Synchronisation Services for Digital Continuous Media

Cormac John Sreenan

Christ's College
University of Cambridge

A dissertation submitted for the degree of
Doctor of Philosophy

October 1992

Abstract

The development of broadband ATM networking makes it attractive to use computer communication networks for the transport of digital audio and motion video. Coupled with advances in workstation technology, this creates the opportunity to integrate these continuous information media within a distributed computing system. Continuous media have an inherent temporal dimension, resulting in a set of synchronisation requirements which have real-time constraints. This dissertation identifies the role and position of synchronisation, in terms of the support which is necessary in an integrated distributed system.

The requirements for synchronisation are divided into two categories. Stream synchronisation is concerned with issues such as delay jitter with respect to the presentation of a single continuous media stream, as well as maintaining the relationships which may exist between multiple streams. Event synchronisation allows applications and system services to co-ordinate their behaviour based on information about the presentation of a stream. The central feature of the proposed solution is known as a synchronisation service. This service performs stream synchronisation under application control and supports external interaction to allow event-based synchronisation. Within the service, each stream is handled by a stream agent which understands stream specific details, including its format and encoding. Agents are coordinated in order to achieve inter-stream synchronisation. An interesting feature of the design is the ability to operate the service in a distributed fashion, allowing synchronisation between agents at different destinations.

An experimental implementation of the synchronisation service exists and has been used to evaluate the proposals. Multi-threaded stream agents run in user space, over a locally developed micro-kernel. Streams are transmitted using a multi-service protocol, while control and feedback are supported using an RPC mechanism. Experiments were performed in an ATM inter-network using multi-media workstations, each equipped with an Olivetti Pandora Box. The feasibility of supporting synchronisation as a system service has been demonstrated and the effectiveness of the design examined. Conclusions are reached regarding the influence of CPU scheduling on the performance of the prototype.

To my wife Joan

and

my parents John and Margaret

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.

No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

Trademarks

Acknowledgements

I would like to thank my supervisor Professor Roger Needham for the essential guidance which he provided throughout this research. He also made useful suggestions regarding specific aspects of the project and the contents of this dissertation. Special thanks go to my wife Joan, who has been an invaluable source of advice and support. She provided encouragement at every stage of this project and made significant comments on the dissertation itself. My parents, brothers and sister have supported me in many ways throughout my studies. Their interest in my work and constant encouragement are particularly appreciated.

Many thanks to the following people for reading and commenting on parts of this dissertation: Cosmos Nicolaou, Derek McAuley, Eoin Hyden, John Bates, Simon Moore, Richard Black and Paul Jardetzky. The interest expressed by John Herbert and Ken Moody at several stages of the project is also appreciated.

I am grateful to the Computer Laboratory for the use of its computing facilities, and to Andy Hopper for the use of equipment provided by Olivetti Research Limited (ORL). In addition, Alan Jones and David Greaves at ORL are thanked for their readiness to offer advice on Pandora and the Cambridge Backbone Network respectively. A studentship from the Science and Engineering Research Council (SERC) covered the cost of University and College fees for three years of study.

Contents

List of Figures

List of Tables

Glossary

ANSA	Advanced Networked Systems Architecture
ATM	Asynchronous Transfer Mode
AVSS	Audio/Video Support System
B-ISDN	Broadband Integrated Services Digital Network
CBN	Cambridge Backbone Network
CBR	Constant Bit Rate
CCITT	Comitée Consultatif International Télégraphique et Téléphonique
CD-I	Compact Disk Interactive
CD-ROM	Compact Disk Read Only Memory
CFR	Cambridge Fast Ring
CMFS	Continuous Media File Server
CODEC	COder-DECoder
DAN	Desk Area Network
DMA	Direct Memory Access
DPCM	Differential Pulse Code Modulation
DVI	Digital Video Interactive
EDF	Earliest Deadline First
FCFS	First Come First Served
FIFO	First In First Out

GPS	Global Position System
IP	Internet Protocol
IPC	Inter-Process Communication
ISLAND	Integrated Services Local Area Network Development
ISO	International Standards Organisation
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
MAN	Metropolitan Area Network
MIDI	Musical Instrument Digital Interface
MPEG	Moving Picture Experts Group
MSNA	Multi-Service Network Architecture
MSNL	Multi-Service Network Level
NTP	Network Time Protocol
OSI	Open Systems Interconnection
PLL	Phase Locked Loop
PTM	Packet Transfer Mode
QOS	Quality Of Service
RFC	Request For Comments
RPC	Remote Procedure Call
SMPTE	Society for Motion Picture and Television Engineers
STM	Synchronous Transfer Mode
TBC	Time Base Corrector
TCP	Transmission Control Protocol
TOG	Thrown On Ground
VBR	Variable Bit Rate
VCO	Voltage Controlled Oscillator
VCR	Video Cassette Recorder

Chapter 1

Introduction

This chapter provides a general introduction to the subject of the dissertation. In addition, the research objective is explained and an overview of the other chapters is given.

1.1 Perspective

Over the last decade, computers have become valuable tools for presenting information, thus enhancing their traditional role as data processors. This has prompted the development of so called *multi-media* applications, which employ several different types of information media, such as still images and graphics, in addition to text. Currently, there is considerable interest in extending this support to encompass digital audio and motion video. These are called *continuous media*, a term which refers to the inherent relationship between the sequence of discrete data samples of which they are composed.

Each encoded sample in a continuous media sequence has a defined temporal position and duration. Video samples are known as *frames*, for which presentation nominally takes place at the 25 Hz value defined by broadcast television standards. The size of a frame depends on the image dimensions and number of bits per pixel. Audio samples are smaller and occur more frequently than video frames, with a telephone quality CODEC producing 8 bit samples at 8000 Hz. At higher levels, audio samples are often grouped into *phrases* (delineated by silence intervals) to facilitate functions such as editing and random access. Continuous media are voluminous in nature, causing bandwidth and storage problems which have been ameliorated by advances in high-speed networking and data compression techniques.

The integration of continuous media into operating system and communication architectures provides many benefits, but is complicated by the need to respect their temporal characteristics. In particular, their presentation can be heavily influenced by values for end to end delay and delay variations (referred to as *jitter*). Typically, these must be low and bounded in order to meet application requirements. Therefore, handling continuous media is a *real-time* activity, in that determining a satisfactory result must take temporal as well as logical aspects into account. Continuous media can be distinguished from more conventional information types, by the fact that they may allow a degree of reliability to be sacrificed in exchange for predictable temporal behaviour.

Research into the design of networks to support data with different traffic characteristics (known as *multi-service* networks) suggests that Asynchronous Transfer Mode (ATM) is the most appropriate choice of multiplexing mechanism. ATM networks transfer data in small packets, known as *cells*, allowing fine granularity bandwidth sharing and reducing the delays due to contention and routeing which are associated with conventional Packet Transfer Mode (PTM) networks. Taken in combination with asynchronous access, this allows sources with different and varying bandwidth requirements to be efficiently supported. This is relevant for continuous media because many compression algorithms aim to conserve bandwidth by generating a variable bit rate, thereby increasing the *burstiness* of the resulting traffic pattern. ATM support for this form of traffic is in contrast with the operation of Synchronous Transfer Mode (STM) networks, which provide delay and jitter guarantees in a regime with fixed bandwidth allocation. ATM is seen as a compromise between the two extremes of PTM and STM, and has been adopted as the basis for the Broadband Integrated Services Digital Network (B-ISDN) [CCITT88].

Current research is addressing the problem of optimising the inherent flexibility of ATM networks and the related task of describing the communication requirements of different traffic classes. The latter is commonly referred to as the desired *Quality of Service* (QOS) and is viewed as taking the form of a set of application specified parameters. These can be used to influence network performance in combination with appropriate mechanisms for managing and policing the allocation of network resources. The ability to negotiate a QOS for communication is particularly important for the satisfactory support of continuous media in an integrated network.

1.2 Integrated Continuous Media

The task of integrating continuous media within current workstation environments is mainly handicapped by the lack of appropriate policies and mechanisms for operating system resource management. In addition, many workstations are incapable of handling the large data volumes which are involved, due to limitations on data copying, bus bandwidth and device access. Perhaps the earliest project to demonstrate the benefits of running continuous media applications in a workstation environment is Pandora [Hopper90, Hopper92, King92], which is a component of the experimental apparatus used in Chapter 5. Pandora minimises many of the aforementioned problems by adopting a hybrid approach, in that continuous media are handled within a special purpose Transputer-based computer. This is equipped with a camera and audio I/O devices, and is directly linked to a UNIX host, allowing video to be presented by a process of analog mixing with the host display output. Application control of a Pandora Box is achieved by interacting with a specially extended server for the X Window System [Scheifler86], which runs on the host. Continuous media are transmitted and received by Pandora over its interface to the Cambridge Fast Ring (CFR) [Hopper88], a locally developed slotted ring LAN which operates at 50 Mb/s. The nature of the CFR media access protocol ensures a bounded delay, which effectively guarantees a minimum point to point bandwidth; while the small slot size (32 bytes) allows fine granularity bandwidth sharing. These properties correspond with the characteristics of an ATM network described in the previous section.

