that seen by [63], of 8.2 microseconds. This distribution width implies that the timestamping method used above can give an accuracy in calculating a packet transmission time of better than 5 microseconds.

5.2.3.3 Ring rotation time

When a packet is placed by the CFR chip onto the network it travels for a complete revolution of the ring before the CFR chip frees the slot it was carried in and generates its successful packet transmission notification signal. As the graphs in figure 5.13 correspond to the distributions of the differences between the packet insertion times into the CFR and the corresponding success notification, they should give an idea of the time for a complete revolution of the ring.

However, after a packet is inserted into the CFR chips transmission FIFO the CFR chip must wait for an empty slot to arrive, into which it can insert the packet data. On a completely unloaded network (as shown in the graphs) this will take at most one slot time plus the gap time (the width of the distribution). The length of this delay cannot be predicted by the transmitter, and it accounts for the variation in packet transmission times shown in the graphs. The minimum value should correspond to the ring rotation time.

Another method for calculating the ring rotation time is to compare the two traces in figure 5.14. The lower trace is the difference between the clock value of the data source when it is notified of a successful transmission of a packet and the clock value of the echo server when it was notified that the packet had been received. The upper trace is the difference between the clock value of the data source when it was notified of a packet being received and the clock value of the echo server when it was notified the packet had been successfully transmitted. The first of these corresponds to the difference in clock values between the echo server and data source minus the time taken for the packet to return along the ring from the echo server to the data source. The second corresponds to the difference in clock values between the echo server and data source plus the time taken for the packet to travel along the ring from the data source to the echo server. The difference between the two values is the time taken for the packet to travel from the data source to the echo server plus the time taken for the packet to travel from the echo server to the data source, i.e. the length of the ring. So the difference is an estimate of the time taken for a packet to travel the length of the ring, which is the ring rotation time.

This second method gives ring rotation times of 45 microseconds. It may be regarded as more accurate than the graphs shown in figure 5.13 as the time taken to transfer data into the transmission FIFO and the time taken by the CFR chip in waiting for an empty slot to transmit into are removed. However, there is an error in the value equal to twice the time taken by the software to respond to the successful transmission signal minus the time taken to respond to a packet received signal. As described above, the time taken to respond to the successful transmission signal is effected by the time taken to receive the previous cell. This interdependence also has an effect on the error.

These two calculations fit well with the size of the ring (six slots) and the timings seen in [63], which is a single slot ring, where the rotation time is seen to be 8.2 microseconds.

5.2.3.4 Relative clock drift

The two traces in figure 5.14 show that the difference between the clock values of the echo server and the data source does not remain constant, but drifts by about 60 microseconds in the 400 milliseconds of the graph. This corresponds to a relative clock drift of 150 microseconds per second. Over the 2 milliseconds duration of a 31-packet block this is negligible (0.3 microseconds), but it does indicate that readings taken over longer time scales need to be corrected.

5.2.3.5 Method and accuracy of calculating packet transit times

The research described in the above section was peformed to find out how message transit times could be accurately calculated for the distribution of a global clock. As is clear from the above results and analysis, there are barriers to producing transit times of an accuracy of the order of microseconds. For example, the ring rotation time is about 40 microseconds, and the inaccuracy of the measurement is half the rotation time (see section 4.5.3.2). However, the above analysis shows that this is at least calculable, and so a known accuracy can be given.

For accurate calculation of transit times, the above results suggest the recording of clock values on reception and successful transmission will give results accurate to better than 10 microseconds, plus the addition ring rotation time inaccuracy. So if messages consisted of two packets, with the first packet stamped with the received clock value and the second packet containing the successful transmission clock value of the first packet, then for messages travelling between two points their transit times should be calculable to the ring rotation time plus or minus 10 microseconds.

5.3 Logical clock models

Three logical clock models were described in section 4.5.4. The graphs in figure 5.15 show some results from experiments with the logical clock models. This section describes the experiments carried out in improving and finalising the logical clock model used in the research, and gives descriptions of the behaviour described in the graphs.

5.3.1 Experimental system

The experiments for examining the performance of logical clock models were started after the initial implementation of a logical clock, as its performance was not satisfactory. This algorithm was re-coded in C using an event-based

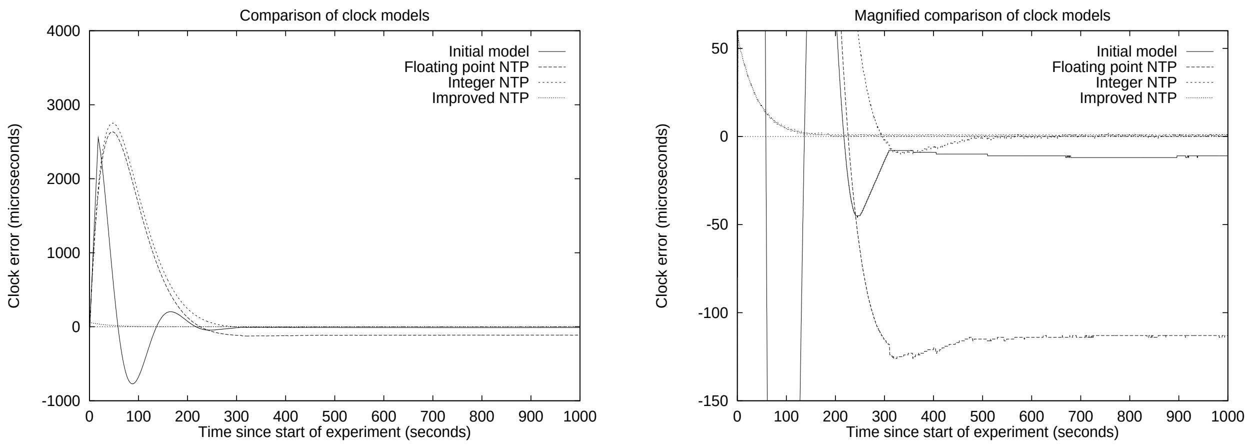


Figure 5.15: Errors in distributed clock values

(floating point does not use message distribution time estimates)

system to accurately simulate the finished system, with a control language to aid experimentation. The experiments mirrored the poor performance, and so more research was carried out into logical clocks.

At this stage the Fuzzball logical clock was implemented in the experimental system, using floating point values throughout to investigate the precisions required for implementing the model in fixed point arithmetic in occam in the finished system. Suitable values for the frequency, phase and compliance weights were chosen to provide a sensible capture rate and accuracy. No clock filter was implemented, as in the experimental system, at this time, there were no message distribution time estimates.

Before implementing the Fuzzball logical clock on the finished system using fixed point arithmetic it too was added to the experimental system. At first a simple conversion from floating point to fixed point was performed, then it was enhanced to use an estimate of the message distribution time in its handling of clock messages. It maintained a running average of the message distribution times, and subtracted this value from the local clock value of the messages. This helped to remove the fixed offset in the clock once message distribution estimates were added to the system.

Finally the experimental fixed point arithmetic system was enhanced again, by adding a clock message filter, improved message distribution estimate handling, and initial frequency error calculation. The clock message filter maintains a minimum message distribution time, which is the minimum value of all the clock message distribution estimates, and a filter value. The filter value is increased if a message arrives with a message distribution estimate that is greater than the minimum value by more than about 120% of the difference between the filter value and the minium value. The filter value is decreased if a message arrives with a message distribution estimate that is greater than the minimum value by less than about 80% of the difference between the filter value and the minium value. The filter value is therefore maintained at slightly above the mode of the distribution of message times. Clock messages are ignored if their message distribution times are greater than the filter value. The inaccuracy in the message distribution estimate is calculated to be the difference between it and the minimum message distribution time. Finally, the initial frequency error is calculated using the first two messages received by the clock model, using the differences in the local clock values and the messages' clock values.

5.3.2 Experimental results

Some experimental results are shown in figure 5.15. These graphs dsisplay the differences between the clocks as calculated by the logical clock models described in the previous section and the actual global clock. The local clock was taken to have a fixed frequency error of -150 parts per million (compare CFR drifts obseved above). The clock messages were given distribution times taken from the real data collected using the ATM switch network for single cell blocks using route C (see figure 5.8 and section 5.1.2.4), and random errors in the distribution time estimates dependent on their distribution times.

The initial logical clock model implemented ("Initial model") overshoots three times before settling at an error of roughly -10 microseconds. The graph for this model is not smooth, with rapid changes in clock adjustment occurring at 20 seconds and 290 seconds into the experiment. This is because the effect of rogue values is not removed with a filter, and the balance between reducing frequency and phase error is not satisfactory.

The floating point Fuzzball logical clock model experiment ("Floating point NTP") shows a smooth curve until just after 300 seconds, where the results become unsettled until an error of about -115 microseconds is attained (this is roughly half the average message distribution time).

The fixed point Fuzzball logical clock model ("Integer NTP") generates a slightly larger error initially before settling to vary from -1 to $+1$ microseconds error, after a slight overshoot at 300 seconds. The larger initial error can be accounted for by the handling of message transmission times, which bias the whole graph in the positive error direction.

The last, enhanced fixed point Fuzzball logical clock model ("Improved NTP") does not exhibit the large initial error distribution, instead it starts from the first second with an error of about ± 50 microseconds. Within one minute it is within ±10 microseconds, and within 200 seconds it has a single microsecond of error, the best that can be achieved. It is interesting to note that the clock message filter implemented in this model allows 86% of the messages to be used, all of which have distribution times within 230 microseconds, most within 226 microseconds. The latter can have distribution errors of at most 5 microseconds.

Chapter 6

Clock distribution implementation

This chapter describes the implementation of the distribution of a precise, accurate clock to Pandora systems over both the CFR and ATM switch networks. It is divided into five sections, covering the logical clock model implementation (used in many places inside Pandora), the time server implementation, the network subsystem's time client required for both the CFR and ATM switch networks, and the extensions to the Pandora box software required to distribute time inside a Pandora box. It uses the logical clock models specified in section 4.5.4 and examined in section 5.3, the research into message transmission times described in sections 5.1 and 5.2, and enhances the Pandora software described in chapters 2 and 3.

6.1 Logical clock model implementation

As described in section 4.5.4, an initial attempt at implementing a logical clock model was performed before research into this area was carried out. This clock model was rudimentary, using 32-bit integers to describe phase and frequency errors of the derived clock with respect to the clock messages received as input to microsecond precision. The second implementation for transputer systems was the enhanced fixed point Fuzzball logical clock. This implementation was performed by converting the experimental implementation (see section 5.3) from C into occam.

The logical clock model implementation consists of three processes which share some state. The first of these takes clock messages and updates the clock correction values in the shared state. The second process updates the clock difference value, which contains information relating to the difference between the local clock value and the derived global clock value. The third process waits for requests from timestamping systems, reads the derived global clock value using the local clock value and the clock difference value from the shared state, and presents this as the response.

All three processes run at high priority, so that access to the shared state may occur atomically. The processes in more detail are:

- Clock correction calculation process

  This process waits for messages, which consist of four integers, on its input channel. The four integer values are the global clock value in seconds and microseconds since midnight January 6th 1980, the message transit time estimate, and the local clock value corresponding to the arrival of the message in the processor.

  The clock message filtering occurs here, as well as calculation of frequency and phase errors of the derived global clock generated by the logical clock model.

  For debugging and informational purposes the frequency error, phase error, compliance, frequency error divisor, and filter values can be displayed on the 16-character LED display attached to the slave transputer of a Pandora network card, if the logical clock model implementation is running on a network card.

- Clock difference adjustment process

  The *clock difference adjustment* process waits on a timer between adjusting the clock difference value in the shared state. The number of microseconds by which the clock difference value needs to be adjusted by in one second is calculated; from this the time interval required between single microsecond adjustments of the clock difference value is calculated. If this time interval is too small (less than a millisecond), perhaps because the phase error is large, or the local clock runs much too fast or too slow, so introducing a large frequency error, then a time interval of a millisecond is used and each time the clock difference value is adjusted by the appropriate number of microseconds this interval corresponds to.

  In the enhanced fixed point Fuzzball logical clock model, the *clock difference adjustment* process also implements the adjustments to the phase error and clock correction parameters, which occur at 0.1 second intervals. This in turn leads to a recalculation of the above microsecond adjustment intervals and adjustment values.

- Clock reading process

The *clock reading* process simply waits for a message, the contents of which are ignored. When one is received the local clock value is read. From this value and the clock difference value the global clock value is calculated according to the logical clock model. The result, in either seconds and microseconds, or as a 64-bit microsecond value, is sent on its outward channel as a response. (In Pandora systems the data source timestamps use the bottom 32 bits of the 64-bit microsecond value).

6.2 Time server implementation

This section describes the implementation of the time server itself. The time server uses the Pandora network cards with an additional serial card designed specifically for the time server to access the Navstar GPS receiver unit.

6.2.1 Navstar GPS interface

The serial card contains a T222 transputer to control a UART, through which it communicates using RS422 to a Navstar GPS receiver unit. The GPS receiver can supply positioning, timing and operational information, and can be told to operate in suitable modes to optimise the accuracy of its timing information. It also provides a single open-collector output which is asserted and then deasserted synchronised with the receiver's derived GPS clock. This signal can interrupt the T222 transputer, as can the UART if it needs servicing.

The software for the T222 in the serial card consists of four parts. Firstly it has a high priority interrupt handling routine, which reads the T222 internal clock as soon as it wakes up. The interrupt source is then serviced. If the UART has received a byte then the interrupt alerts the second process, which handles the UART. This runs at low priority, so as not to effect the timing of the interrupt handler. It inserts bytes into and extracts bytes from the UART's FIFOs. Any bytes extracted are built into messages, which are then sent to the third process, which decodes the messages. The fourth process uses a timer to generate regular requests to the GPS receiver unit for the time, with an interval of 1.25 seconds between requests. The requests must go through the UART handler, which manages a queue of bytes for the UART's transmit FIFO.

Accurate time information is transmitted from the T222 to the Pandora network card using a 20 Mbps INMOS OS-link. This occurs when the message handler decodes a time message from the GPS receiver unit. The time message contains a time in seconds since midnight, January 6th 1980. The accuracy is achieved using the microsecond timer of the T222, and subtracting the time between the GPS-synchronised interrupt and the message being transmitted over the INMOS

OS-link. This difference is equal to the T222's "microseconds" since the last GPS second boundary, the time specified in the message. The T222's clock drift must be accounted for by the Pandora network card. To help achieve this the same oscillator output is used to drive the serial card's T222 transputer and the Pandora network card's transputers.