1.3 Synchronisation Issues

A collection of continuous media samples is referred to as a *stream* and is viewed as flowing from source to sink during transmission. A stream which is sourced from an input device, such as a camera or microphone, is described as being *live*, as opposed to a *stored* stream which is played back from a secondary storage device. This dissertation considers the subject of synchronisation for streams in a distributed workstation environment. The issues are divided into three related topics:

- intra-stream synchronisation;
- inter-stream synchronisation;
- event-based synchronisation.

Intra-stream synchronisation stems from the temporal characteristics of continuous media. It aims to maintain a correspondence between the presentation of stream samples and the passage of real time, being mainly concerned with the problems caused by delay and jitter. Hence, it is influenced by QOS issues and network performance. Inter-stream synchronisation deals with relationships which may exist between individual streams. An example is synchronising voice output with the lip movement of the speaker, commonly known as *lip-synch*. Finally, event-based synchronisation aims to allow applications to co-ordinate other activities with stream output. This type of synchronisation is of a discrete nature and could be used, for example, to synchronise the display of subtitles to a video stream during its presentation.

In a distributed system, it must be stressed that event-based synchronisation plays an important role in supporting the application level tools which have been developed for multi-media presentation and document systems. These programs commonly allow synchronisation requirements to be specified and executed in terms of a time-line, along which data objects can be arranged to indicate their desired presentation time. Examples include Athena Muse [Hodges89] and QuickTime [Wayner91]. Other approaches use a script and/or linked based mechanism to describe the presentation order, including those of [Zellweger89] and [Fujikawa91].

1.4 Objective

This dissertation deals with synchronisation for continuous media in a distributed system. The aim of the research is to gain a deeper understanding of the relevant problems and use this to identify a suitable role and position for synchronisation within system architectures. An important influence in an integrated context is that of QOS, and a logical extension is to determine its relationship with mechanisms to perform synchronisation, notably in regard to operating system scheduling.

1.5 Overview

The remainder of the dissertation is structured as follows:

Chapter 2 discusses intra-stream synchronisation, examining the issues and proposing the synchronisation service as a suitable design for use in an integrated distributed system.

In Chapter 3, inter-stream synchronisation is investigated and a highly flexible solution is proposed, extending the design presented in the previous chapter.

Event-based synchronisation is dealt with in Chapter 4, where the supporting role of the synchronisation service is discussed.

Chapter 5 details an experimental evaluation of a prototype synchronisation server. The experiments were performed in an ATM inter-network and characterise the influence of CPU scheduling and kernel timing support on server operation.

In Chapter 6, a synopsis of recent research on continuous media synchronisation is provided, along with a detailed comparison between the proposed design and other closely related work.

Chapter 7 reviews the work and summarises the conclusions.

Chapter 2

Intra-Stream Synchronisation

This chapter discusses synchronisation for an individual stream of continuous media. Design issues for a facility to realise this function are examined in detail.

2.1 Introduction

The fundamental characteristics of a timed stream of continuous media were described in the previous chapter. A stream consists of a sequence of encoded samples which are transferred between source and sink devices, typically for presentation purposes. An inherent property is its temporal dimension, which defines a mapping between sample presentation and real time. For live presentation, the rate of advance nominally equals that of real time, but otherwise, both the rate and direction may be altered to achieve special effects such as fast-forward and reverse play-back. This intra-stream synchronisation must be maintained, both to reproduce the information as originally captured and to ensure a quality level which is acceptable to the application. In a distributed system, this task requires explicit support, due to the occurrence of variable delays and transmission errors. Sections 2.2 to 2.4 look at the problems which can lead to a loss of synchronisation. The origins of these problems are identified, and solutions are examined. Section 2.5 investigates software support for synchronisation, in terms of its position within the system architecture. Problems with existing systems are pointed out, and a more suitable design is introduced. Detailed aspects of this design are discussed in Section 2.6, followed by a study of CPU scheduling support in Section 2.7. Section 2.8 is concerned with the physical location of this software support and leads to the Synchronisation Service proposal in Section 2.9.

2.2 Jitter in the Network

A synchronous system achieves synchronisation by using a single clock to control the capture of samples, their communication and eventual presentation at an output device. In particular, an STM network ensures that they are guaranteed to incur equal communication delays. Thus, the sink simply presents samples as they arrive, in the knowledge that it is synchronised with the source. In an asynchronous system it is expected that samples incur different delay values, making this approach unsuitable.

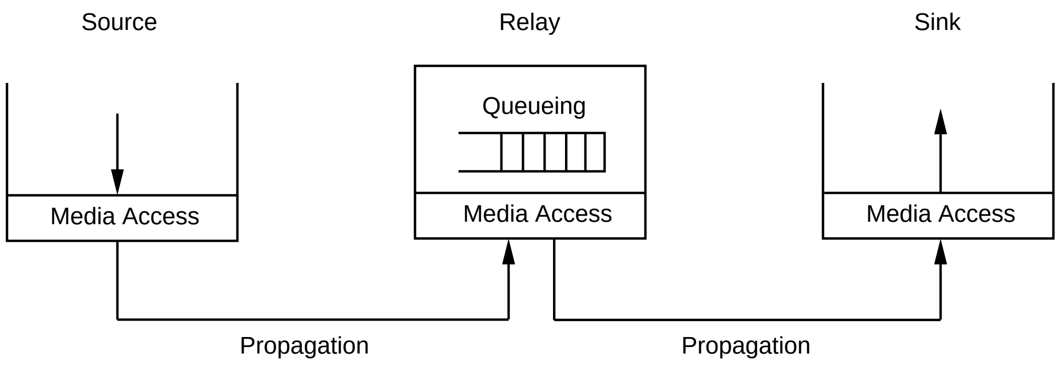


Figure 2.1: Delays in the Network.

Delay variations occur at several points in the network, as illustrated in Figure 2.1, but are dominated by queueing at intermediate relay nodes. The media access mechanism may also produce significant jitter, especially if it is non-deterministic, like that of the Ethernet, which backs off exponentially in the face of contention. The result is that samples arrive late at the sink, causing gaps in the presentation, as shown in Figure 2.2. Unpredictable jitter can result in losses due to sink and/or relay buffers being overrun by backed-up samples which arrive in clusters, perhaps leading to a serious degradation in presentation quality. The basic approach of discarding samples which are late upon arrival at the sink, can also result in significant losses, making it equally unattractive.

As previously described, there is broad agreement that ATM networks are an attractive compromise between the extremes of STM and PTM for the support of multi-service traffic. Conventional PTM networks were not designed to handle data which is sensitive to delay and jitter. In limited contexts, such as within a single LAN, it may be possible to engineer a solution which provides satisfactory performance under suitable operating conditions. An example is the Etherphone system [Swinehart83], which transmits voice over a dedicated Ethernet using a small packet protocol to minimise contention. ATM networks allow a more general solution, capable of operation in wider area environments and without imposing such highly restrictive constraints. They offer the benefit of being able to influence the network service which is provided, often in terms of negotiating a guarantee based on traffic characteristics. It may be possible to express the acceptable QOS in terms of the following parameters (see the statistical and deterministic descriptions in [Ferrari90a]):

- throughput;
- delay;
- jitter;
- cell losses;
- bit errors.

For continuous media traffic, the low and predictable delay and jitter characteristics of ATM networks are important and allow inter-network operation to be considered. This is facilitated by using efficient hardware cell relays with lightweight virtual circuits on which routeing and QOS may be based.

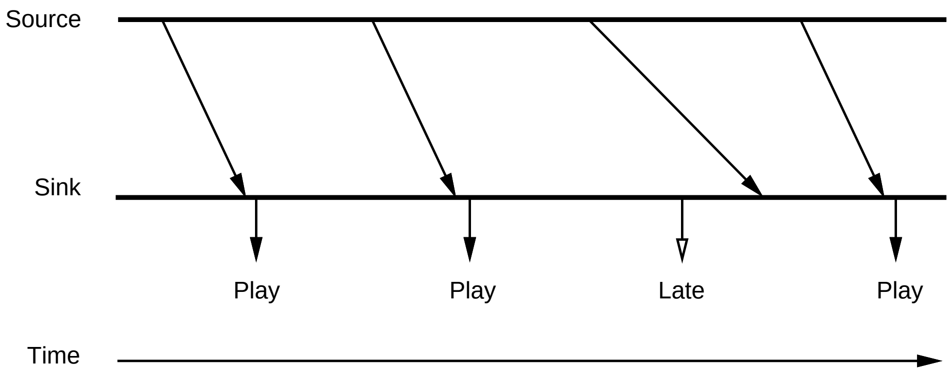


Figure 2.2: Late Sample Arrival.

Tolerance to jitter in sample presentation is determined by several factors, including the sample rate, encoding and target application. Typically, telephone quality audio is not seriously impaired by up to 4ms worth of jitter [Ades86], but a stream of high quality music reduces this value to under a millisecond. TV quality video is generally unaffected by up to 10ms jitter [Campbell92b, Leydekkers91]. In an ATM LAN it may be possible to achieve these targets by negotiating a matching QOS guarantee, so that components which handle a stream do so within minimum and maximum delay bounds. However, extending these constraints into an inter-network is impractical and if attainable, would lead to extremely inefficient and expensive resource usage. Therefore, it is necessary to provide a mechanism to *smooth* superfluous jitter prior to presentation, which in turn, may allow network jitter constraints to be relaxed. This approach is obligatory in a PTM network, where excessive jitter is highly probable. The task of smoothing jitter is carried out by placing an *elastic buffer* in the stream path, which contracts and expands to ensure that each sample has incurred a controlled end to end delay. The optimum location for this buffer is at the sink, where any subsequent jitter, prior to presentation, is at a minimum.

2.2.1 Elastic Buffer Operation

An elastic buffer is an ordered queue in which samples reaching the sink are stored, prior to being subsequently removed for presentation. While in the buffer, each sample accumulates a delay which complements that received in the network, thus ensuring equal values of end to end latency upon removal from the buffer as shown in Figure 2.3. The aim is to smooth jitter, thereby avoiding the arrival of late samples as illustrated in Figure 2.4. Consequently those samples which experience a minimum communication delay spend the maximum amount of time in the buffer and vice versa. The term elastic refers to the buffer occupation, which varies over time depending on the jitter of the buffered samples. Thus, if all samples received the minimum delay, the buffer would be perpetually full. Memory requirements for the buffer are calculated based on knowledge of the sample size and rate. This is straightforward for Constant Bit Rate (CBR) streams, but using a Variable Bit Rate (VBR) the variance in bandwidth must be considered.

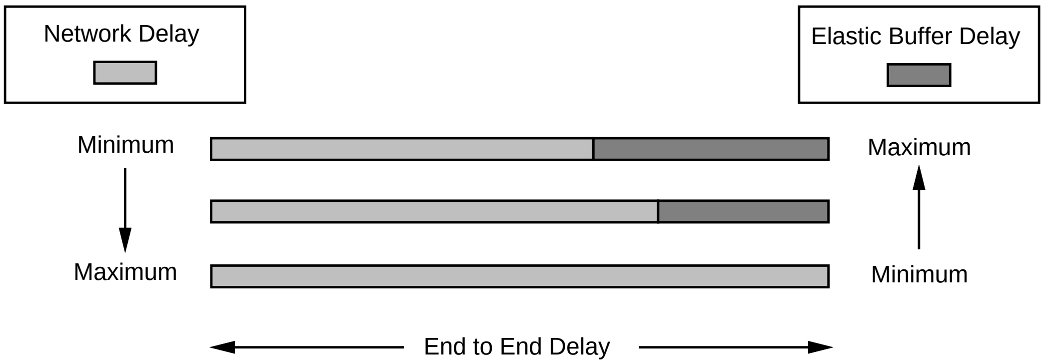


Figure 2.3: Elastic Buffer Principle.

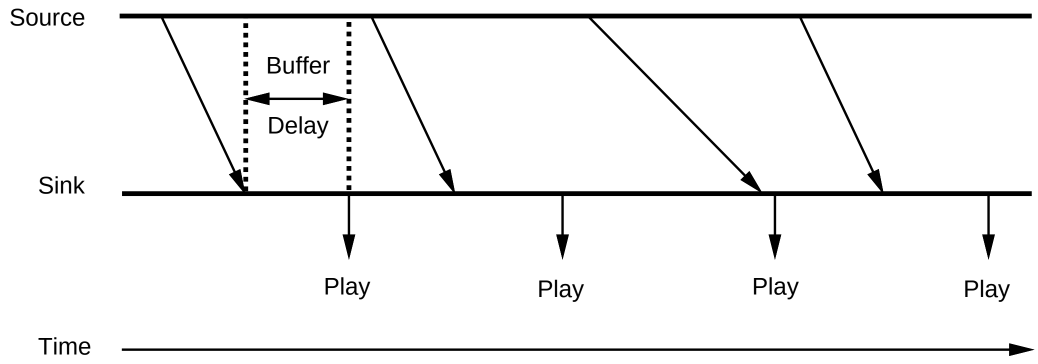


Figure 2.4: Elastic Buffer Operation.

Buffer length refers to the upper limit on the elastic buffer capacity and is usually expressed in terms of time units rather than samples or bytes. Thus, the sink presents each sample at a fixed time after it originated from the source. In essence, this delay to the target play out time results from concatenating the network and elastic buffer delays. In order to be capable of dealing with the expected jitter, the buffer length must be large enough to smooth samples which have received a network delay between expected minimum and maximum values. If the buffer length is less than the maximum variation in network delay, then the buffer will underflow, because of samples which arrive later than expected.

2.2.2 Length Requirements

An application may choose to select an elastic buffer length which operates over a smaller jitter range than that which is expected. This generally applies to situations where the predicted jitter is sufficiently large that the required buffer length would cause a significant increase in end to end delay and/or memory requirements. In particular, if jitter is unbounded then this compromise is unavoidable.

This decision is also influenced by the application tolerance to jitter and sample losses. When such a buffer length is used, some samples arrive later than their target play out time and are said to be excessively late. To an application with no tolerance to jitter, these are effectively useless and should be discarded. On the other hand, an application may wish to indicate some amount of jitter tolerance in order to reduce this source of lost samples. This represents an end to end QOS requirement and as such subsumes that which is supported by the network. An application may also consent to increased loss and/or jitter for a corresponding reduction in the real or virtual cost of network and elastic buffer resources. Experimental evidence shows that an ATM inter-network exhibits a delay distribution where most cells lie within a narrow range [Dixon91]. Using predictable information of this form, it should be possible to derive an estimate of total discarded samples for a selected buffer length.

The action of discarding late samples can be extremely serious, particularly if the stream is compressed using predictive techniques, where a sample relies on the contents of its predecessors such as in the MPEG video standard [Le Gall91]. It may be possible to provide limited compensation by detecting that a sample is going to be late and invoking an extrapolation algorithm based on a previous sample. However, in general it is necessary for an application to choose a balance between buffer length and network QOS. This fact should be taken into account by the designers of compression algorithms which require reliable transport facilities.

While it may be tempting to reduce losses and network QOS demands by using a large buffer length, there are a number of important restrictions to be considered. In addition to memory size limitations, the effect of buffering on total delay cannot be ignored. This is significant for many applications which involve live streams such as telephony and video conferencing. In these cases a degree of *interactivity* occurs between the communicants in real-time [Sventek87]. This may be quantified by the delay between data being issued by one party before the other has an opportunity to reply [Nicolaou90b]. There exist thresholds for delay above which the level of interactivity between users in these applications is impaired.

The ISLAND project found that values up to 125ms are acceptable [Ades87] for telephone conversation, equivalent to the 100ms to 150ms range which is recommended for the public telephone system [Campbell92b]. Experience with the Pandora system [Hopper90] (reported in [Campbell92b]) led to the observation that an audio delay greater than 10ms was noticeable without the use of echo suppression. Video can generally sustain larger values as exemplified by the 230ms estimate in [Jeffay91], but significant delay differences between streams is problematic for inter-stream synchronisation. This issue is considerably less important for access to stored data, as long as the initial latency in starting a stream lies within bounds which are acceptable to a human observer.

Application QOS requirements are relatively static in that, for example, the maximum tolerable jitter for a telephone quality audio conversation remains constant even though it may be possible to obtain a smaller bound. Meanwhile, technology advances continue to yield networks with lower delay and increased predictability, suggesting an eventual relaxation in buffer length constraints. In a PTM network, it is difficult to guarantee QOS, necessitating a balance to be made between delay and loss requirements.

2.2.3 Dynamic Response

Delay and loss requirements can be balanced dynamically by taking account of current jitter and accumulated losses. This is advisable in situations where jitter may render elastic buffer operation ineffective, producing excessive losses. This can occur if a network QOS is unreliable or the buffer length lies too far outside the jitter bound. The response is to allow the buffer length to vary dynamically, based on the prevailing conditions. When normal conditions return, the buffer regains its desirable length.