6.2.2 Network card software

The software on the Pandora network card divides into three parts:

- Serial card interface

  The serial card interface takes messages from the T222 transputer, records the time of their arrival, and forwards them to the logical clock model. Its job is similar to the low levels of the clock message reception software in clients of the time server. The clock messages passed to the logical clock model contain the global clock value in seconds. The clock message distribution time is given as the number of microseconds since the interrupt on the T222 transputer. This method (compared to the alternative of specifying the global clock time in seconds and microseconds) allows the standard logical clock model to be used, as it can then adjust the message distribution time with the clock drift of the processor, rather than assuming the microsecond count is accurate.

- Logical clock model

  The logical clock model implementation in the time server is similar to that described above in section 6.1, but with the clock message filter turned off: in the time server the message distribution times are considered to be accurate local clock times (there is not, after all, a complicated message distribution system like a network).

- Network interface

  The network interface contains specific code for the network handled, as well as standard MSNL connection management. The time server maintains a SAP to which clients may connect. A client sends a request for a time to the time server, which is timestamped appropriately (in a network-specific fashion, see the following sections), filled with the global time, timestamped again, and then returned to the client.

  The time server is restricted in the number of clients it can support, so that the probability of two clients sending requests simultaneously is kept very small.

6.3 Clock distribution over the CFR

The global clock implementation using the CFR network requires that a network card which wants the global time become a time client of the CFR-based time server. It does this by making an MSNL connection to the SAP that the time server maintains. It then sends single packet requests to the time server at regular intervals. The time server responds to each of these requests with two single-packet messages, as suggested by 5.2.3.5. Information from the timing of those messages and the contents of them is passed to the logical clock model of the time client. In detail the communication runs as follows:

The time client transmits a single packet request to the time server, and records the time that its CFR CMOS chip signals the successful transmission of the packet (time *ClientTxed*).

The slave transputer of the time server is waiting for an interrupt from its CFR CMOS chip. When the packet is received by the slave transputer's ring interface the CFR CMOS chip will generate a packet received interrupt. The local clock value of the interrupt is recorded (time *ServerRxed*). The packet is then read in to the slave, and if it is a time request message (rather than an MSNL PDU) the global clock value is requested from its master transputer. When the global clock value is received, the local clock value of the slave is recorded (time *ServerGlobal*). A response packet is then transmitted to the time client. The CFR CMOS chip then generates an interrupt to mark the succesful transmission of this packet, and the local clock value of this interrupt is also recorded (time *ServerTxed*). A second response packet is then sent to the time client containing all three local clock values and the global clock value previously received from the master transputer.

The time client records the local clock value of the packet received interrupt from its CFR CMOS chip for the first response packet (time *ClientRxed*). When it receives the second response packet it calculates an estimate for the message distribution time, and passes a clock message containing this and the global clock value specified in the second response packet to its logical clock model.

The message distribution time estimate is calculated in two stages. Firstly, the offset between the clocks of the client and server are calculated. If the time client is considered to be the data source, and the time server the echo server, in the experiments described in section 5.2, then the timestamp values enumerated in section 5.2.2.1 correspond to those local clock values mentioned above as follows: *ClientTxed*, timestamp 1; *ServerRxed*, timestamp 4; *ServerTxed*, timestamp 5; *ClientRxed*, timestamp 8. Then the two traces in the graph in figure 5.14 corresponds to the differences between *ServerTxed* and *ClientRxed* (upper trace), and *ServerRxed* and *ClientTxed* (lower trace). If these are averaged then the result is a good estimate for the difference between the local clocks of the time server and time client (accurate to approximately a ring rotation time). If this value is subtracted from *ServerGlobal* then the result is an estimate for the time client's local clock value at the point the global clock value was received by the time server's slave transputer (call it *ClientGlobal*). The message distribution time estimate is then *ClientRxed* minus *ClientGlobal*.

Extra inaccuracies in the message distribution time estimate may be incurred if the clock readings are inaccurate. This can occur if, for example, the server is asked to serve more than one client's message at one time, or the client does not respond to the CFR CMOS chip's interrupts immediately as it is executing another high priority task. The first case is eliminated by making the method unreliable — that is, there is no guarantee that a request to the server will generate a response: in particular, the server will abandon all simultaneous requests as it detects them. The second case can be paritally removed if the client only sends requests to the time server when it is idle for at least a small period of time, although packets might be received from other transmitters, and thus disrupt the handling of the server response packets, at the same time as the replies from the time server.

6.4 Clock distribution over the ATM switch network

The ATM switch network time server and the time client operate in a similar fashion to the CFR system. However, in the ATM switch network system the time server only responds with a single cell message containing the local clock values *ServerRxed*, *ServerGlobal* and *ServerTxed* and the global clock value.

The time client/time server system attempts to mimic the data source and echo server system described in section 5.1. Transmission times for cells *ClientTxed* are the local clock values of the time the cell is inserted into the transmit FIFOs. Reception times for cells *ClientRxed* are the local clock values at which the receive interrupt is detected.

The message distribution times are taken to be *ClientRxed* minus *ClientTxed*, minus the time taken in the time server, which is *ServerTxed* minus *ServerRxed*, plus the difference between *ServerTxed* and *ServerGlobal*. This assumes the cell transmission times from client to server equal that from server to client, and that the ATM cell FIFOs are empty except for the timing cells so that the local clock readings are accurate. As for the CFR system, the accuracy is improved by detecting two clients simultaneously attempting to access the server, and by the client only requesting time messages when its receive and transmit FIFOs are empty. The ATM switch network system is more prone to problems as the FIFOs are much longer than for the CFR. However, the clock filter suggested in section 5.1.2.6 will filter these messages out if they are not too frequent. The ATM switch network implementation attempts to only transmit requests when its FIFOs are empty by monitoring its block transmit queue. If it is empty, then the request is transmitted, else the requesting process backs off. After a number of attempts the requirement for an empty queue is relaxed to a small queue. This may relax even more if the transmit load is constantly heavy, so that time requests are not completely stopped. If the load occurs for only a short time then the filtering mechanism will cut out requests during the loaded periods. If the load is consistent then the filtering mechanism will adapt to those conditions.

6.5 Distributing time inside Pandora

The first stage of distributing the global time inside a Pandora system is to distribute the global time to the Pandora box over the network. The second stage is to take the global time from the network card to the server card. Finally the server can distribute the time to the other subsystems, which are not directly reachable from the network card. This section describes the additions to the Pandora box software which were implemented to perform the distribution in those stages.

6.5.1 Network card improvements

The Pandora network card software, in both CFR T4 card and ATM switch versions, had to be updated to include the time client software. This is performed in the master transputer software with an extra debugging application process, the *time manager*, the logical clock model implementation, and additional commands and messages to let the network card's host read the global clock value. The slave software versions for the different networks needed to be changed to provide the recording of the local clock values *ClientTxed* and *ClientRxed* for clock distribution packets on reception and transmission, for accurate message distribution time estimation. An extra slave command was specified to read these values for the last clock distribution packets. Also, so the slave can determine which packets are clock distribution packets, extra slave commands were added to inform it of the VCI's for reception and transmission of the clock distribution packets.

Only small changes in the slave software were required to implement the extra facilities required, as no extra processes were required. The rest of this section describes the changes to the master transputer software.

The *time manager* process is responsible for the following:

- MSNL clock distribution connection

  The MSNL connection between the Pandora box and the time server must be made by the client. In the case of the Pandora box, the *time manager* process is responsible for making the connection when global clock distribution is required, killing it if it is not required, and recovering from any network events which kill the connection.

  The MSNL address and SAP for the time server are set by a command to the master transputer software, and are stored in shared memory. The *time manager* will attempt to make a connection to the time server if the specified address is not zero. If the connection fails to make the *time manager* process will receive a timeout message from the *connection management* process via the *debug manager*. In this case it backs off for thirty seconds before rereading the address and SAP and retrying. Once the connection has been made the slave software is informed of the VCIs corresponding to the connection for the recording of local clock values for transmission and reception of time requests and responses.

  If the connection is ever killed by the network or the time server then the *time manager* process will attempt to remake the connection after the thirty second backoff period.

  At present the time connection is deemed to be always required, so the *time manager* process never kills the connection itself. However, future implementations may kill the connection once the compliance of the logical clock model reaches a satisfactory level, indicating small phase and frequency errors. The connection would then be remade after a certain time to realign the logical clock model with the time server.

- Time request generation

  Once the connection to the time server has been made, clock distribution requests are generated by the *time manager* at regular intervals of approximately one second. These appear to the system as normal buffers for transmission. If the *time manager* detects the network transmission system is not idle it will delay sending the message for a short time before retrying. If the system does not become idle after a number of retries the request will be generated anyway, as inaccurate time is better than no time at all.

- Time response reception handling

  When a response to a time message is received by the *time manager* the slave software is interrogated to find the local clock values corresponding to the times of transmission and reception of the time request message and response. From these the message distribution time estimate is calculated according to the network type, and then the logical clock model is given this and the global clock value from the response.

In addition to the *time manager* process, the *buffer to host* process needed to be enhanced to allow high accuracy time messages to be sent to the host. When the *buffer to host* process receives a time message, it starts to send the message to the host, reads the global time from the logical clock model, then finishes the message to the host with the global time value read. Using this system the host is guaranteed to be ready to receive the global time value when it is sent, as the buffer will block on outputting the start of the message until the host is ready to read the whole message. The host must then read its local clock value immediately after receiving the message from the buffer process, to get an accurate value corresponding to the global clock value in the message. The time messages are generated in response to time request commands from the host.

6.5.2 Server card improvements

The software for the server card required changes to handle the distribution of the global clock value to it from the network, and an extra processes to run the logical clock model and to help distribute its global clock values to the other subsystems.

The *network output process* in the server card software was changed to send time request commands to the network subsystem at regular intervals of approximately one second. The *network source* process, which reads messages from the network, was changed to read the server's local clock value after receiving any message from the network. If the message is a time message, then the server's logical clock model is sent a clock message with a fixed distribution time estimate. When a time message is received a Pandora response (section 2.1.2.4)is also generated, which travels up along the response path to just before it leaves the server transputer.

At this point the response arrives at a new process, which passes on most rerports unchanged, except for those with report types indicating a time message. These reports are changed to include the global clock value read from the logical clock model, and are then sent on. From the server the responses reach the capture card (see figure 2.3).

6.5.3 Other subsystem improvements

The next Pandora subsystem on the clock distribution path is the capture card. This extracts global time messages from the incoming response path, and passes them to the capture processor's logical clock model. Further distribution of time messages is performed using a dedicated channel to the audio subsystem. This is possible as the INMOS OS-link from the capture transputer to the audio transputer is normally only used in the audio to capture direction (for requests, see figure 2.3), and not from capture to audio. A new process sends messages down this channel at regular intervals of approximately one second, containing values read from the capture transputer's logical clock model.

The audio transputer takes the messages from the channel, records the local clock time of their arrival, and then passes them to its logical clock model. The audio subsystem could also forward them down a dedicated channel to the video mixer subsystem, as the INMOS OS-link from the audio transputer to the video mixer transputer is only used for the response path, in the video mixer to audio direction. However, in the experiments planned for the Pandora system the video mixer is a data sink only, and so generates no timestamps and no results, and it is therefore not necessary to distribute global time to it.

6.5.4 Summary of system changes

The changes to the Pandora system provide for distribution of global time to all the data sources in the Pandora box down a chain of messages. At each link of the chain inaccuracy can be introduced due to the time taken to transmit the distribution messages down the links, which may not be estimated with accuracy, and the transmission will not be free of jitter. The current system could be improved in both respects, although the constant message distribution time estimates used generate adequate results. It must be borne in mind that the data generation processes which require the timestamps do not require timestamps accurate to a microsecond, as the data they are timestamping is made up of quite long samples.

Overall, the Pandora system was changed as little as possible to implement the distribution of the global clock so as to maintain confidence in the software. If the whole Pandora box software were to be redesigned then a more accurate global clock distribution algorithm would be implemented.

Chapter 7

Results of Pandora Experiments

Using the Pandora network implementation (chapter 3) and the additonal clock distribution implementation (chapter 6), numerous performance tests of the Pandora systems became possible. This chapter splits the results of some of those perfomed into 3 main sections: clock distribution; MSNL connection management; intra- and inter-Pandora stream handling. The chapter closes with a summary of the results.

7.1 Clock distribution

This section describes some experiments carried out on the global clock distribution clients to evaluate the accuracy of the global clock distribution, and the results of those experiments. The global clock distribution system was fully implemented with the enhanced fixed point Fuzzball logical clock model, purely an implementation of the algorithms derived from the timing of echo packets and the clock modelling described in chapter 5: it was felt that it would provide a global clock to any networked Pandora system to an accuracy suitable for measuring the operation and performance of a Pandora box.

7.1.1 CFR

One of the main reasons for designing the Pandora ATM switch network card was to replace the CFR network interfaces for Pandora. This is because, even though the CFR had lasted beyond its expected lifetime, maintenance of its hardware had become too difficult to be worthwhile.

Unfortunately, the CFR was removed from service before final results could be taken for this research. It was felt that this was not very important to the research as the results for the ATM switch network provide similar results to those preliminary results that were acheived using the CFR.

7.1.2 ATM switch network

The ATM switch network implementation of the global clock distribution followed the experimentation detailed in section 6.4.

To evaluate the performance of the global clock distribution system a pair of Pandora ATM switch network cards, outside Pandora boxes, were used as time clients of a standard time server. In addition to the Pandora network software implementation, complete with the extra time client as described in section 6.5, an extra process was added. This process flashes an LED at the start of every global clock second. To perform this accurately it calculates the approximate time to the next global clock second, and waits until shortly before that time. At this point (approximately 4 milliseconds prior to the next global clock second) it uses the global clock to accurately calculate the value of the local clock at the start of the next global clock second. Using the transputer's timer-based scheduling, it waits until that local clock value. Upon being rescheduled it turns the LED on, waits one hundred microseconds, then turns the LED off.

An oscilloscope can be used to monitor the voltage of the appropriate leg of the LED: when on the voltage is low. By monitoring the LEDs from the two different Pandora network cards on the oscilloscope (triggering off one of them) traces similar to those in figures 7.1 and 7.3 can be seen. These traces show two aspects of the accuracy of the global clock distribution: general running; continuous accuracy at start-of-day. These are described in detail in the following sections.

7.1.2.1 Steady-state accuracy of clock distribution

The traces shown in figure 7.1 graph the accumulated traces over 60 seconds of the two LEDs. The oscilloscope is triggering off the signal shown in the upper trace, *C1*. The lower trace *C2* is from the other network card. The timebase for the traces is one microsecond per division. The lower trace drops from high to low at between 2 microseconds before and 3 microseconds after the upper trace. The two network cards are being served by the same time server, and are connected to the same switch and so the network distance from them to the time server is the same.