In the ISLAND system [Ades86], network delays were low and reasonably predictable, while a small end to end delay was required. A static value for buffer length was selected, ensuring that each voice sample actually received this fixed delay. This choice was influenced by the use of a low cost telephone device with a small amount of memory. The UNIVERSE system [Leslie84] operated in an environment where network conditions could often vary substantially during a telephone call, prompting the use of a dynamic scheme. In the Pandora system [King92] network delays are very predictable, but excesses can occur due to occasional re-framing breaks in CFR operation. In addition, streams which are retrieved from the storage facility are subject to less predictable behaviour at the source, often resulting in audio jitter of 50ms[1]. A dynamic response mechanism is used to deal with these situations for audio streams.

Dynamic response is usually implemented by maintaining low and high bounds on the acceptable variation in buffer length, often referred to as *water marks*. Reaching a water mark causes an increase or decrease of the buffer length, achieved by pausing or skipping samples respectively. It is important to ensure that this does not unduly affect the presentation quality. For audio this is aided by making effective use of silence intervals; while for video it may be possible to extrapolate samples during a pause and skip during periods of low motion. Particular care is needed so that excessive output jitter isn't caused by these activities, due to changes in the buffer delay which successive samples incur. Similarly, operation must be monitored and tuned, so that constant adjustment and consequential oscillations are avoided. An inherent weakness of the approach is the inaccuracy which may result from using a delayed arrival to indicate changing network conditions and predict future behaviour.

[1]Information supplied by A. H. Jones of Olivetti Research Ltd.

2.2.4 Determining Delays

An elastic buffer aims to normalise sample delays by extending the total latency between transmission and presentation. However, this requires knowledge of the network delay which a sample has encountered prior to reaching the sink, so that the holding time can be determined. Obtaining a *precise* value requires a clock to be shared between the source and sink. As the first sample leaves the source it is stamped with the current time, which is compared against its arrival time at the sink using the same clock. If stream data is transmitted at fixed intervals the timing of subsequent samples can be calculated using sequence numbers. Otherwise, each sample must carry an explicit *time stamp* which is large enough to convey the time with sufficient resolution throughout a stream lifetime. Consistently good accuracy for the time must be maintained, given the high frequency of audio samples in particular. The additional use of time stamps to support Earliest Deadline First (EDF) scheduling along the route of a stream is considered in [Chen89], but the practical problems of that scheme are significant and are not explored in that work.

The use of a clock which is physically shared is only feasible in a small geographical area. There are two basic approaches to implementing a shared logical clock in a distributed system. Central clock systems use special purpose hardware as in the Global Position System (GPS), which maintains accuracy to within a few nanoseconds [Levi90]. In a system with many hosts, economic reasons make it necessary to use software techniques to allow other hosts to synchronise with this type of central authority. This can be achieved very accurately over dedicated links with known propagation delays. This approach is commonly used by telecommunication companies [Abate89], but is expensive to install and operate.

Alternatively, operation can make use of estimates for transmission delays over shared communication lines, allowing clock differences to be approximated by the exchange of messages. Accuracy and reliability suffer, but the approach is inexpensive and very effective, as demonstrated by several existing algorithms. An early system achieved a 40ms bound between clock variations in a LAN [Gusella83]. More recent work on the Network Time Protocol (NTP) reports accuracy in the millisecond range for a local area and to within tens of milliseconds across the *Internet* [Mills89]. This is achieved using a self organising, hierarchical network of time servers, configured on the basis of estimated accuracy and reliability.

On the other hand, a delay estimate may be used within a stream to work out a suitable presentation time for the first sample, used in combination with relative timing information for subsequent data. In essence, an estimate is obtained by measuring the round trip time for a message and using it to determine the typical one way propagation delay. In a local area this solution can produce a very accurate result if knowledge of message processing times is reliable. Unexpected delays can be minimised by providing kernel support for message delay measurement as in [Mills83], or hardware support for time stamping message transmission and arrival [Kopetz87, Damm89]. Outside a local area it is much less accurate, because the probability of having a symmetric delay distribution is significantly reduced. Use of a suitable jitter bound in an ATM network provides a solution to the problem of making delay estimates.

A method where each message keeps a record of the delays which it encounters can be very accurate [Montgomery83], but suffers from the overheads of performing calculations at each hop along the route. This is in conflict with the principles of ATM, in which relays are designed to be as simple as possible.

The most straightforward approach makes a basic assumption regarding the network delay experienced by the initial sample on a stream. Upon arrival the sample is buffered for a predetermined period of time and subsequent samples are presented based on relative timing information. It is commonly assumed that the initial sample incurs a minimum network delay and hence a delay equivalent to that of the buffer length is imposed. The worst case is that the initial sample actually receives the maximum communications delay, resulting in a situation whereby the desired end to end latency may be exceeded by an amount equal to the jitter bound. In practice an initial sample is more likely to encounter exceptional delays, for reasons such as cache misses in network relays. Consequently it is suggested that the described assumption is unsuitable. Instead, one should either base operation on a sample which occurs later in the stream, or assume that the initial sample encountered a maximum delay. In any case, the technique is simple to implement and suitable for an environment where values of jitter are low and bounded.

2.3 Jitter in the End Systems

In addition to delay variations caused by the network, the behaviour of the hosts at the source and sink of a stream, referred to as *end systems*, may also be problematic. Figure 2.5 shows the typical stages of end system processing which are necessary for a stream sample. Devices may be synchronous, in which case they are driven by an internal clock, or asynchronous where I/O is performed in response to the advance of the end system clock. Samples are encapsulated in packets for transmission along with any relevant header information such as time stamps. Henceforth in this dissertation it is assumed that explicit time stamps are available, although these may be replaced by equivalent rate information as described in Section 2.2.4. For audio it is common to place several samples in a single packet in order to avoid low network utilisation and excessive overheads. This requires a limited amount of buffering prior to packetisation. Typical values for the duration of audio packet contents lie in the 2ms to 4ms range, chosen since the loss of this quantity is not generally noticeable. Video samples are larger and may need to be dispatched in multiple packets depending on network limitations and coding issues.

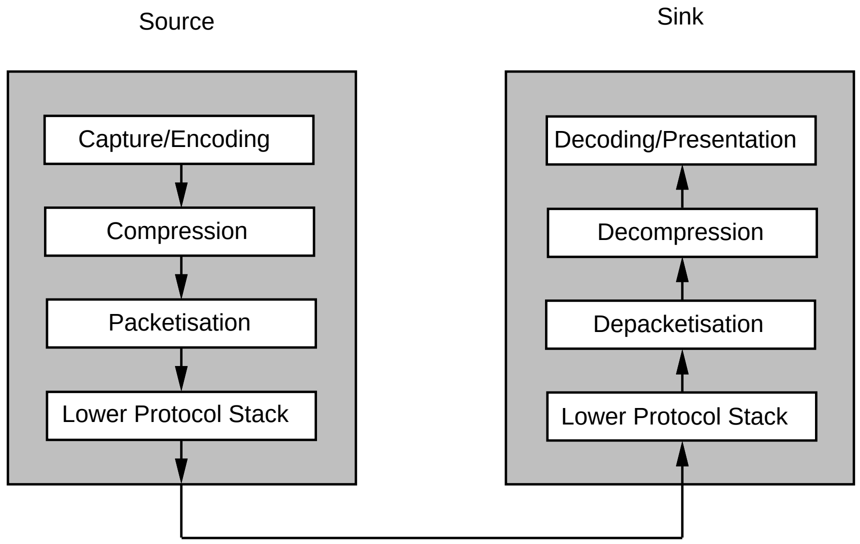


Figure 2.5: Delays in the End Systems.

Jitter due to protocol processing at the end systems and relays can be particularly severe. Protocol stacks are commonly described in terms of a set of functional layers, the most well known example being the seven layer Open Systems Interconnection (OSI) reference model of the International Standards Organisation (ISO). In many cases, services existing at higher layers are multiplexed onto those offered at lower layers. This can introduce significant statistical jitter and cause QOS *cross-talk*, where the network performance experienced by a given stream is influenced by other ongoing communication activities [Tennenhouse89]. An architecture suitable for multi-service traffic is proposed in [McAuley89], in which this is minimised by adopting a high degree of vertical integration. Further protocol delays result from the operation of flow control algorithms, which are used to avoid buffer overrun at the sink. For continuous media these algorithms are likely to be rate-based, because of the delays involved in conventional sliding window approaches.

An important source of jitter may be derived from inappropriate CPU and device scheduling. At the source, this imposes jitter on the outgoing stream, which may be particularly evident if the data is being retrieved from a disk storage device. In a reciprocal fashion, sink processing may cause jitter far in excess of that introduced by the network. Unless these are bounded, then the effectiveness of an elastic buffer may be seriously compromised and samples can be lost due to buffer overrun along the path. In addition, highly predictable scheduling is needed to allow sample presentation from an elastic buffer, without violating total QOS requirements.

Increasing the predictability of CPU scheduling requires attention to be paid to sources of *priority inversion*. This condition is said to have occurred when the progress of a high priority activity is blocked by processing of a lower priority. This can happen when a resource is shared between tasks of different priorities. The *priority inheritance* algorithm attempts to limit the duration of inversion in this situation, by dynamically changing the priority of the blocking tasks. Priority inversion can also occur when there are lengthy periods of time during which preemption is prohibited, for example, while executing within the kernel of conventional UNIX systems. A recent publication describes an implementation of UNIX in which making the kernel preemptive was an important goal [Khanna92]. In addition to preemption, the traditional manner in which interrupts are handled at elevated priorities is a source of inversion. This can be ameliorated by the use of interrupt threads to which part of the handling activity can be farmed out for scheduled processing.

In the majority of existing projects, scheduling problems have been minimised by using *dedicated systems* to deal with streams. This approach is often in keeping with the application domain, as in the networked telephones used in ISLAND and Etherphone. Both the Multimedia Network Interface [Blair90] and Pandora systems are dedicated to handling audio and video under control of an attached UNIX host. This relaxes contention for host bus bandwidth and CPU processing, while alleviating scheduling considerations by making the end system operation more deterministic. Of course, problems from non-deterministic scheduling of control operations on the host remain.

This dedicated approach has allowed scheduling problems to be minimised to the point where they can safely be ignored. Developments in workstation technology make it possible to consider a system, without dedicated support, in which continuous media are fully integrated. In comparison, this is likely to be less costly and to provide more flexibility, but it requires attention to be paid to resource management in order to support QOS based stream processing.

2.4 Clock Drift

The capture and presentation of a stream is usually driven by different physical clocks. The frequency of the crystal oscillator within a clock can vary over relatively long periods of time due to temperature changes and imperfections in the crystal lattice. This results in a *frequency offset* with regard to real time, causing a drift rate which can be as high as 1 part in 10^3, although values approaching 1 part in 10^6 are more common. The actual value depends on the overall quality of the device. This issue is important because the rates of operation for a stream source and sink may differ, leading to an eventual loss of synchronisation if left unchecked. This results from changes in buffer length which lead to jitter and the occurrence of overflow or underflow.

Using clocks of high quality it may be possible to ignore this problem for the duration of a stream. However, if significant drift is present, a solution which synchronises clock frequencies to the correct value may be desirable. This was proposed for ISLAND, in which idle telephones were forced to listen to regular transmissions from a dedicated network time service. Adjustments which resulted from this protocol were realised using a Voltage Controlled Oscillator (VCO) within each telephone. In general, a simpler scheme may be acceptable, whereby it is ensured that clocks operate at the same frequency, but not necessarily that which is absolutely *correct*. This lends itself to a solution in which a sink dynamically adapts to the frequency of the source, based on observation of the incoming data rate.

A commonly used algorithm employs a digital Phase Locked Loop (PLL) to derive an approximation of the source frequency and guide corresponding adjustments at the sink using a VCO [Ades87]. The PLL can be driven by the filling level of the elastic buffer as illustrated in Figure 2.6, or alternatively by locking on directly to sample arrivals [Cochennec85]. For increased accuracy a scheme which also performs an initial adjustment phase prior to transmission has been proposed [De Prycker87]. An important design problem is making sure that uncontrolled oscillations do not result from misinterpreted delay variations. A recent simulation study has shown satisfactory results in this regard for a variety of jitter distributions [Almeida91].

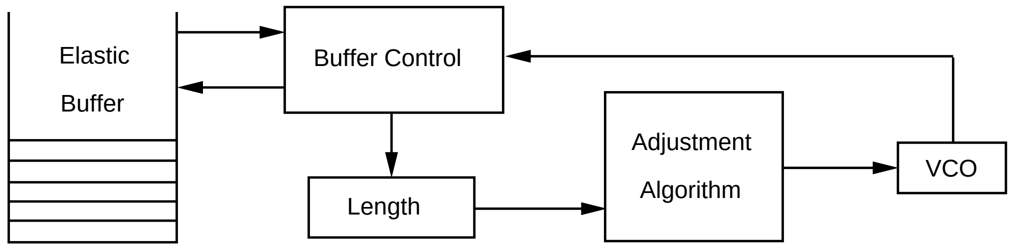


Figure 2.6: Buffer Based Frequency Adaption.

PLL-based schemes are designed to deal with fixed data rates. This is a significant drawback, given that a major benefit of packet networks is their ability to efficiently support variable bandwidth traffic. VBR streams offer many benefits over the use of CBR, notably in regard to bandwidth reductions [Verbiest86]. A related approach uses sample time stamps to approximate the frequency difference, allowing support for variable rates. However this requires knowledge of time stamp formats and locations, making a hardware solution very restrictive. The precision that can be achieved is limited in practice [Almeida91].

An algorithm known as *slipping* can solve the problem without actually making frequency adjustments. This is achieved by performing phase slips at the sink to avoid undesirable buffer conditions. These are made by simply adjusting the elastic buffer length. To accommodate the allowable drift in length before intervention takes place, it is necessary to make a corresponding increase to the specified maximum length of the elastic buffer. As before, the need for adjustment can be detected by observing the average length or comparing the time stamps of incoming samples with the sink clock. A similar approach is suggested in [Nicolaou90a], where the average jitter over a selection of samples is monitored.

Phase slipping has the drawback of interfering with the stream contents. However, the effect of slip and pause operations can be minimised by using specialised stream knowledge, like the occurrence of audio silence intervals. The ability to make these adjustments is, of course, governed by the loss parameters of the stream QOS. Given the simplicity of this scheme it is amenable to a software solution and consequently issues of scale for multiple streams are not limited by the number of physically adjustable sink clocks. An implementation is likely to produce less accurate results than frequency adjustment algorithms, but given the increasing accuracy of clock devices this is not an unreasonable compromise.

Synchronisation losses can also occur due to inaccuracies in maintaining a software clock. This is normally updated by a kernel in response to hardware interrupts from a timing device, driven by a physical clock. In general, these devices do not provide a facility for counting interrupts and consequently the recorded clock value will drift if they are allowed to pass without being acknowledged. This can lead to substantial drift in terms of sample time stamps and their use. Section 5.2.2 describes the modifications which were made to the micro-kernel used in the prototype system to offer a solution to this problem. In some computers the frequency at which clock interrupts occur is non-integral. This can also lead to drift and requires attention to be paid to handling rounding errors in updating the software clock. The scale of these forms of drift can be an order of magnitude greater than that which results from the characteristics of physical clocks.

2.5 Software Support

The basic configuration for intra-stream synchronisation is illustrated in Figure 2.7. Samples arrive at the destination, are inserted into the buffer and in response to the advance of time are removed for presentation. The following sections discuss this mechanism in terms of the system software architecture at the sink.

2.5.1 Embedded Approach

The majority of existing systems, including those mentioned in the preceding sections, deal with continuous media in the context of an *embedded system*. This has the benefit of allowing temporal and bandwidth requirements to be satisfied without the need for sophisticated resource management. It is equivalent to the design of many conventional real-time control systems, where the pre-allocation of resources ensures that contention is not an issue, even for worst case loading scenarios. Continuous media systems using this approach are characterised by tight controls on external interaction and built-in assumptions regarding stream requirements and behaviour of the operating environment. In addition, system operation is generally hidden within a monolithic control program, accessible only through the use of a predefined interface. Intra-stream synchronisation is also a function of this program, typically provided within the lower levels of the communications protocol. An elastic buffer mechanism is tightly integrated with the operation of the protocol and output devices, while data is usually added and removed directly by interrupt handlers.

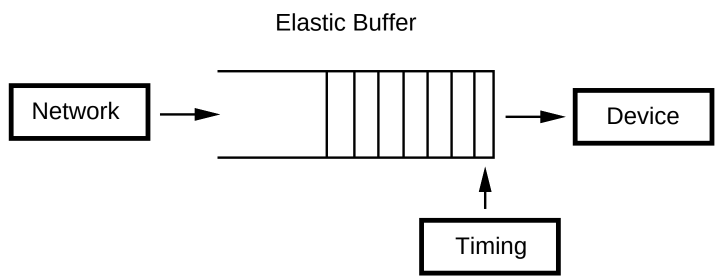


Figure 2.7: Basic Elastic Buffer Arrangement.