The traces shown in figure 7.2 graph the accumulated traces over 120 seconds of the two LEDs, but this time the two network cards are connected by different network routes, of different lengths, to the time server. (The network card with the upper trace is two ATM switches further from the time server than the

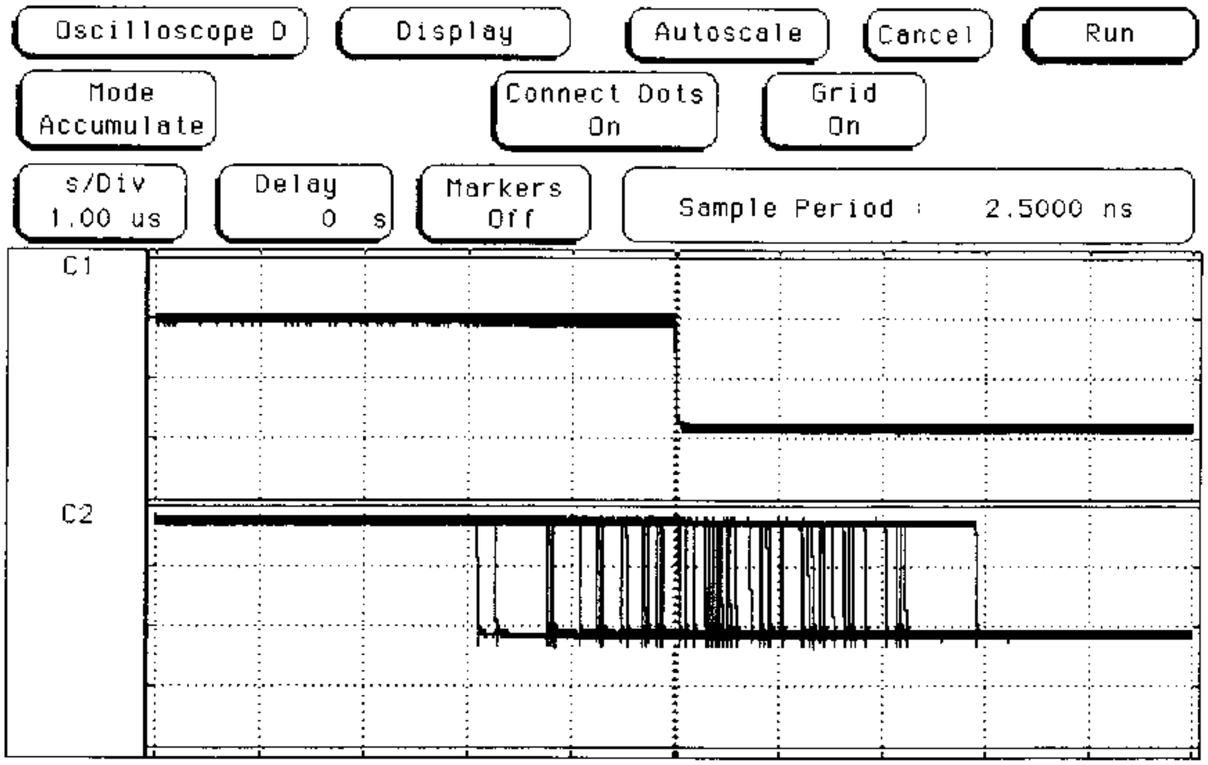
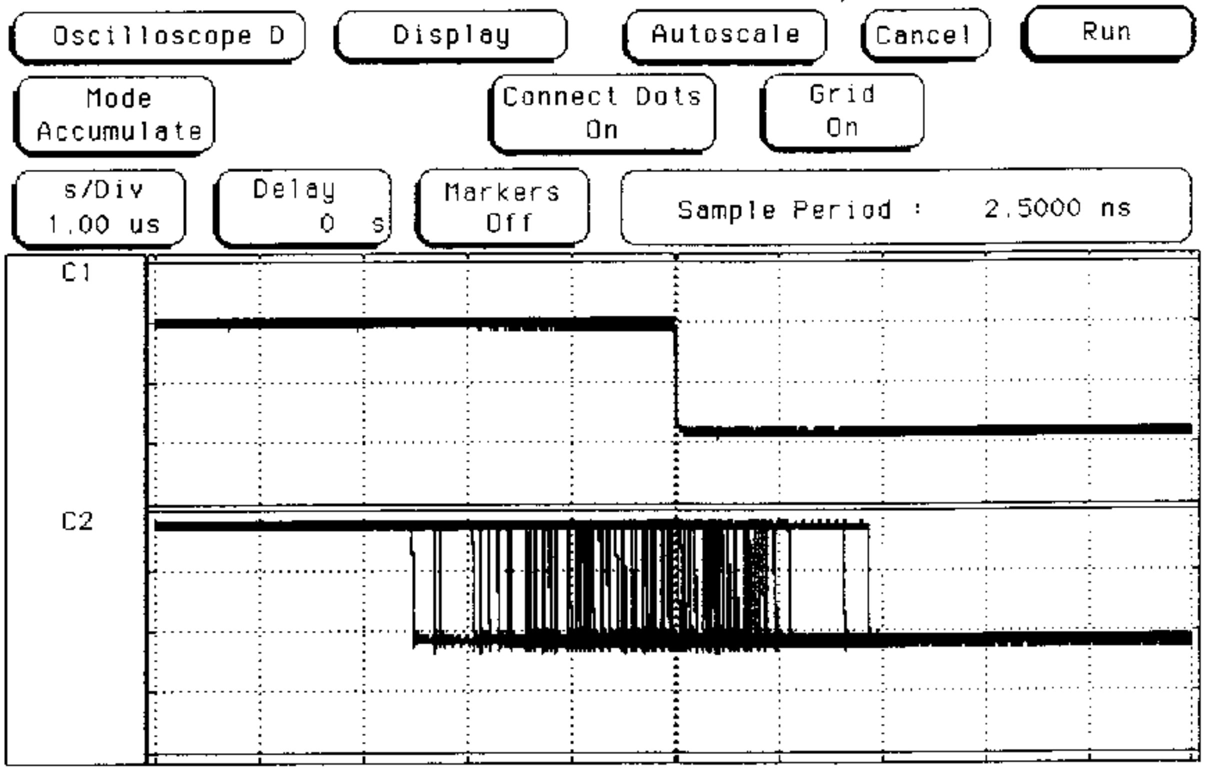


Figure 7.1: Two time clients with the same distance from time server



Mode Accumulate
Connect Dots On
Grid On
s/Div 1.00 us
Delay 0 s
Markers Off
Sample Period : 2.5000 ns

Figure 7.2: Two time clients with different distances from time server

network card with the lower trace.) The timebase is again one microsecond per division, and the separation between the two voltages falling varies from -3 to +2 microseconds.

In both cases the network cards had been connected to the time server for about 5 minutes prior to the measurements being taken.

The best possible results would be a difference of at most one microsecond between the two network cards (as this is the precision of the local clock). This is an unreasonable expectation, however, for a number of reasons:

- Accuracy of reading the local clock

  The accuracy of reading the local clock available to the test process is one microsecond, hence its scheduling times have an inaccuracy of at least one microsecond.

- Inaccuracy in global clock calculation

  The calculation of the next global clock second boundary in terms of the local clock can be at most one microsecond in error if an adjustment event is still to occur during the current second.

- Inaccuracies in the derived global clock

  The logical clock model is implemented in a manner to attempt to remove any errors in distribution due to inaccuracies in the message distribution time. However, this is not perfectly effective. There is also likely to be a small amount of error in the global clock derived by the time server and distributed to the clients (variations of about a microsecond occur for reasons similar to the previous two described).

More drastic effects may occur if the transputer fails to schedule the test process at the ideal time, which may occur if any other high priority (i.e. noninterruptable) process is already running at that time.

Taking the above reasons into account, it is felt that the fullest reasonable accuracy has been acheived.

7.1.2.2 Initial accuracy of clock distribution

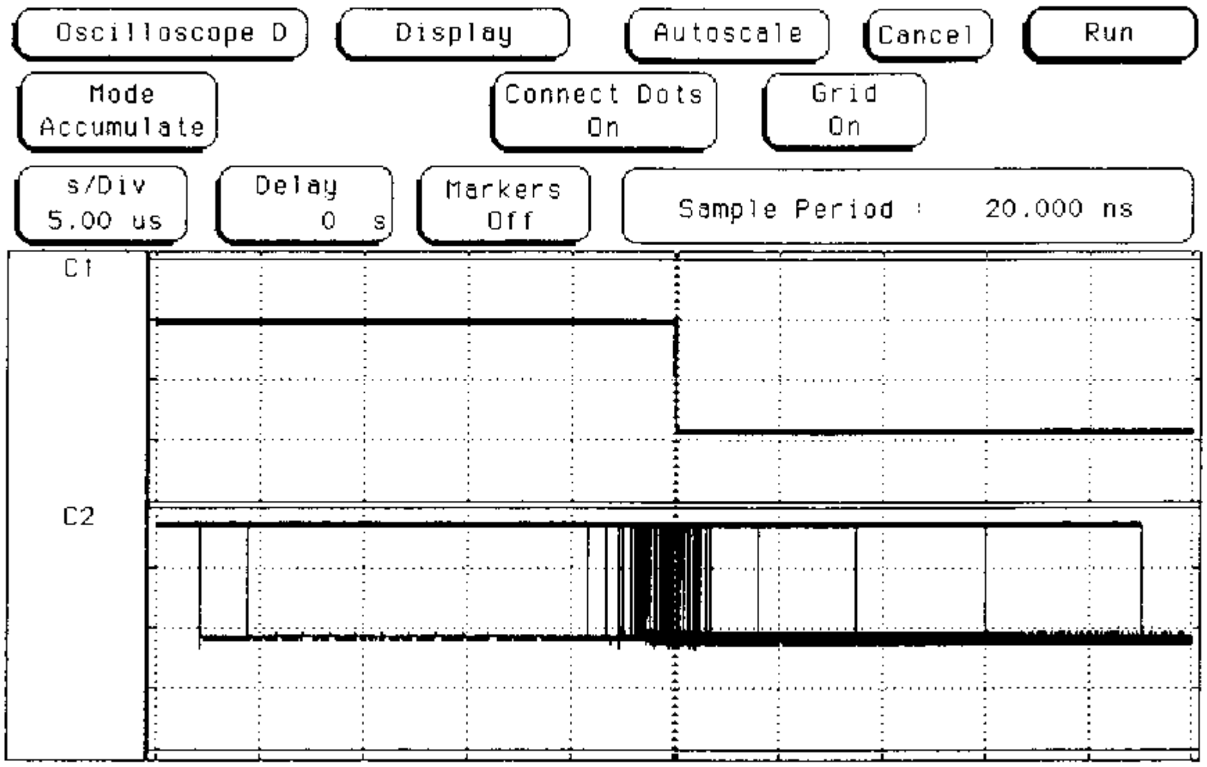
The oscilloscope trace shown in figure 7.3 provides a comparison of the two clocks during their initial start-up period. The trigger is the same as before, but the timescale is five microseconds per division. The test process does not start flashing the LEDs until ten seconds after initialisation of the network card, and this corresponds to about two seconds after the first clock message has been received by the logical clock model. The trace shows the first 60 seconds from the first test process LED flash.

The two network cards were reset by hand at the same time (within human hand accuracy). The first flash is shown at the left of the trace, about 23 microseconds between the two cards, the second flash has about 21 microseconds between them, then the third flash occurred near 0. The fourth flash lies further to the right, and as the clock models lock on to their frequency errors they approach the central portion of the trace.

The time for startup of the two clocks is within that predicted by the computer model in section 5.3.2, ($\pm$10 microseconds within a minute), remembering that that described a single client with an internal clock drift of -150 microseconds per second.

7.1.2.3 Reliability

If the time server were to fail, or if the network between the time client and time server were to fail, then the time client should maintain the best time possible. With the test system described above the network was unplugged from one of the cards, while the frequency error was not using the full precision available. The



Mode Accumulate

Connect Dots On

Grid On

s/Div 5.00 us

Delay 0 s

Markers Off

Sample Period : 20.000 ns

Figure 7.3: Initialisation of two time clients the same distance from time server

comparison of the two oscilloscope traces showed that in 30 seconds the clock had drifted by 4 microseconds.

7.1.2.4 Extractable information

The 16-character LED display attached to the slave transputer of the ATM network card can be used by the time client to display relevant information. This consists of the frequency error, frequency error divisor, compliance, phase error, offset detected, message distribution time estimate filter value, minimum message distribution time estimate, and an indication of the running time and percentage of messages unfiltered.

- Frequency error and divisor

  With time the logical clock model locks in accurately on the frequency error of the local clock. When the appropriate steady conditions have been achieved the frequency divisor is increased, to tighten the lock and increase the accuracy (see section 4.5.4.3). For this to occur the compliance must be small for 10 seconds, and so it only happens if the network is quiet enough for the clock message distribution time estimates to be accurate enough not to introduce large phase errors. When these conditions occur the largest divisor (and hence most accurate frequency error) is reached. This can be seen on the LED display when the frequency error is using all 32 bits.

  The frequency errors of the cards used for the oscilloscope traces were determined from the display. The card with the lower trace had a frequency error of -28.5 microseconds per second, the card with the upper trace has a frequency error of -164.9 microseconds per second.

- Output offset, phase error and compliance

  The output offset (microsecond difference between derived global clock and incoming clock messages) is a useful indication of how close the logical clock is to the global clock, and so gives an idea of the current accuracy. After a few minutes it settles to either 0, 1 or -1 microseconds. The phase error depends on the output offset of incoming messages, and so tends to reflect that. As the compliance is a long-term average of the phase error, it gives an idea of the recent output offsets, and so indicates whether the frequency error has not been locked onto. Once the logical clock has locked it varies from positive to negative quite frequently, as the phase error depends mainly on the errors in incoming clock message distribution time estimates. In most cases the phase error and compliance were found to vary from -2 to $+2$ microseconds.

- Message distribution times and filtering

> The LED display will show the minimum clock message distribution time estimate, and the current filter value. These give a rough idea of the loading of the network and network card, as well as an idea of the distance from the card to the time server. The percentage of messages filtered is not displayed directly, but must be calculated. The percentage of messages not filtered out due to large distribution time estimates varies from time to time according to network load, and is usually between 55 and 70 percent. These figures are for a network distance from time client to time server of 5 switches. Commonly the difference between the filter value and the minimum message distribution time estimate is at most 4 microseconds on these time clients.