A temporal guarantee for sample output to the device is easily satisfied by the traditional manner in which interrupts are handled at an elevated priority. Exceptional conditions are dealt with in a best effort manner, using built-in knowledge of application requirements and resource availability. This general approach of supporting time constrained processing in interrupt handlers is similar to the method used in the FORMULA computer music system [Anderson90], where commands to be fed as output to a synthesiser are processed according to a strict schedule by the clock interrupt handler [Anderson86]. The interface to an embedded system is comparable to that of a sophisticated peripheral, in that the set of operations and the scope for control and feedback is statically predetermined. Continuous media systems of this form do not support the conventional concept of an application, but instead provide a simple path from the network to an I/O device and vice versa.

Supporting intra-stream synchronisation in this way is very restrictive and does not provide an attractive solution, except in a limited application context. There is no scope for an application to influence synchronisation by specifying a desired QOS. This is fundamental to guarantee satisfactory support for continuous media, but cannot simply be based on assumptions and built-in information, such as the media or network type. For example, in an ATM environment, the network jitter imposed on each stream should be determined by the negotiated QOS and consequently it is unreasonable to simply assume an elastic buffer of standard length. Each application should be allowed to balance, perhaps dynamically, the demands which it places on the network and the synchronisation mechanism, based on factors such as resource availability and cost. In some circumstances it may be acceptable to simply allow an application to select from a fixed set of service classes along the lines of the scheme in Etherphone. In that system, each stream has an expected jitter bound upon arrival at the sink of either 10ms of 100ms, depending on whether it is live or coming from the storage facility respectively. Conversely, an application should be able to gain access and respond to feedback of an informative or exceptional nature. Subsequent chapters discuss the reasons why this is an important element in the support of inter-stream and event-based synchronisation.

Close coupling of the elastic buffer mechanism with operation of the protocol and output device provides benefits for data throughput and predicting delays, but has several drawbacks. It is not suitable for the support of non-trivial synchronisation activities, notably those which involve external interaction. It exacerbates priority inversion problems by increasing interrupt processing latencies (see Section 2.3). Similarly, it makes it difficult to assign different priorities for individual streams. For a general solution it is important that CPU and device access control are determined according to known parameters, allowing a co-ordinated system for resource management and service degradation if the situation arises.

This tightly coupled arrangement also fails to isolate specific attributes of output devices from the synchronisation mechanism. Introducing this separation facilitates the use of different synchronisation approaches and heterogeneous stream types across different devices. Features which are specific to a given stream type, such as the ability to extrapolate when samples are excessively delayed, should be made

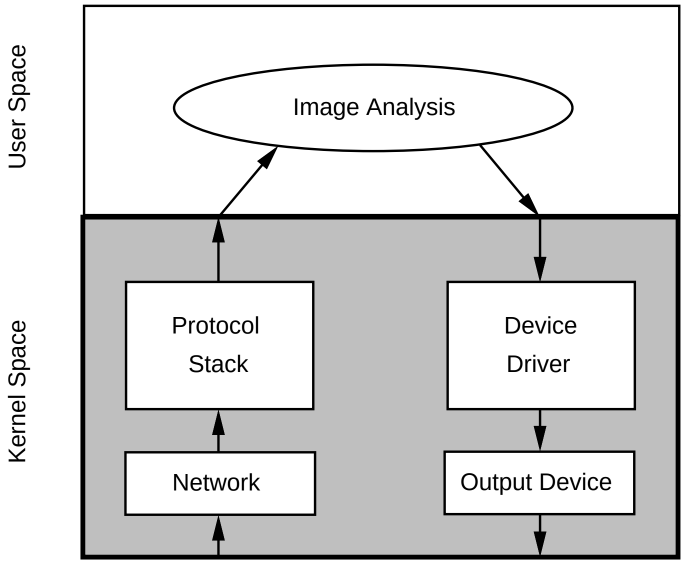


Figure 2.8: User Space Processing Scenario.

accessible. This is essential for operation in a heterogeneous environment and to allow informed decisions on when to make buffer length modifications. Imposing standard formats is a possibility, since even the basic functions require access to stream type information for the extraction of timing details. With the present lack of widely accepted international standards, notably for video, the need to allow the introduction of new types is important. This is facilitated by synchronising outside the kernel or in the higher layers of the communications protocol. Inter-stream synchronisation across multiple nodes, as discussed in Chapter 3, requires interactions which are also best performed outside of the lower levels of the protocol stack.

Handling streams *outside* of the kernel is necessary to support maximum flexibility for applications. By containing streams within the confines of the kernel, applications such as Window System servers act in the capacity of a stream controller. This is similar in spirit to the arrangement used in Pandora, where the X Window System server runs on an attached UNIX workstation and never directly handles streams. To date, continuous media have not been accessible to applications in a general way, mainly due to a lack of available bus bandwidth and CPU cycles within workstations, but this situation is rapidly changing. In most cases, this is only essential where stream specific manipulation is required, such as for image processing or software decompression as illustrated in Figure 2.8. This processing may be offered as an application level service. An example is an application which presents live images from a camera on a remotely controlled underwater vehicle, only after it processes them to improve contrast and eliminate the effects of water distortion. In this case, safety considerations with regard to vehicle control are likely to demand strict delay and loss bounds. By comparison, processing of stored streams should allow temporal constraints to be relaxed, notably in regard to overall delay.

Manipulation of streams in user space imposes severe penalties in terms of data movement, which may be eased by advances in bus and I/O architectures as well as operating system design. In particular, the UNIX approach of copying data between kernel and user space buffers is entirely unsuitable. The DASH kernel provides an alternative mechanism which allows messages to be mapped into and between process virtual address spaces, without physical copying of the data [Anderson89b]. A scheme based upon access to a contiguous set of I/O buffers, similar to that used in Topaz [McJones87], is supported in the WANDA kernel, used for the experimental programme in Chapter 5. A proposal by Sun for a UNIX multi-media environment envisages a scheme whereby streams are normally dealt with in the kernel, but if necessary may be accessed directly by those applications willing to incur the additional overheads [Hanko91a].

In most systems to date, sink processing for a stream is confined to data movement and decompression. In conjunction with the dedicated nature of an embedded system, this meant that smoothing jitter in the incoming protocol stack was perfectly acceptable, since subsequent processing was highly predictable in regard to delay. A recent proposal suggests that it should be possible to execute application modules on a dedicated system by employing it as an object-oriented platform [Blair92]. This is a move away from the inflexibility of an embedded system, but retaining a tighter degree of control over system behaviour than is possible in a workstation configuration. However, supporting this scheme while maintaining support for real-time behavioural requirements is very complex, except for the execution of very simple modules. In addition, a workstation based application remains unable to control operation of the dedicated platform in a time constrained manner.

The assumption that data is handled predictably after leaving the elastic buffer requires examination when a stream is routed via an application on its way from the network to the output device. Unless application level processing is controlled within acceptable limits, then jitter due to inappropriate resource scheduling may cause a loss of synchronisation. Providing a jitter guarantee for an application requires deterministic CPU scheduling based on a mechanism for describing processing requirements as proposed in [Anderson89b]. Further research is needed to determine the feasibility of operating a scheduler in this manner, where an application effectively negotiates a QOS with the operating system. Even if this is available, processing a stream outside the kernel could still result in significant jitter. The use of an intra-stream synchronisation algorithm located in the incoming protocol stack is of little use in this scenario.

2.5.2 Stream Agent Role

As an alternative to the embedded approach for synchronisation, the *stream agent* concept is proposed. In this dissertation the term agent is used in general to describe any stream processing entity, implemented as part of an application, within an application level service, in the kernel or even in hardware. A stream agent deals specifically with synchronisation, encapsulating an elastic buffer and related activities as shown in Figure 2.9. *Independent scheduling* is a central feature of its design, appropriate in a system where resource management is based on specified application requirements. Each stream agent is typed, based on the stream format and sample encoding which it handles, and can communicate externally, allowing interaction with applications for control and feedback purposes. It operates according to application supplied parameters, which define a desired QOS to be met for elastic buffer output.

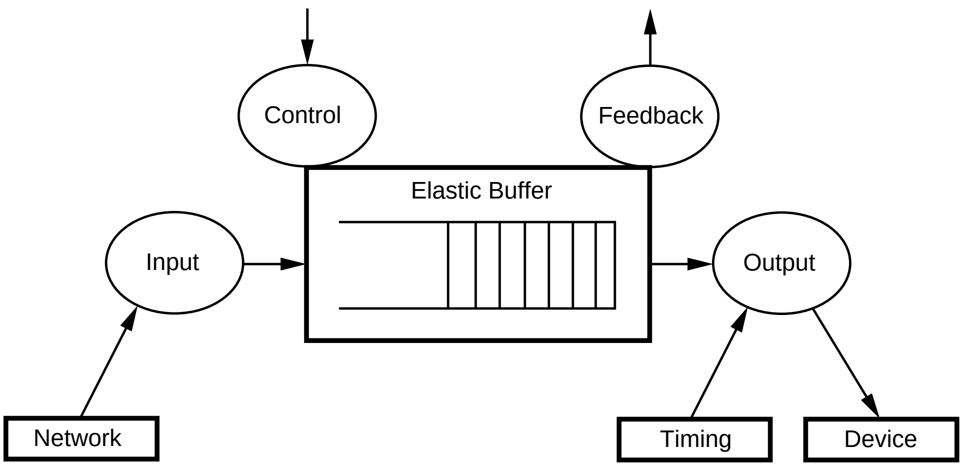


Figure 2.9: Stream Agent.