7.2 ATM connection management

The ATM switch network provides fast cell routing for virtual circuits. One of the possibilities provided by the global clock distribution was to study the time required to make a virtual circuit between two network interfaces over the ATM switch network. The Pandora system was an ideal vehicle for this performance measurement because it sets up virtual circuits for its data streams.

7.2.1 MSNL event and data transport timestamp recording

To provide facilities for timing the connection management of the ATM switch network the Pandora network software was upgraded to record the global times of transmission and reception of MSNL PDU's, and also to record the global times of the transmission and reception of blocks of data. These records are transmitted ten times a second by a new debugger application, the *record manager* process, on a record logging connection, if that connection has been made. This allows a Unix-based DecStation to log the records for transmission and reception of both MSNL PDU's and data blocks.

7.2.2 Virtual circuit creation

The graph in figure 7.4 shows a timeline for the events which occurred for an *xvlook* from a Pandora system named *melon* at a Pandora system named *grape.* An *xvlook* lets a user of one Pandora system see through the camera of another Pandora system, and the user of that system sees a small picture from the camera of the instigator's Pandora system. As can be seen from the timeline, the *xvlook* was only run for about 5 seconds. The events which occurred between 196000

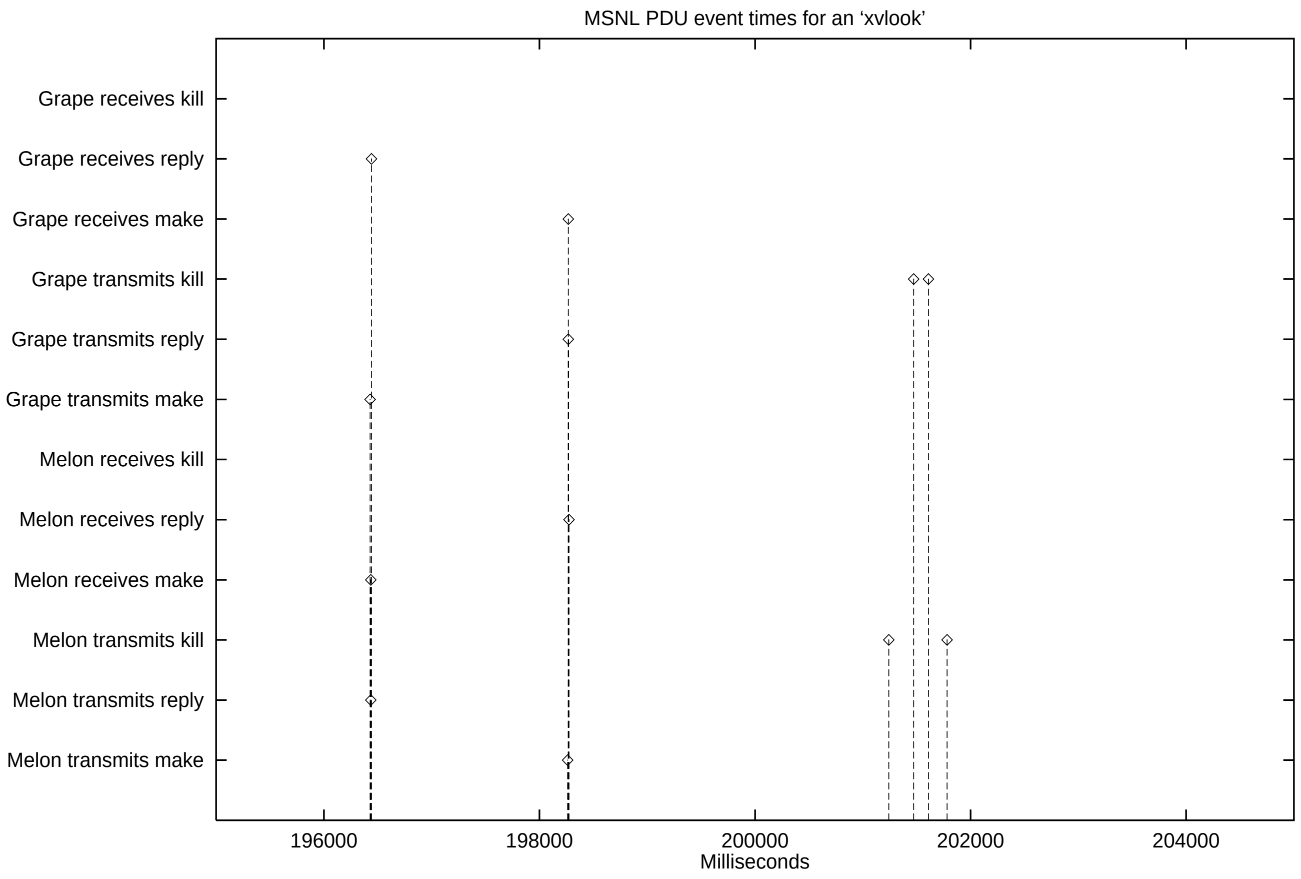


Figure 7.4: Timelines of connection management for an *xvlook*

and 198000 milliseconds are for the video data connection from *grape* to *melon*, and are magnified in the left-hand half of figure 7.5. The events which occurred between 198000 and 200000 milliseconds are for the video data connection the other way, and are shown in the right-hand half if figure 7.5. The events between 200000 and 202000 are due to the closing down of the *xvlook*.

From the left-hand graph of figure 7.5 the time taken to make the virtual circuit can be calculated. With approximate times, the events occur as follows:

1. At 196429 milliseconds *grape* initiates a connection by sending an MSNL connection make PDU on its ATM port, which arrives at its local switch.

2. The switch knows the route to the destination specified in the received MSNL PDU (in this case connections had already been recently made, so the route through the switch was cached), and forwards the MSNL PDU along that route.

3. After passing through another 4 switches, the MSNL connection make PDU

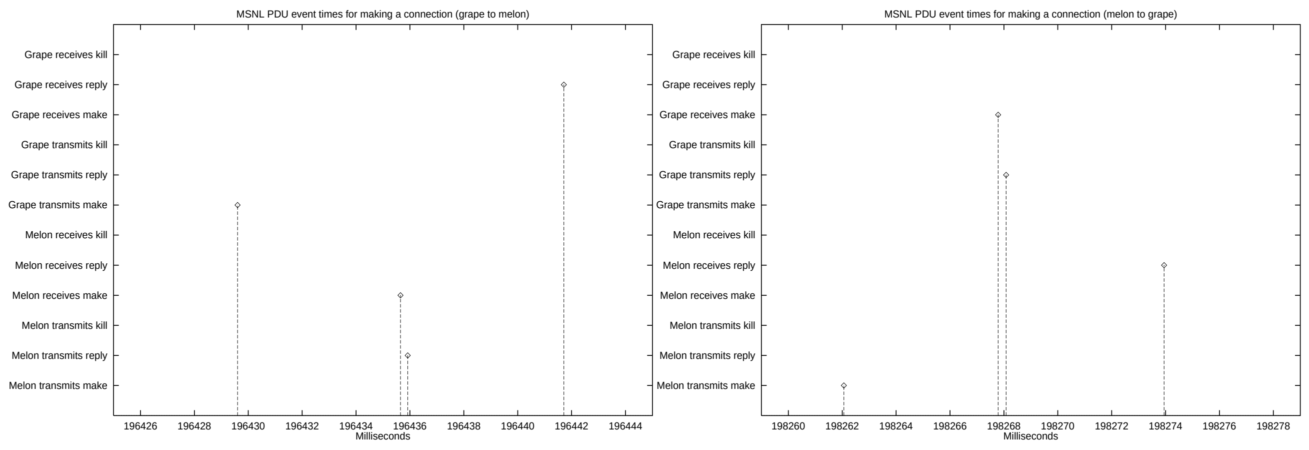


Figure 7.5: Magnified timelines for connection creation

arrives at *melon* after 196435. This message has taken 6.047 milliseconds to travel from *grape* to *melon*.

4. The Pandora box *melon* checks the connection is expected by the server card, and then responds by sending an MSNL connection reply PDU to its switch just before 196436. This decision and transmission took 0.268 milliseconds.

5. After travelling along the return path through the five switches, the MSNL connection reply PDU arrives at *grape* before 196442 milliseconds, and the virtual circuit is complete. The total time taken is 12.110 milliseconds.

The right-hand graph, with the expanded timeline of the making of the return connection, follows the same outline with the two machines reversed. This virtual circuit took 11.882 milliseconds to make.

Further experimentation yields an average connection make time of 11.7 milliseconds, of which an average 277 microseconds are taken by the Pandora system to which the connection is being made. Each switch in the route between the two machines takes 2.3 milliseconds, which splits evenly between the outward and return journeys.

7.2.3 Virtual circuit destruction

A look at the timings for the closing of the connections for the *xvlook* is also enlightening. It would appear that at time 201241 *melon* kills one of its connections to *grape*, followed at 201471 *grape* kills its end of that connection. These actions are both prompted by the Pandora interface, which was in turn commanded by the workstation to kill the network video connection. An odd feature of this timeline is that both Pandora boxes should transmit the MSNL connection kill PDU; it would be expected that the first one transmitted by *melon* would propagate through the switches to *grape*. It was possible that the propagation delay for each switch was large enough that the MSNL PDU has not propagated to *grape* within 230 millisconds, and so *grape* killed the connection from its end. The same is true for the other connection, with a gap of 172 milliseconds between the two ends transmitting the MSNL connection kill PDUs. However, when the large time (230 milliseconds) without successful propagation of the kill PDUs was discovered, further investigations were made: the actual reason for the failure of the propagation in the time given is a misunderstanding of the nature of the connection kill PDUs by the switches, which leads to them being discarded as invalid, and hence not propagated by the switches. (With this problem was solved, the propagation time becomes (as for connection make and reply PDUs) 11 milliseconds, and both ends do not transmit MSNL kill PDUs.)

7.3 Pandora system

In addition to the implementation of the distribution of the global clock inside a Pandora system, described in section 6.5, extra facilities were added to the server to allow the measurement of the performance of the Pandora system on its own and with other Pandora systems through the network. These are described in the first section below.

With the capability of the full system there are very many experiments which may be performed on single and multiple Pandora boxes, to study single stream behaviour, stream interaction, network tranport, and so on. To completely cover this area is beyond the scope of the research for this thesis. However, a reasonable number of experiments were performed to examine simple Pandora box stream behaviour and networking behaviour, to demonstrate the power of the distributed global clock as a performance evaluation aid, and to investigate the network principles in action. The remaining sections describe those experiments.

7.3.1 Segment and data timestamp recording

As described in section 2.2.5, the server uses a central segment buffer allocator for all the segments which it handles. This segment allocator was enhanced to record the global time of each segment buffer allocation, the duration of allocation, the process to which it was allocated, the process which deallocated it, and the length, segment sequence number and Pandora timestamp of the segment stored in the buffer at deallocation. By updating the network sink process, these records can be transmitted over the network every quarter of a second. Coupled to the record system added to the network software, described above (section 7.2.1), this system allows a log to be made by a Unix-based DecStation of the path of data from source, through the network, to server sink process of data in any stream of the Pandora systems, all using global timestamps.

The data source handling processes in the server transputer (see section 2.2.5) allocate buffers from the central buffer area at initialisation, so that any data which they may receive from a source may be read directly into the buffer area. The buffers are freed by the sink processes. As described above, the global time of allocation and duration of allocation are now recorded. To extract useful information from these times the system must be well understood:

- Time of allocation

  A buffer is allocated to a process just after the previous buffer allocated to that process has been passed to the switching element of the server, therefore this time is approximately equivalent to the time the previous segment reached the server switch process.

- Time of deallocation

  A buffer is deallocated once all the handshaking sink processes using the buffer have succeeded in passing the data contained within it to the sink. The deallocation time is therefore the time at which the data is fully accepted by the data sink.

- Other times

  The time the buffer is actually used by the source process, and the time the buffer passes from the server switch process to the sink process, are not deducible. However, the global timestamp of the generation of the data stored in the buffer gives an idea of the initial use time.

7.3.2 Data source timestamps

The introduction of the distributed global clock to Pandora systems increased the accuracy of timestamps in the Pandora segments from 64 microseconds to 1 microsecond. This allows a finer examination of the data sources than was previously possible. Both audio and video are examined here.

7.3.2.1 Audio data generation

As described in section 2.2.2, the audio capture is performed using an 8kHz CODEC, which produces one sample of audio data every 125 microseconds. This data is collected in a FIFO until 32 interrupts from the CODEC have occurred,

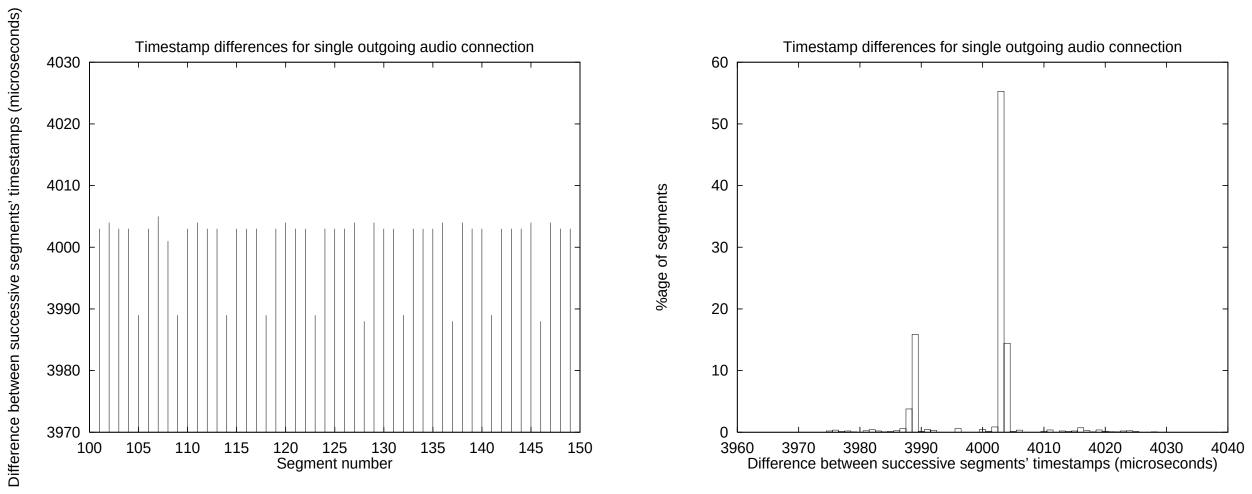


Figure 7.6: Timestamp differences between segments for outgoing audio stream

indicating the FIFO contains 32 samples, 4 milliseconds of data. At this point the audio transputer generates a timestamp and extracts the data from the FIFO, forming a Pandora audio segment, which then travels to the server.

Figure 7.6 shows results from an experiment using a single Pandora audio stream. The *xvmedia* application was run to create audio source on a Pandora box *grape*, and route the data through the ATM switch network to another Pandora box *melon*. From the above description, it may be assumed that the timestamps from successive Pandora audio segments will differ by approximately 4 milliseconds, the length of the segments.

The left-hand graph in figure 7.6 shows the times between successive audio segments, calculated from their timestamps, for a small number of audio segments (50 segments, which corresponds to 200 milliseconds). The right-hand graph shows a histogram of the differences between successive timestamps for all the segments in the audio stream, from creation to deletion of the stream. As expected, the distribution is centred on about 4 milliseconds. However, the distribution clearly splits into two spikes, one at 4003 microseconds (70% of the segments) and a smaller one at 3989 microseconds (20% of the segments). However, the average of all the inter-segment times for the stream is 4000.1 microseconds. A look at the left-hand graph shows why the two spikes occur. It is clear that about one in 4 or 5 segments is timestamped at about 3989 microseconds after the previous segment, and the others around 4003 microseconds. Now the audio transputer is idle except when handling the CODEC interrupts. There is no time delay in the code for handling these interrupts, and the timestamping should occur at a consistent time after the preceeding interrupt. This leads to the belief that the CODEC itself introduces the unexpected distribution.

Figure 7.7 shows similar graphs to 7.6, except in these graphs the *xvlocal* application was used to create a local audio stream. The data is sourced by the audio

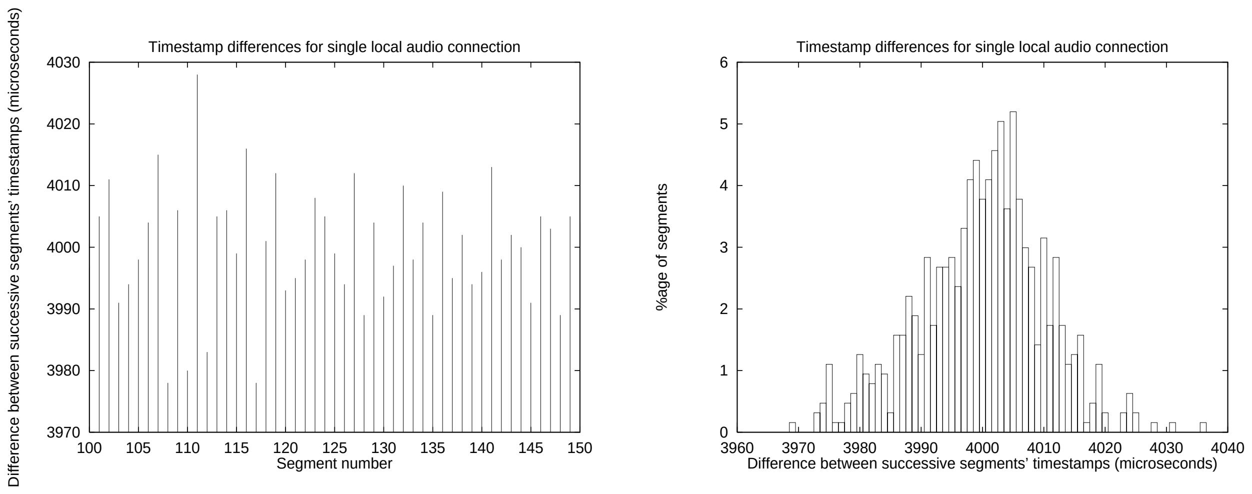


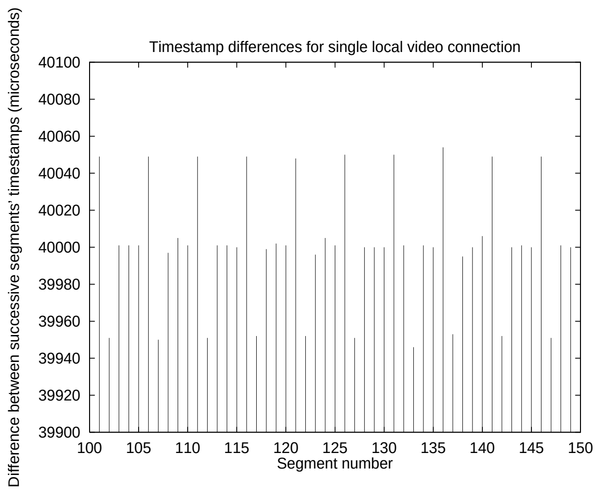
Figure 7.7: Timestamp differences between segments for local audio stream

card in the Pandora box *melon*, sent to the server card, and returned to the audio card for replay through the CODEC. As can be seen from the right-hand graph, the distribution is not as clean as before. The average of the whole stream's inter-segment timestamp differences is now 4000.2 microseconds (note that this figure is for audio captured on *melon*, whereas the previous figure is for audio captured on *grape*), but the spikes in the distribution have been smoothed as the audio processor must now cope with muting the captured data, outputting the data from the server as well as capturing the data as performed above. The server processor will introduce jitter in the stream as well, as it must handle other tasks as well as returning the audio data it receives from the audio processor. The left-hand graph exhibits none of the patterning which could be seen in figure 7.6, as this is probably all masked by the jitter.

It is worth noting that the data plotted in the graphs here are not independent. If one segment is timestamped late between segments timestamped as normal, the differences plotted will be first smaller then larger than the mean. This will tend to introduce a symmetric distribution of inter-segment timestamp differences, and alternating low and high impulses on the left-hand graph of the figures.

7.3.2.2 Video data generation

Figure 7.8 shows similar results to those described in the previous section on audio data generation, except this time for video data generation. A small local video source was created using *xvlocal*. This captures video frames at 25Hz into single 4 kilobyte Pandora video segments, each of which is timestamped prior to the data for the segment being fed to the compression unit. The right-hand graph shows the distribution of inter-segment timestamp differences, and the left-hand graph shows the differences for particular segments.



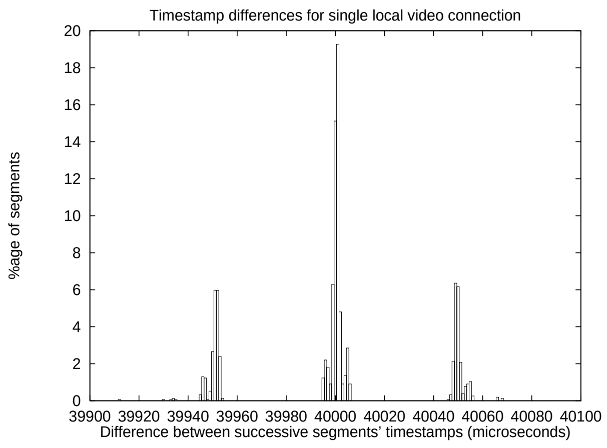


Figure 7.8: Timestamp differences between segments for small local video stream

The strange distribution is unexpected. The right-hand graph clearly shows a proportion of segments being timestamped 48 microseconds earlier or later than expected. From the left-hand graph it can be seen that these are actually timestamped late: with the following segment being timestamped on time, this gives the symmetric distribution shown in the right-hand graph. Occasionally (e.g. segments 131 and 132) there will be two late segments in a row, shown by a 40048 microsecond spike followed by a 40000 microsecond spike, followed by a 39952 microsecond spike, in the left-hand graph.

With a video frame consisting of two Pandora segments the difference between timestamps of first segments of successive frames are distributed as shown in figure 7.9. This distribution again has symmetry, but with spikes at 40016, 40048, 40064 and 40080 microseconds.

No definite explanations has been brought forward to cover these distributions. However, there is a theory. The task of managing a segment prior to timestamping consists of: the capture video synchronistaion process (section 2.2.3) counts horizontal synchronisation pulses, every 64 microseconds; on reaching the end of a segment the control process is sent a message indicating the whole segment is in VRAM; after a small amount of handling the control process sends a message to the FIFO handler process indicating the stream's segment is ready; the FIFO handler process then gets the global timestamp.

Now, the transputer itself is responsible for the order of execution of the processes: once a message has been sent, if the recipient immediately accepts the message, then either process may be scheduled. In the case of the video capture software, the video synchronisation process runs at high priority, and so it will continue to execute after sending the message to the control process; however, the other processes are both low priority.

So, it is possible that in some circumstances the FIFO hander executes as soon as

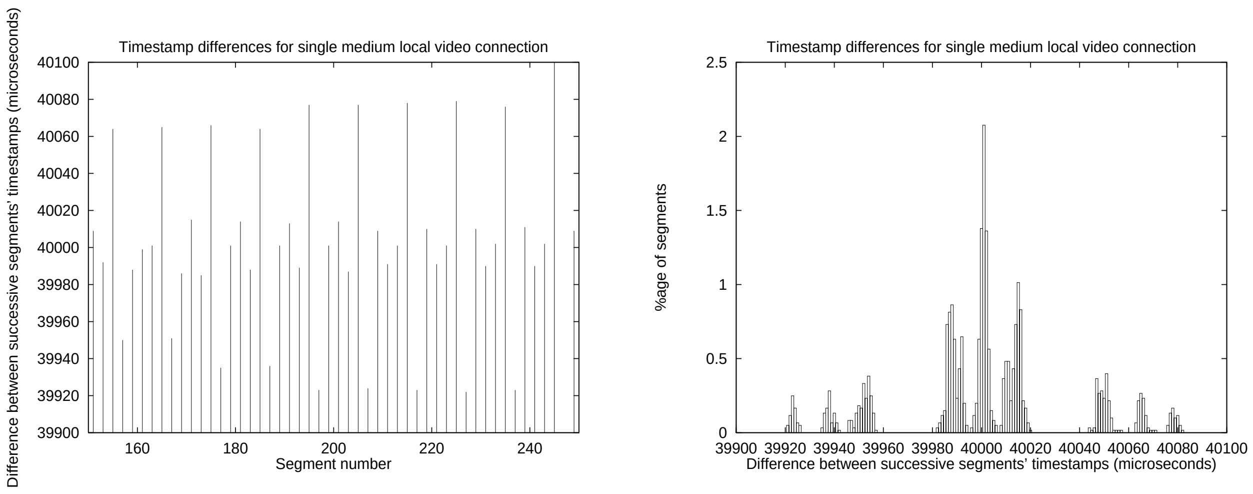


Figure 7.9: Timestamp differences between segments for medium local video stream

it receives the message from the control process; at other times the control process continues to execute until it blocks before the FIFO handler is scheduled. With single segment video streams, where the control process must cope with the end of the first segment being the end of the capture for that vertical synchronisation, this could take 48 microseconds. With the larger video streams consisting of two segments the control process must do less work after sending the message to the FIFO process before blocking, perhaps taking 16 microseconds.

Now, this can account for the spikes, but without explaining the actual scheduling for the different segments, particularly the almost constant pattern of one delayed segment followed by four undelayed segments. Also unexplained is the additional patterning produced in the case of two segment video, where 64 microseconds extra delay is possible. By looking at the right-hand graph in figure 7.9 the two patterns, it can be seen that the two effects are not independent. An explanation for the additional 64 microsecond patterning is the video synchronisation process missing horizontal synchronisation interrupts, a real possibility.

Unlike the audio data timestamping, where the distributions of inter-segment timestamp differences depend on incoming streams as well as outgoing streams, the video data timestamping is not affected by any video streams being replayed by the Pandora box, as the video capture card is not involved in video replay.

7.3.3 Server data switching times

Having examined the Pandora data generation, which does not require global clock distribution, an examination of the timing of Pandora data communication becomes reasonable. The second stage that any Pandora data passes through after the generation stage is the server transputer, so the timing of data handling by the server transputer is the obvious next step.

Using the available global times the approximate time between the capture of a segment to its being passed to the server switching process can be calculated (*source-to-server time*), as the difference between the allocation time of a buffer and the capture time of the segment stored in the previous buffer allocated to the appropriate source process. Also the approximate time that the data remains in the server after reaching the switch process (*switching time*) is the difference between the allocation time of a buffer to a source process (i.e. the time the previous buffer was passed to the server switch process) and the deallocation time of the previous buffer by the sink process.

7.3.3.1 Local audio streams

The graphs in figure 7.10 show the distribution of these two times for a local audio stream, and figure 7.12 shows the distribution for a local video stream. The right-hand graph of figure 7.10 shows that the *switching time* for the data to the sink is consistent at about 672 microseconds, whereas there is jitter in the *source-to-switch time*, transferring the data from the audio CODEC FIFOs to the server switch, shown in the left-hand graph. The most likely place for this to occur is inside the audio processing card, as there is no jitter in the *switching time* distribution indicating the server is very lightly loaded. A comparison of the *source-to-switch time* distribution with the right-hand graph in figure 7.7, which refers to the same experimental data, is useful. The jitter shown in the latter graph occurs prior to timestamping of the data, which in turn is prior to reading the data from the FIFOs. The jitter in the former occurs after reading the timestamp. If these were independent then the jitter at the audio sink would be a summation of the distributions, wider than both. However, if the jitter is introduced in both cases in the audio processing card, then the distributions are not independent. Now, the left-hand graph in figure 7.11 shows the distribution of the differences between the time the buffer for a segment is passed to the switching process and the time the previous segment was timestamped by the audio source. The width of this distribution is similar to that of figure 7.7; this shows that the source timestamping jitter and the jitter in the *source-to-switch* time distribtion are interdependent. The right-hand graph in figure 7.11 shows the distribution of times between deallocation of buffers by the server, a distribution closely related to that of the times between presentation of successive segments to the audio sink. This again shows a similar width of distribution, providing more support for the argument that the audio processor is introducing the jitter.