The term *end to end QOS* is commonly used to describe communication requirements for a stream, i.e. it governs behaviour between the point of leaving the source program to arrival at the sink program. In the embedded approach, this incorporates elastic buffer operation and hence presentation. This is no longer true in a system composed from scheduled agents, which handle continuous media outside of the communications facility for synchronisation and other activities such as compression and decompression. This is because the conventional use of the term does not encompass their behaviour, and thus is insufficient to fully satisfy application requirements. For example, consider the situation where a high QOS audio stream undergoes analysis by a sink based agent to allow silence detection. If agent scheduling is not properly controlled, the work of enforcing the end to end QOS will become undone, due to the introduction of unwanted jitter and perhaps losses.

Hence the end points must be extended to allow QOS requirements to be described in terms of behaviour between the source and sink *devices* as illustrated for a video stream in Figure 2.10. An application should negotiate a QOS with each agent and other components along the path of a stream, in a linked manner to guarantee such an end to end QOS. Individual component guarantees should only be approved after interaction with the supporting operating systems to ensure that adequate resources can be made available. Thus, a *synchronisation QOS* refers to the expected performance of a stream agent within the all encompassing device to device arrangements. This series of guarantees could be established using a mechanism similar to that proposed by [Ferrari90b], which incorporates a two-phase protocol to negotiate required performance with relays along the route of a stream.

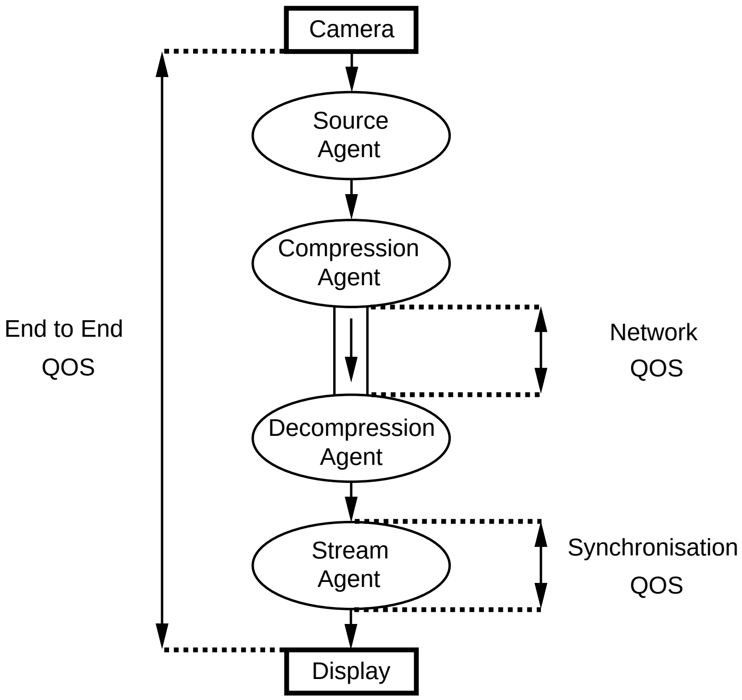


Figure 2.10: QOS between Devices.

Thus, intra-stream synchronisation is viewed as a service which augments network supported QOS guarantees in order to meet end to end QOS requirements. Within this service, stream agents are instantiated on behalf of applications to provide jitter smoothing which complements the jitter control function of the network. Failure to meet the agreed synchronisation QOS requirements is reported back to the application as an indication that the operating parameters have been exceeded. At this point, an application is at liberty to take steps to renegotiate the contract with the stream agent or some other stream handling components.

2.6 Stream Agent Design Issues

In terms of input, a stream agent accepts data from a source agent, which captures or retrieves a stream for transmission. An application wishing to present a stream, arranges for both source and stream agents to be created and a transport connection to be established between them. For output, a stream agent is connected to a device handler, perhaps via an intermediate agent for decompression or application specific processing. The behaviour of these agents must be carefully controlled to ensure that the work of the synchronisation agent does not become undone. Hence, a high grade *follow-on* QOS is essential.

Associated with each stream is a time dimension, known as its *stream time*, which is maintained internally by a stream agent and used to determine the mapping of sample presentation to real time. Its value at any point in time lies between the time stamp of the most recently presented sample and that of the oldest sample in the elastic buffer. Its relationship with real time is nominally linear, but may be varied for synchronisation purposes. Special effects such as increasing or decreasing the play-back speed are achieved by informing the stream agent of the new mapping so that it can react accordingly. It must also be made aware of *epoch* changes in the stream time stamps, caused by gaps introduced by editing or seeking within a file. Similarly, it must be possible to allow a reverse presentation by altering the direction of the mapping. This presents difficulties in dealing with data already in the elastic buffer and may be simplified by emptying it by presentation or flushing. When one considers the special effects described above, it is evident that interaction with the source agent is necessary in order to support changes in rate, granularity and direction.

A stream agent is created in response to a request from an application or service. Parameters to this request describe the stream type, the required synchronisation QOS and the expected input QOS. In most cases, a stream agent can supply default values for the above parameters which can later be re-negotiated. The supplied parameters should be evaluated and used to determine whether the synchronisation QOS can be satisfied. This requires examination of the available operating system resources. The reliability of an operating system guarantee, notably for CPU scheduling, determines the ability of the stream agent to maintain its synchronisation QOS. The failure of a stream agent to maintain the arranged QOS must be reported to the application as an exceptional condition. This can occur due to unreliable operating system QOS and/or inaccurate estimates for the QOS of the incoming stream. The stream agent provides a first level of handling to deal with these violations, but a longer term solution requires renegotiation with the operating system and/or application intervention.

Reporting QOS exceptions can be achieved by signalling the application. The urgency of the situation may be reflected in the timing constraints associated with the signal. An application response can take several forms:

- arrange better operating system service for the stream agent;
- relax the synchronisation QOS, for example, to accept greater jitter;
- renegotiate the QOS required from other components (see Section 3.4);
- cancel the stream presentation.

Exceptions should be generated in advance, if possible, so as to preempt the occurrence of a predicted problem and give the application additional time to react. This is made possible using knowledge about the short term future of the stream, gained from observation of elastic buffer length and operational trends. The default reaction to an exceptional occurrence is specific to the stream agent, but typically results in the provision of a best effort service. For video this could be achieved using a layered coding technique [Verbiest88] and degrading in a controlled manner as deemed appropriate in the prevailing circumstances.

Internally, the stream agent is composed of a buffer module and a number of paths of execution, known as *threads*. The buffer module implements the elastic buffer as a queue of stream samples ordered on their temporal position within the stream. The use of order preserving circuits for stream transport allows this to degenerate to a simple FIFO queue, simplifying its operation. As stream samples arrive they are received by the input thread and deposited in the buffer. The output thread commences operation once the qualifications regarding buffer length are satisfied. Its operation is iterative, arranging to be scheduled such that each sample can be removed and dispatched at the desired time. Consequently, the problem of inappropriate CPU scheduling can have a significant influence on its operation. Implementing scheduling so that jitter is minimised is particularly difficult when the operation of multiple output threads happens to be in-phase. It is advisable to detect this situation and respond by adjusting the phase of the streams. This can be performed in the manner described for clock drift in Section 2.4.

Housekeeping threads exist to deal with application control and to support feedback and exception signalling. Additionally, a frequency adjustment thread may be used to monitor the elastic buffer for the purpose of dealing with clock drift as discussed in Section 2.4. A stream agent is best implemented within a single address space, allowing efficient access to shared data structures, synchronised using conventional mechanisms such as semaphores.

2.7 CPU Scheduling Requirements

In this dissertation, it is generally assumed that threads exist as operating system objects and are scheduled directly by the kernel. An alternative scheme is to allow user level scheduling of *lightweight processes* with CPU time divided amongst different conventional processes. A hybrid proposal in [Govindan91], extends this approach with a mechanism for influencing kernel scheduling based on the needs of individual lightweight processes. Conventional algorithms such as FCFS and Round-Robin are unsuitable for scheduling threads which have real time operating constraints. Similarly, although the Rate-Monotonic algorithm is optimal for fixed period activities, the behaviour of threads within a stream agent is not regular. Even for a stream with a fixed transmission rate, the imposition of jitter means that arrivals do not occur at fixed intervals. In addition, execution of the output thread is not strictly periodic because it may occasionally adjust the mapping of stream time to real time. In order to support a stream agent the scheduler must comprehend the notion of urgency, so that behaviour is predictable in the face of contention. Thus a deadline and/or priority based algorithm such as the earliest deadline first is more suitable. Ideally, the operating system must be able to allocate its resources so that QOS requirements are not violated.