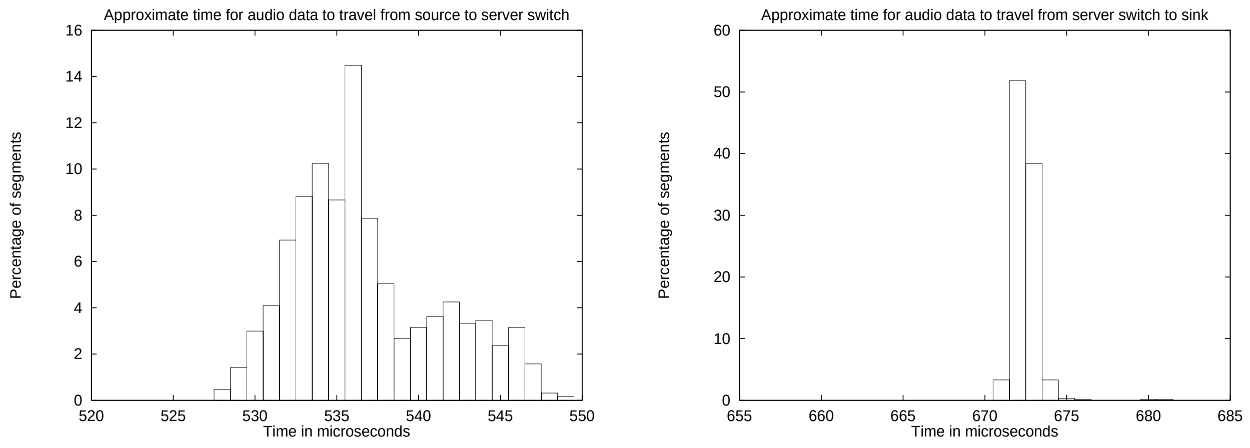


Figure 7.10: Distribution of transit times for a local audio stream inside Pandora

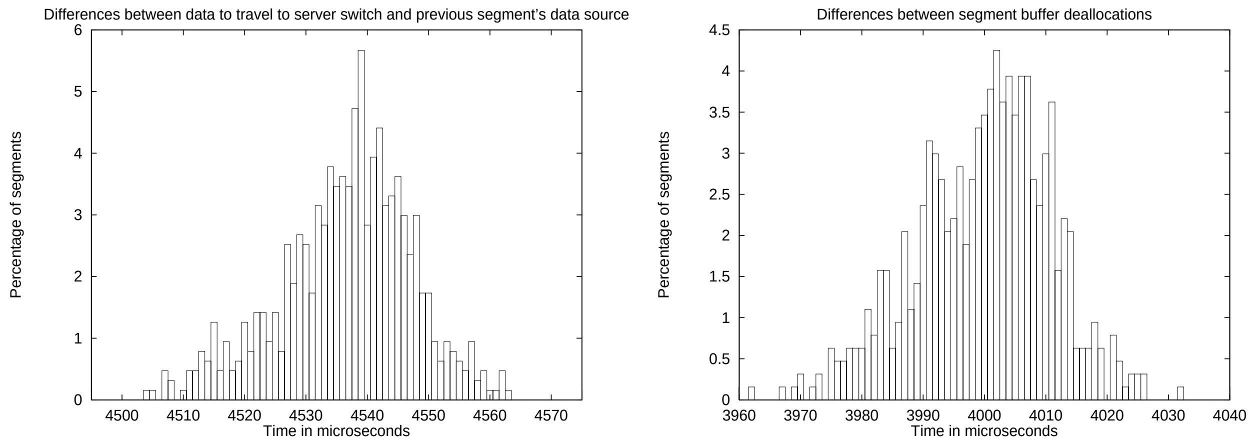


Figure 7.11: Internal time differences between segments for a local audio stream

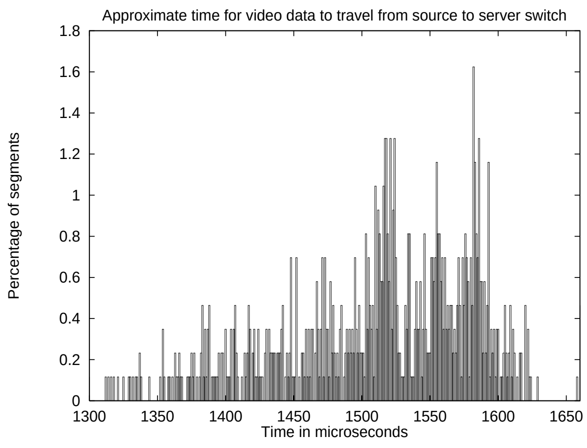
7.3.3.2 Local video streams

Comparing the jitter of the audio data stream to that of the video data stream is enlightening. The jitter in the audio stream's *source-to-switch* time distribution is about 20 microseconds (from the left-hand graph in figure 7.10), whereas the jitter in the video stream's *source-to-switch* time distribution is 300 microseconds (from the left-hand graph in figure 7.12). A look at the right-hand graph shows there is a great deal more jitter inside the server switch and video sink processes, almost up to 1.5 milliseconds. This can occur either because the server transputer is overloaded, or because the video capture and video mixer are both active and cause interaction. It is unlikely that the server transputer is overloaded, as the only data it is switching is a small video stream with 40 milliseconds between segments, a time much longer than the jitter. Much more plausible is the interaction between the server video sink process and the video mixer transputer processes which handshake lines of video through FIFOs and the decompression system, and similarly for the video capture transputer processes. Both of these processors have hardware to maintain, and the video mixer transputer processes must handle the video output which is not synchronised with the video input. Indeed, there are so many possibilities that the precise reasons for the jitter are not important. However, the size of jitter is worth recognising.

Another important feature of the video timing distributions is the difference between the means of the distributions. The video source and sink systems in Pandora both use FIFOs for communicating the video data from and to the compression and decompression hardware. The interprocessor communication structure is also identical. So why does the video *source-to-switch time* appear to be only between 1.3 and 1.65 milliseconds (left-hand graph), whereas the video *server switching time* is at least 6.6 milliseconds (right-hand graph)? The reason is this: the time differences used in the video *source-to-switch time* distribution are the differences between the timestamp of the video data and the time at which the next buffer is requested by the video source process on the server transputer; the video source process reads in the video data from the video capture card in parallel with requesting the next buffer. So the times shown in the left-hand graph do not refer to the transfer of the whole of the video data, but for all the control of the video; however, it does provide a fairly accurate picture of the jitter in the capturing of the data. Because of the manner of the time calculations, the data transfer time not included in the video *source-to-switch time* will appear on the *server switching time* for the data.

7.3.3.3 Network audio streams

Figure 7.13 shows the distribution of transit times for segments from a network audio stream from the server switch to the network (*source server switching*



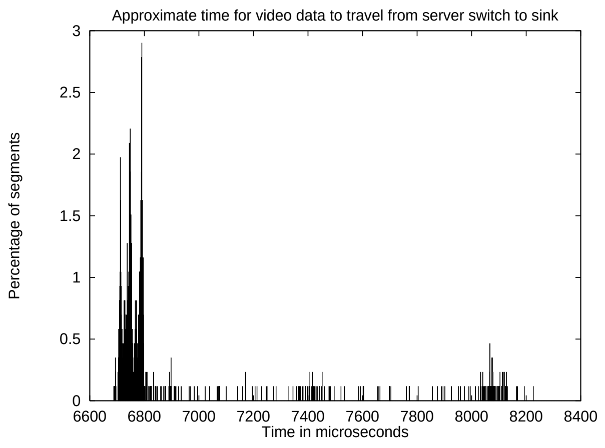
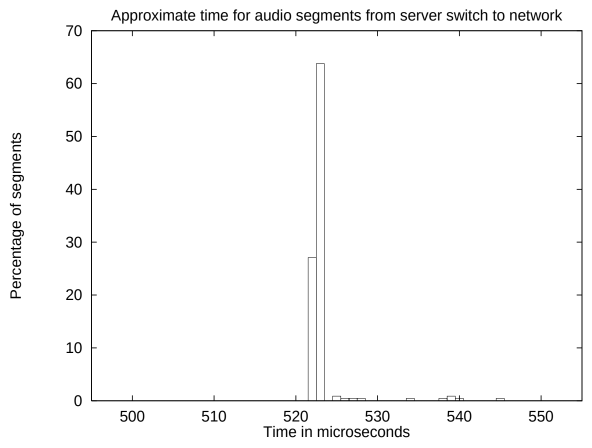


Figure 7.12: Distribution of transit times for a small local video stream inside Pandora

time), and that from the server switch to the audio sink at the other end of a network audio connection (*sink server switching time*). The second of these relates closely to the *server switching time* distribution shown in the right-hand graph show in figure 7.10. The same initial peak at 673 microseconds can be seen in both graphs, but the networked audio stream requires more management than the local audio stream, and this interferes with the distribution of the audio to the give the second peak at 684 microseconds in the right-hand graph of figure 7.13. A similar comparison can be made between the *source server switching time* distribution, the left-hand graph of this figure, and the right-hand graph of figure 7.10. Both distributions exhibit the single peak, but this is centred at 524 microseconds for switching the data to the network. This extra speed is gained because the network software is always ready for a complete segment of data, whereas the audio sink must be fed data in 16 sample portions. This requires extra management and twice as many communications with the audio sink as are needed by the network.

7.3.3.4 Network video streams

Similar graphs are shown in figure 7.14, but this time for a small video stream. Comparisons may be drawn between these and the right-hand graph in figure 7.12, as they all relate to *server switching times*. Both the right-hand graphs are for server switch to video mixer, but the graph in figure 7.14 refers to data coming from the network. The difference in the peaks of the distribution times between 4.2 milliseconds for the network video source and 6.7 milliseconds for the local video source is explained in section 7.3.3.2: the data transfer time from the video source is not all included in the *source-to-switch time*, but included in the *server switching time* from the video source. If it is assumed that video



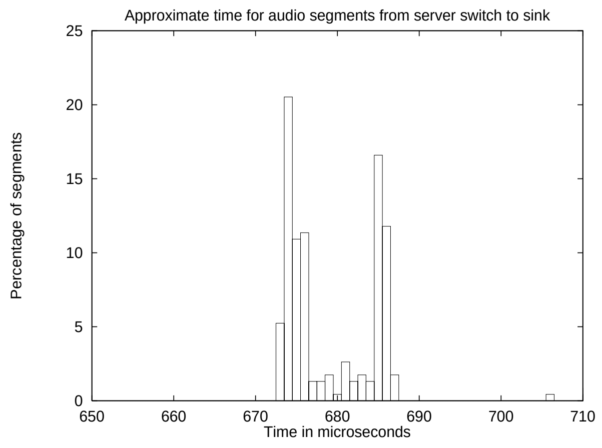


Figure 7.13: Distribution of transit times for a network audio stream inside Pandora

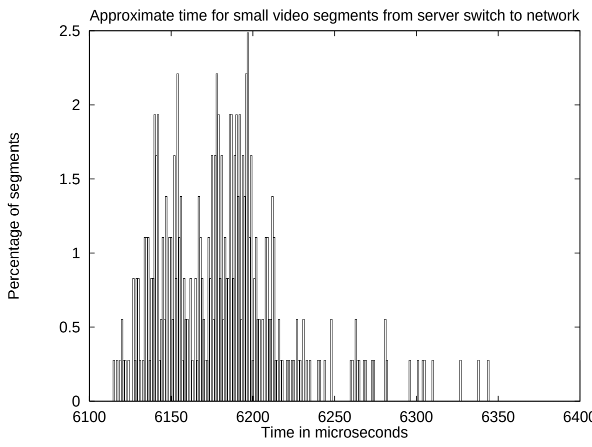
switching operations from both network to video sink and video source to video sink take the same mean time, and there is no reason not to assume this, then the means of the right-hand graph in figure 7.12 and that in figure 7.14 should be the same, excluding the mean data transfer time not included in the *source-to-switch time*. This implies the mean data transfer time is approximately $6.7 - 4.2 = 2.5$ milliseconds. This 2.5 milliseconds should be added to the left-hand graph of figure 7.12, and the times shown in the right-hand graph of that figure may be taken to be 2.5 milliseconds less than shown. This reduction also applies to the left-hand graph of figure 7.14.

A comparison of these last two graphs is intriguing. The local video sink appears to take about 500 microseconds longer to handle a segment of video than the network does. This is in spite of the extra hardware supplied to support the video sink, the FIFO's to the decompression hardware. This is probably because the protocol used over the INMOS OS-link between the server transputer and the video mixer transputer is slow enough to waste the advantage of the extra hardware.

7.3.4 Network transit times

The network performance is very important to the performance of a large multi-Pandora system. The graphs in figure 7.15 show the distribution of *network transit times* for audio streams (left-hand graph) and small video streams (right-hand graph), over 5 ATM switches.

The audio stream takes a fairly constant time per segment, around 250 microseconds. This compares with the 268 microseconds between connection make reception and connection reply transmission for the MSNL connection management



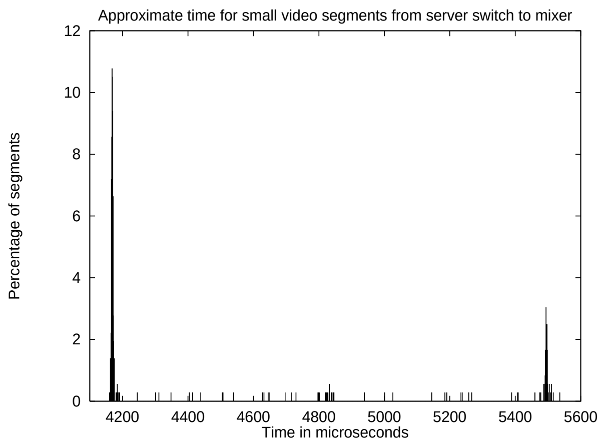


Figure 7.14: Distribution of transit times for a network video stream inside Pandora

(see section 7.2.2. Additionally, the time taken for a single cell to traverse the 5 ATM switches between the two Pandora boxes is about 64 microseconds (6 ATM links at 7.6 microseconds, 5 ATM switches at 3.6 microseconds, see section 5.1.2.3). This implies that the segmentation, reassembly and queue management in both Pandora boxes takes about 186 microseconds for the audio segments. The audio segments are two ATM cells long, as the generic header is 5 words long, the audio specific header 4 words, and the data is 32 8-bit samples. This corresponds to an instantaneous data rate of 2.9 Mbps. The video data has a transit time of around 1930 microseconds, small video segments are 1880 bytes long (40 ATM cells), which corresponds to an instantaneous data rate of 8.1 Mbps. This compares to the maximum transmission rate on the CFR of about 3 Mbps, and 4 Mbps aggregate reception and transmission data rates. Combining the transit time figures for audio and video segments, 250 microseconds for 2 cells and 1930 microseconds for 40 cells, leads to an estimate that 45 microseconds is required to handle a cell, 96 microseconds of overhead per block, with 64 microseconds network delay.

It is interesting to compare the *network transit times* for segments of data with the *source-to-switch times* and *server switching times* for similar segments. For audio data the former is, as stated above, about 250 microseconds, while the latter are 535 and 672 microseconds. Similarly, the video takes only around 1930 microseconds to transmit over the network, whereas the internal Pandora data transit times are more than twice that (around 4200 microseconds). So clearly (as there is no pipelinig of data inside Pandora) the network itself is not a throughput bottleneck, rather the switching, data generation or data replay limit the performance available.

The graphs do not show the full jitter which occurred during the experiments. The audio stream had a maximum transmission time of 355 microseconds, and

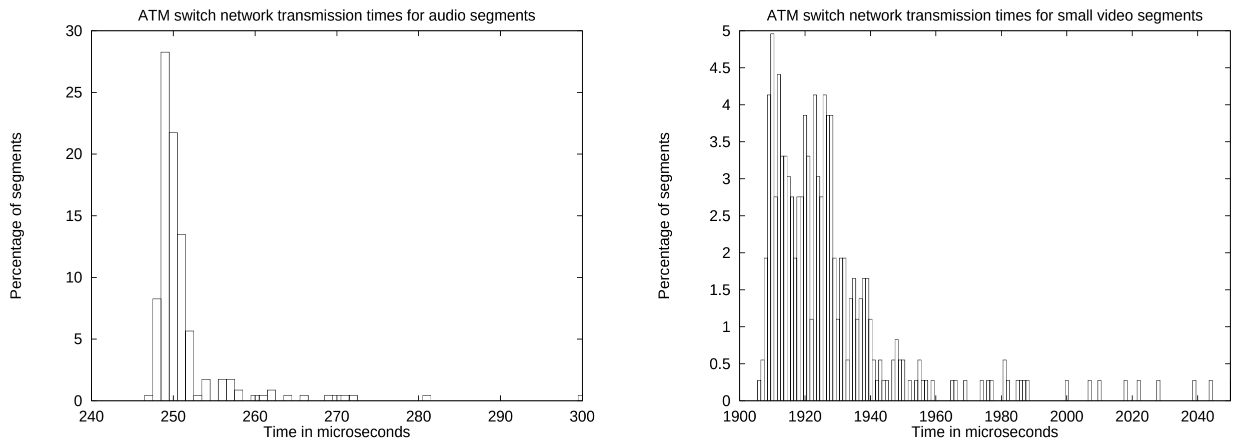


Figure 7.15: Distribution of network transit times for audio streams and small video streams

the video stream a maximum transmission time of 2105 microseconds.

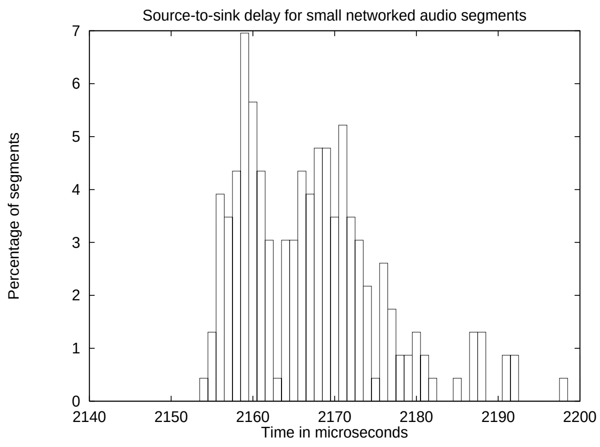
7.3.5 End-to-end delays

Figure 7.16 shows the distribution of the time taken between a Pandora data source being captured and it arriving at its data sink processor, having travelled through the ATM network in between. This provides the best picture of the jitter in the streams once they have travelled through the network. The left-hand graph shows the distribution for audio streams, the right-hand graph for small video streams. The left-hand graph is expected to be about $535 + 522 + 250 + 680 = 1987$ microseconds (adding the *source-to-switch*, two *server switching* and *network transit* times together); the difference of 170 microseconds is taken up by the time taken after network reception to delivering the segment to the server switch. For the video the calculation is $1500 + 6250 + 1930 + 4200 = 13880$, with about 1000 microseconds taken in transferring the data.

7.4 Summary

This chapter has provided details of experiments performed with the Pandora box implementation of the global clock distribution system described in chapter 6.

First the accuracy of the global clock distribution system was outlined, showing that it achieves relative errors of less than 2 microseconds. This provided a basis for examining the performance of connection management on the ATM switch network, where it was shown that making a connection though the ORL ATM



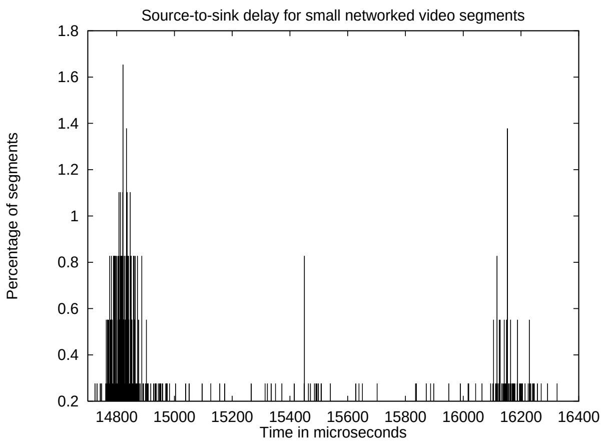


Figure 7.16: Source to sink delays for networked audio and small video streams

switches take about 2.3 milliseconds per hop.

Detailed results were then given of experiments with the Pandora system, where the interactions between the Pandora box processors and their hardware was examined to explain the distributions of times taken to perform certain stages of the transfer of data along the path of Pandora streams. From this it was deduced that the video and audio subsystems limit the amount of Pandora data that can be handled by a Pandora box, and that the network is no longer a performance bottleneck.

Other experimental work has been done on clock distribution before: [22] presents a master-slave clock synchronisation system with an accuracy of 10 milliseconds; [15] examines the true wide area with experiments on packet delays on trans-Atlantic links for clock synchronisation, where the accuracy of synchronization could be in the order of milliseconds. Also relevant to the experiments performed here is the examination of performance of the CFR Pandora system presented in [4], which utilised the 16-bit CFR network card.

Chapter 8

Conclusions

With the proliferation of high speed networks, and specifically the coming-of-age of ATM networks, real-time multimedia communication systems are becoming a reality. The experimental Pandora system is now over five years old, and the lessons which it can teach in this area need to be learned. This dissertation has exposed many of those lessons. In addition, it has shown that just as high speed networks enable the building of real-time multimedia communication systems, they also enable the accurate distribution of a global clock to allow detailed analysis of them. To fully implement such a clock more than just the network is involved: stable hardware counters, deterministic network driver software and precise clock sources are required. For precise clock sources, this dissertation has shown that the Navstar GPS system works very well in this regard, and the growing availability of GPS receiver units increases the availability of precise clock sources, to the extent of allowing any small LAN to have its own local clock source and clock distribution service.

8.1 Results of the work

In this dissertation the design of networking software for the Pandora distributed multimedia system has been addressed. An implementation of this has been detailed. Furthermore, a system for the accurate distribution of a precise global clock has been examined, experiments described to evaluate the expected performance if the system, and an implementation integrated with the Pandora system described. Finally, the combined system has been shown to be able to provide detailed performance measurements of a distributed networking system, a specific application of which is the Pandora system.

Overall the examination of the performance of the Pandora system using the full distributed global clock system highlights two areas. Firstly, the network principles described in section 2.3, to which the networking software for Pandora was designed to conform, are good foundations on which real-time networked multimedia can be built with confidence. Secondly, that access to a precise, accurate global clock at any point in a real-time distributed multimedia system is very useful if that system is to be fully understood, particulary because it is the complex interactions between the many parts of the system which must be studied. These two main areas are examined in more detail below.

8.1.1 Proposed network principles examined

This section looks at the proposed network principles thought to be required to be supported by a multimedia system's network subsystem (see section 2.3), the implementation of them in Pandora's network subsystem (see chapter 3), and the results of the experiments performed on Pandora (from chapter 7), to draw conclusions on their correctness, importance and usefulness.

8.1.1.1 Commands, messages and control

Principle 2.3.1 draws together the requirements for the control of the network subsystem of a Pandora box. The implementation, in particular the handling of messages for a debugging and remote access to the interface request/response system described in section 3.3.6.5, provides an invaluable path for both the management of experiments and collection of experimental data from a Pandora system while it is operating. This path was used for all experimental data collection in the experiments described in chapter 7.

8.1.1.2 Overload handling

The network support for the requirements for overload handling (principle 2.3.2) are hard to test, particularly on the ATM network card because, as described in section 7.4, the ATM network was not a performance bottleneck. However, the Pandora system is well regarded by users for gentle performance degradation under overload: the overload handling is shown to be good because of the lack of overload problems. Indeed, as noted in section 7.4, the video subsystem is the performance bottleneck, and its overload characteristics are a lowering of the frame-rate of the video, which generally passes unnoticed.

8.1.1.3 Data reliability

The requirements for data reliability (principle 2.3.4) are met by the system. Indeed, the system does not apparently lose any data, although this is helped by the underlying reliability of the CFR network (section 3.2), and the lossless nature of the ATM network when not running near overload. The push of ATM switch vendors and the standards body on ATM continues to be towards providing a lossless network, particularly for constant bit-rate streams, so the simplisitic implementation should continue to work.

8.1.1.4 Data buffering

The implementation of the data buffering, with unlimited receive buffering (see section 3.3.5.2), and with the T4 card the transmit buffer limiting (section 3.4), is supposed to fulfill principle 2.3.5. From the results in section 7.3, the former is not expected to have any visible effect, as the server card is never overloaded, and therefore will continue to sink the received data faster than it is received over the network.

The transmit buffer limiting, however, was introduced because the CFR network was slower than the video capture, and the buffer limiting did indeed remove the artifacts described in section 2.2.3.3. Once the ATM card was in use the video capture became the bottleneck (see section 7.4), so the buffer limiting remains out of operation.

8.1.1.5 Data bandwidth

Principle 2.3.6 requires that different bandwidth streams be supported, from under 100kbps to over 2Mbps. The implementation, with its emphasis on low per-block overheads and no copying of data, provides the required support, as is borne out by the results shown in section 7.3.4.

From the point of view of a user, with the CFR network subsystem the network is a bottleneck, and this always leads to questioning of the support for the highest possible bandwidth. Simple tests, such as changing the ring size, show that it is the CFR network itself which limits the bandwidth. Indeed, with the 100Mbps ATM network the Pandora network subsystem can sink all the data produced by the rest of the Pandora box, and still receive data from other Pandora systems without degradation: the 100Mbps ATM network subsystem is faster than Pandora internals, and the user's perception is that of seamless integration with other systems.

It is worth noting that the bandwidth required by Pandora's audio streams of 136 kbps is so small compared to the achieved network transit rate of about 3 Mbps for audio streams that it is unlikely any problems with the audio would ever be seen on the ATM network.

8.1.1.6 Data priorities

The support for principle 2.3.7 differs between the CFR network and ATM network subsystems, with the CFR network subsystem providing fine-grained prioritising of stream transmission (see section 3.3.4.3), while the 100Mbps ATM network's high bandwidth relieves the ATM network subsystem of those requirements. Both, however, try to balance transmission and reception by using equal priority transputer processes at the slave transputer level, while tilting that balance towards receive priority at the higher levels (see section 3.3.4.3).

Again, as the ATM network supplies such a high bandwidth connection, yet the Pandora internals provide less data than the network subsystem can manage, the prioritising is not visible in the ATM network subsystem. However, the CFR network subsystem shows the stream prioritising to good effect; when the network becomes overloaded, the audio streams operate well, with the video performance being degraded as lower priority.

8.1.1.7 Data latency

Principle 2.3.8 specifies strict constraints for latency of data transport. The support for these constraints include the balance in the slave software (see section 3.2) between buffering (for high bandwidth) and latency; the design requiring no copying of data (see section 3.3.2.3); latency control with selection of the priority of streams, section 3.3.4.3; a small overhead for each block, see section 3.3.6.5.

The final result seems to be good, as the latency is low (2ms) even for the high bandwidth video, and the perception from a user's view is instantaneous transfer. The figure in section 7.3.4 of 96 microseconds for the overhead of a block, with 45 microseconds per cell of a block, can be used to determine suitable block sizes for desired latencies, if necessary.

8.1.1.8 Data jitter

In principle 2.3.9 the jitter of a stream is required to be small: the real value depends on the stream type (video of significantly less than one inter-frame time, audio covered by downstream buffering and such that the latency requirement is not exceeded). The support for jitter control is generally implicit in the design rather than through particular aspects of the design, but there are some specific areas: section 3.2.3 identifies the immediate removal of receive data from the CFR chip FIFO, which is supported by the prioritising of the CFR slave software; the data transfer from master transputer to slave transputer multiplexes streams, see section 3.3.4.3; the reassembly process (section 3.3.5.1) also maintains the multiplexing of streams.

The outcome is that jitter in audio over ATM (section 7.3.4) is seen to be usually within 40 microseconds, although higher values are seen (up to 100 microseconds), all of which are less than a single sample time, and therefore will be no problem at all. Jitter in the video is larger (150 microseconds), with the worst recorded case of 200 microseconds: however, milliseconds of jitter are allowed.

8.1.2 Accurate distribution of a precise global clock

The widest conclusion that can be reached is that accurate access to a precise global clock is immeasurably useful for the performance evaluation of a distributed multimedia system. The task is not difficult, and the precision and accuracy acheived can be good with a simple implementation. ATM networks provide a very good distribution mechanism and GPS is a very good clock source. It is an ideal way to base an examination of the performance of any distributed system.

For Pandora, the clock distribution led to a much better understanding of how the system works, and also to uncovering performance-degrading bugs in the underlying networks. Without the accuracy of the clock distribution, the real effectiveness of the subsystems of Pandora would still be a mystery.

8.2 Scope of the conclusions

Although the network principles were derived from a detailed examination of the Pandora system, and were tested in a network subsystem for that system, it is believed that the network principles defined in this thesis extend beyond Pandora, into the network subsystems for any multimedia communication system. Some of the specific details are affected by the bandwidths of the data streams that Pandora requires; others by the nature of the control structures of Pandora, which may well not apply in other multimedia systems. Similarly, the work performed with global clock distribution has applications outside the Pandora world, wherever evaluation of real-time distributed systems is required.

8.3 Further work

One minor aim of this dissertation is to complete the documentation of the Pandora system, and the network subsystem work for Pandora is now over. The network principles underpinning the design need to be implemented in other systems, such as those using H.261 video coding ([38]) or MPEG-2 ([18]), and their performance examined further to test their validity. The precise, accurate global clock distribution work has only just started: this section draws attention to a few of the areas which need more detailed work.

8.3.1 Different physical layers

The ATM Forum has currently defined 6 different physical media for ATM transport supporting 10 different bit rates ranging from 1.544Mbps to 622.08Mbps [65]. Different ATM switches use different architectures. Much more work is needed to discover how effectively the global clock distribution can be performed outside the particular world that the current experimentation has used. Many questions, such as how SONET framing or low bandwidth on some links inside an ATM switch network effect the performance of the distribution, and what extra support for clock distribution could be implemented in switches or nodes themselves.

8.3.2 Inter-stream synchronisation

There are standards for constant bit rate and variable bit rate traffic streams on ATM networks. [19] and [16] present possible strategies for admission control, and through these delay guarantees, which may help inter-stream synchronisation. Both of these may reduce the need for global clock distribution, but at the same time the accuracy of the clock distribution could be improved by the same guarantees.

8.3.3 Applications of the global clock

The main reason for an examination of the possibility of supporting a global clock was to aid synchronisation of streams. This work still needs to be done. Many papers have been published on the problems of synchronisation, from the presentation level using scripts to control computerised slide-shows, down to the inter-stream synchronisation for real-time playback or communication of multimedia (e.g. [42], [61], [41]). Most solutions that are presented use some explicit form of data connection between a synchronisation subsystem and the data sinks. With a distributed global clock these synchronisation subsystems may be removed, and synchronisation performed by the sinks themselves. However, the sinks then require full knowledge of the global replay times of the various data blocks they will receive: in a sense, they require their own script. For real-time multimedia communications this is not a problem, but for presentation-level synchronisation it may not be the best solution.

Additionally, there are applications made possible by the accuracy and precision of the distributed clock that need to be examined. For example, the human ear is extremely sensitive, and to support very high quality audio the synchronisation between left and right speakers in stereo pairs needs to be very accurate. With more sophisticated audio systems becoming more widespread, with many more speakers, the synchronisation burdens increase. Is it possible to distribute the speakers on a network, rather than having a single audio node on the network into which all the speakers should be plugged?

8.3.4 Performance evaluation of commercial systems

There are now quite a few multimedia communication systems commercially available. The quality varies dramatically, from high bandwidth motion JPEG compressed images using up to 20Mbps data rates, to H.261 coded streams of 64kbps or Intel Indeo encoded video of 0.5Mbps. These systems are ideal subjects for performance evaluation, perhaps using the global clock distribution to timestamp appropriate ATM cell arrival times at switches.

8.4 Final summary

The results of this thesis can be summarised in three points. Firstly, simple precise, accurate global clock distribution is not difficult using ATM networks with a GPS clock source, and it is very useful. Secondly, real-time multimedia communication systems based on the principles presented in this thesis should work well, as Pandora does. Thirdly, complex real-time distributed systems do not necessarily work as intended, as the Pandora box, although fine-tuned for high bandwidth video data with FIFO's between source, switch and sink, does not perform as well as its designers believed it did: with the ATM network card the network itself has been shown to be no longer a bottleneck in the system, rather the video subsystem significantly underperforms.

Bibliography

[1] Acorn Computers Limited. *R140 Operations Guide*, December 1988.

[2] Acorn Computers Limited. *CFR Station Chip Datasheet*, November 1994. (Revision 1.1).

[3] Advanced Micro Devices. *AM7968/AM7969 TAXIchip Handbook*, 1994.

[4] C. S. Ang. *'Continuous Media in Fast Networks'*. PhD thesis, University of Cambridge Computer Laboratory, January 1992.

[5] ATM Forum. *ATM User-Network Interface Specification Version 3.0.* Prentice-Hall, 1993.

[6] R. Black. 'FDL Cell Format and Meta Signalling'. *ATM Document Collection 2 (The Orange Book, Systems Research Group Technical Note, University of Cambridge Computer Laboratory)*, February 1992.

[7] R. Black. 'MDL Cell Format and Meta Signalling'. *ATM Document Collection 2 (The Orange Book, Systems Research Group Technical Note, University of Cambridge Computer Laboratory)*, February 1992.

[8] R. Black. 'Common MSDL features'. *ATM Document Collection 2 (The Orange Book, Systems Research Group Technical Note, University of Cambridge Computer Laboratory)*, February 1993.

[9] G. Blair, A. Campbell, G. Coulson, F. Garcia, D. Hutchison, A. Scott, and D. Shepherd. 'A Network Interface Unit to Support Continuous Media'. *IEEE Journal on Selected Areas in Communications*, 11(2):264–275, February 1993.

[10] J. F. K. Buford. 'Architectures and Issues for Distributed Multimedia Systems'. In J. F. K. Buford, editor, *Multimedia Systems*, pages 45–64. ACM press books, Siggraph series, 1994.

[11] D. R. Cheriton. 'VMTP: A Transport Protocol for the Next Generation of Communication Systems'. In *Proceedings of Sigcomm '86*, pages 406–415, 1986.

[12] D. Clark, M. Lambert, and L. Zhang. 'NETBLT: A High Throughput Transport Protocol'. In *Proceedings of Sigcomm '87*, pages 353–359, 1987.

[13] D. J. Clarke. 'Pandora 1 ATM Network Card – Hardware Introduction and Reference Manual'. Olivetti Research Laboratory internal document, January 1991.

[14] D. J. Clarke and G. J. Stark. 'Network Cards for the Pandora Multimedia System'. Technical Report 94–5, Olivetti Research Laboratory, Trumpington Street, Cambridge, November 1994.

[15] R. Cole and C. Foxcroft. 'An Experiment in Clock Synchronisation'. *The Computer Journal*, 31(6):496–502, 1988.

[16] D. Ferrari and D. C. Verma. 'A Scheme for Real-Time Channel Establishment in Wide-Area Networks'. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[17] D. L. Gall. 'MPEG: A Video Compression Standard for Multimedia Applications'. *Communications of the ACM*, 34(4):46–58, April 1991.

[18] M. Goldman. 'MPEG over ATM'. *ATM Forum Newsletter*, 3(4), October 1995.

[19] S. J. Golestani. 'A Framing Strategy for Congestion Management'. *IEEE Journal on Selected Areas in Communications*, 9(7):1064–1077, September 1991.

[20] D. J. Greaves, D. Lioupis, and A. Hopper. 'The Cambridge Backbone Ring'. In *Proceedings of Infocom '90*, pages 8–14, 1990.

[21] D. J. Greaves and I. D. Wilson. 'Cambridge HSLAN protocol review'. Technical Report 89–1, Olivetti Research Laboratory, Trumpington Street, Cambridge, May 1989. Presented at IFIP WG6.1 Workshop, IBM Ruschlikon.

[22] R. Gusella and S. Zatti. 'The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD'. *IEEE Transactions on Software Engineering*, 15(7):847–853, July 1989.

[23] M. Hayter. *'A Workstation Architecture to Support Multimedia'*. PhD thesis, University of Cambridge Computer Laboratory, September 1993.

[24] M. Hayter. 'Fairisle / Yes / Maybe ATM Taxi Link Protocol'. *ATM Document Collection 2 (The Orange Book, Systems Research Group Technical Note, University of Cambridge Computer Laboratory)*, February 1993.

[25] C. A. R. Hoare. 'Communicating Sequential Processes'. *Communications of the ACM*, 21(8):666–677, August 1978.

[26] C. A. R. Hoare, editor. *occam 2 Reference Manual.* Prentice-Hall, 1988.

[27] A. Hopper. 'Pandora - An Experimental System for Multimedia Applications'. *ACM Operating Systems Review*, 24(2):19–34, April 1990.

[28] A. Hopper and R. M. Needham. 'The Cambridge Fast Ring Networking System'. *IEEE Transactions on Computers*, 37(10):1214–1223, October 1988.

[29] INMOS Limited. *Transputer Instruction Set, A compiler writer's guide*, 1988.

[30] INMOS Limited. *The Transputer Applications Notebook, System and Performance*, 1989.

[31] INMOS Limited. *The Transputer Databook*, 1989.

[32] IQD Limited. *Crystal Product Databook*, 1995. IQEXO-3 series, pp 184–185.

[33] A. Jones. 'Pandora Low-level Software'. Olivetti Research Laboratory internal document, April 1990.

[34] A. Jones and A. Hopper. 'Handling Audio and Video Streams in a Distributed Environment'. *14th ACM Symposium on Operating Systems Principles*, pages 231–243, December 1993.

[35] T. King. 'Pandora: An Experiment in Distributed Multimedia'. In *Proceedings of Eurographics '92*, 1992. Volume 11 Number 3.

[36] H. Kopetz and W. Ochsenreiter. 'Clock Synchronization in Distributed Real-Time Systems'. *IEEE Transactions on Computers*, 36(8):933–940, August 1987.

[37] L. Lamport. 'Time, Clocks, and the Ordering of Events in a Distributed System'. *Communications of the ACM*, 21(7):558–565, July 1978.

[38] M. Liou. 'Overview of the px64 kbit/s Video Coding Standard'. *Communications of the ACM*, 34(4):59–63, April 1991.

[39] T. D. C. Little. 'Time-based Media Representation and Delivery'. In J. F. K. Buford, editor, *Multimedia Systems*, pages 175–200. ACM press books, Siggraph series, Boston University, 1994.

[40] T. D. C. Little and A. Ghafoor. 'Network Considerations for Distributed Multimedia Object Composition and Communication'. *IEEE Network*, pages 32–49, November 1990.

[41] T. D. C. Little and A. Ghafoor. 'Synchronization and Storage Models for Multimedia Objects'. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[42] T. D. C. Little and A. Ghafoor. 'Multimedia Synchronization Protocols for Broadband Integrated Services'. *IEEE Journal on Selected Areas in Communications*, 9(9):1368–1381, December 1991.

[43] O. N. Mason, D. J. Clarke, T. H. Glauert, A. H. Jones, T. R. King, S. C. Wray, R. Want, and A. Hopper. 'A Network of Multimedia Workstations using the Pandora Multimedia Communications Processor'. Olivetti Research Laboratory internal document, August 1991.

[44] D. R. McAuley. *'Protocol Design for High Speed Networks'.* PhD thesis, University of Cambridge Computer Laboratory, September 1989.

[45] D. L. Mills. 'The Fuzzball'. In *Proceedings of Sigcomm '88*, pages 115–122, August 1988. Volume 18 Number 4.

[46] D. L. Mills. 'Network Time Protocol (verson 2) specification and implementation'. Technical report, DARPA Network Working Group Report RFC 1119, University of Delaware, September 1989.

[47] D. L. Mills. 'Internet Time Synchronization: The Network Time Protocol'. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[48] NATO. *Technical Characteristics of the Navstar GPS*, June 1991.

[49] P. Newman. 'A Fast Packet Switch for the Integrated Services Backbone Network'. *IEEE Journal on Selected Areas in Communications*, 6(9):1468–1479, December 1988.

[50] C. Nicolaou. 'An Architecture for Real-Time Multimedia Communication Systems'. *IEEE Journal on Selected Areas in Communications*, 8(3):391–400, April 1990.

[51] Y. Ofek. 'Generating a Fault Tolerant Global Clock in a High Speed Distributed System'. In *Proceedings of the Ninth International Conference on Distributed Computer Systems*, pages 218–226, June 1989.

[52] Philips Semiconductors. *80C51-Based 8-Bit Microcontrollers data handbook (IC20)*, 1993. I^{2}C specification, pages 136–154.

[53] Philips Semiconductors. *High-speed CMOS Logic family data handbook (IC06)*, 1996. 74HC7403.

[54] Project Unison. *CFR Chip Set Datasheets.* Document numbers UA013 and UA014.

[55] S. Ramanathan and P. V. Rangan. 'Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks'. *IEEE Transactions on Networking*, 1(2):246–259, April 1993.

[56] P. V. Rangan, H. M. Vin, and S. Ramanathan. 'Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences'. *IEEE Transactions on Networking*, 1(1):20–30, February 1993.

[57] S. Rangarajan and S. K. Tripathi. 'Efficient Synchronization of Clocks in a Distributed System'. In *Proceedings of the Real-Time Systems Symposium*, pages 22–31, December 1991.

[58] N. W. Rickert. 'Non byzantine clock synchronization - a programming experiment'. *ACM Operating Systems Review*, 22(1):73–78, January 1988.

[59] C. J. Sreenan. *'Synchronisation Services for Digital Continuous Media'.* PhD thesis, University of Cambridge Computer Laboratory, October 1992.

[60] T. K. Srikanth and S. Toueg. 'Optimal Clock Synchronization'. *Journal of the ACM*, 34(3):626–645, July 1987.

[61] R. Steinmetz. 'Synchronization Properties in Multimedia Systems'. *IEEE Journal on Selected Areas in Communications*, 8(3):401–412, April 1990.

[62] D. L. Tennenhouse. 'The Unison Data Link protocol specification'. The Unison Project, Document Reference UC022, October 1986.

[63] D. L. Tennenhouse. *'Site Interconnection and the Exchange Architecture'.* PhD thesis, University of Cambridge Computer Laboratory, September 1988.

[64] H. Tokuda. 'Operating System Support for Continuous Media Applications'. In J. F. K. Buford, editor, *Multimedia Systems*, pages 201–220. ACM press books, Siggraph series, Carnegie Mellon University and Keio University, 1994.

[65] R. Townsend. 'Physical Layer Specifications'. *ATM Forum Newsletter*, 3(3), July 1995.

[66] S. T. Treves, E. S. Hashem, B. A. Majmudar, K. Mitchell, and D. J. Michaud. 'Multimedia Communications in Medical Imaging'. *IEEE Journal on Selected Areas in Communications*, 10(7):1121–1133, September 1992.

[67] W. Verbiest, L. Pinnoo, and B. Voeten. 'The Impact of the ATM Concept on Video Coding'. *IEEE Journal on Selected Areas in Communications*, 6(9):1623–1632, December 1988.

[68] J. P. Vorstermans and A. P. de Vleeschouwer. 'Layered ATM Systems and Architectural Concepts for Subscribers' Premises Networks'. *IEEE Journal on Selected Areas in Communications*, 6(9):1545–1555, December 1988.

[69] G. K. Wallace. 'The JPEG Still Picture Compression Standard'. *Communications of the ACM*, 34(4):30–44, April 1991.

[70] R. Want. *'Reliable Management of Voice in a Distributed System'.* PhD thesis, University of Cambridge Computer Laboratory, December 1987.

[71] R. Want, A. Hopper, V. Falcao, and J. Gibbons. 'The Active Badge Location System'. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[72] I. D. Wilson, D. Milway, and A. Hopper. 'Experiments in Digital Video for Workstations'. Technical Report 89–2, Olivetti Research Laboratory, Trumpington Street, Cambridge, March 1989.

[73] S. Wray. 'The Interface to Pandora's Box'. Technical Report 89–4, Olivetti Research Laboratory, Trumpington Street, Cambridge, November 1989.

[74] S. Wray, T. Glauert, and A. Hopper. 'Networked Multimedia: The Medusa Environment'. *IEEE Multimedia*, 1(4):54–63, Winter 1994.