In a stream agent, the input thread needs to be scheduled in response to data arrival. Variations in this scheduling latency appear as an increase in communication jitter. The main requirement is to bound this value so the newly arrived data is not delayed unpredictably. In other words, the data must appear in the buffer prior to when it is required by the output thread, or otherwise losses may reach the point at which synchronisation QOS is lost. Another benefit of predictable variations in the input thread scheduling is to avoid confusing algorithms for clock drift detection (described in Section 2.4), which typically discriminate jitter from frequency effects based on a probability distribution for network delays [Almeida91]. In addition, it gives a false impression of the buffer length, restraining the scope for making adjustments. Accurate measurement of the arrival time could be made available to the stream agent by employing kernel time stamps, making it possible to determine the precise time of arrival, regardless of the input thread scheduling. This information can be used to determine whether excessive jitter results from activities prior to, or after arrival. Another possibility is to include this scheduling latency as part of the end to end QOS.

The deadline for an arriving sample depends on the current temporal position of the stream. The slack time is related to the jitter which the sample encountered, but this information is not readily available within the kernel. For the output thread the use of a *work ahead* scheme is unlikely to be very beneficial, as output operations cannot be provided well in advance of their deadline. A more reasonable approach is deadline scheduling. For a stream with fixed rate output, the thread is nominally periodic. The input thread never operates with a fixed period due to jitter in the arriving stream, but it could usefully be scheduled based on aperiodic deadlines. The importance of a thread should be related to its ability to miss a deadline without upsetting its QOS, based perhaps on acceptable slip. Further research is needed to allow evaluation of the cost of missing deadlines in this context.

There are also time constraints associated with the other threads, notably those for reporting exceptions (see Section 4.5.2). Scheduling access to the buffer may require the use of a priority inheritance algorithm, although a policy of avoidance may be preferable, given that all accesses are bounded. The general need for both deadlines and priorities is recommended, but they do not provide a complete solution. Further research is required to determine a method of translating the QOS requirements of a stream agent into CPU scheduling demands which can be guaranteed.

2.8 Stream Agent Location

The conclusion drawn from Section 2.5.1, is that, in comparison with traditional embedded approaches, an integrated solution offers much more flexibility for synchronisation and other stream processing. Advances in the speed of workstation bus and Direct Memory Access (DMA) may encourage this direction to be followed. An attractive hybrid architecture allows a stream to be routed directly to an output device on a separate sink host, while maintaining a reasonably high degree of control over its operation. This avoids traversal of the workstation bus in cases where the sink is capable of performing the desired stream processing. The sink may exist with its own network interface, possibly controlled over a direct link from the workstation. Alternatively, it may run as an I/O processor within the workstation, with data transferred over a separate access channel after demultiplexing at the network interface [McAuley89]. This is similar to the multi-media co-processor of [Bulterman91] which intends to allow stream processing to be off-loaded from the main workstation processor, while reaping the benefits of partial integration. This section examines stream agent operation in this type of environment, with particular emphasis on retaining the flexibility which is possible in a fully integrated approach.

2.8.1 Distributed Operation

A stream agent can be decomposed and its functionality distributed over the sink and a controlling host. This is suitable when the sink is relatively simple, perhaps only able to support device access, communications and basic maintenance of a stream data buffer. By insisting on relatively tight coupling with the sink, this allows the controlling host to make synchronisation decisions and direct accurate data presentation.

Internally, a stream agent has an structure in which the input and output activities can be viewed as modules which used the services of a buffer module. A buffer module provides an interface for managing an elastic buffer, supporting operations to release samples and adjust buffer length. These are invoked by the output module to maintain synchronisation, based upon an accurate representation of the buffer state which is maintained by the input module. Buffer operations must be delivered with a small and fixed delay so that samples are presented without introducing jitter. This general arrangement is similar to the use of an application level stream agent, which may have access only to state information about streams which are handled in the kernel. A distributed stream agent is illustrated in Figure 2.11.

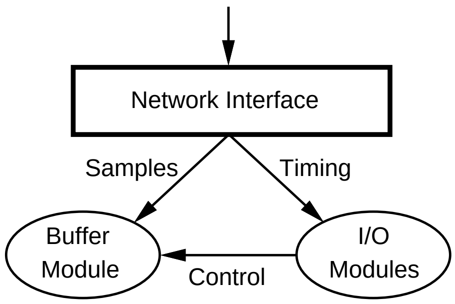


Figure 2.11: Distributed Stream Agent.

The input module can obtain information describing the buffer state in one of two ways. As stream segments are delivered to the sink, their timing information may be extracted and made available to the input module with a bounded latency. Alternatively, it may be arranged for this information to arrive at the input module directly. In order to ensure correspondence with the arrival time of the related data at the sink, stream timing and data should be split and delivered with approximately equal delay to the input and buffer modules respectively. In essence, a distributed approach allows the use of a relatively simple sink, while retaining the flexibility to influence synchronisation.

2.8.2 Remote Operation

If adequate control of synchronisation is not possible using a distributed stream agent, a remote approach can be adopted. In this case, the stream agent runs on a controlling host (possibly the workstation) and obtains an onward connection to a sink which has a high grade QOS. The characteristics of a local ATM network should support this configuration, without having to make excessive resource reservations and incur associated real costs.

This approach is particularly suitable for the novel architecture of the Desk Area Network (DAN) [Hayter91]. This uses an ATM network as the interconnect for a multi-media workstation allowing the use of multiple high bandwidth streams without significant contention. Devices exist as nodes attached directly to the switch as illustrated in Figure 2.12. Similarly, facilities such as synchronisation and decompression are viewed as shared networked services within a DAN. In conventional architectures, lack of bus bandwidth generally forces decompression to be performed on the same physical board as the device [Luther89, Szabo91], requiring hardware which is dedicated to each workstation.

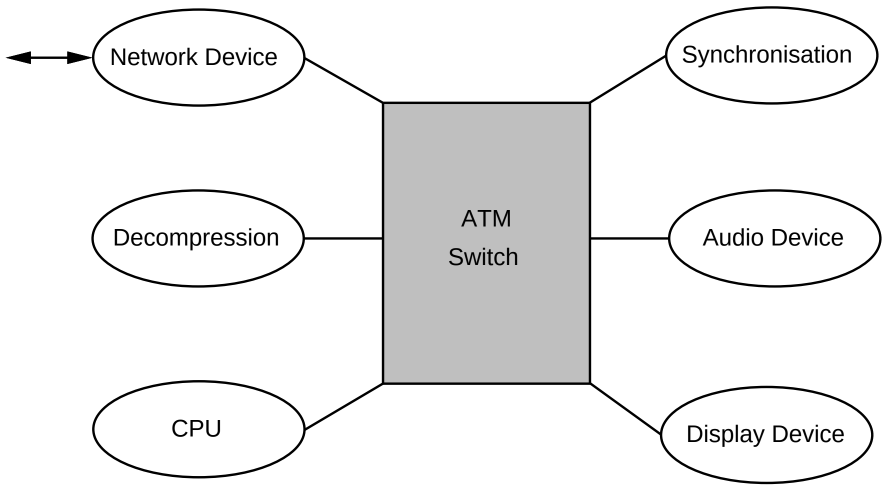


Figure 2.12: The DAN Multimedia Workstation.

The local nature of the environment provides highly predictable communication delays, allowing a stream agent to operate in a remote, or in fact, a distributed form. In the remote form, a stream travels to the sink device after passing through a synchronisation and possibly decompression node. The distributed approach allows the stream to bypass the synchronisation node by operating the buffer module at the sink device node and controlling presentation from a stream agent located across the switch.

2.9 Synchronisation Service

It is proposed that intra-stream synchronisation is best supported by a set of *synchronisation servers* which function as stream agent managers. These are collectively referred to as a synchronisation service and may be located at sink nodes and/or remote sites across a predictable network. They offer considerable flexibility to applications, both in terms of selecting stream agent types and describing synchronisation QOS requirements.

A synchronisation server may be used in a two stage configuration. In this way a remote server is used in combination with a sink based stream agent as illustrated in Figure 2.13. The main use of this arrangement is in the situation where the latter has limited resources and hence is unable to handle the expected jitter. In particular, it is common for embedded sinks to allocate fixed amounts of per-stream buffering, based on a standard estimate for local jitter. Operation in a wider area can be achieved in a relatively straightforward manner with this two stage arrangement, by the ability to distribute the resource requirements. The alternative is to try to extend local QOS requirements into an inter-network.

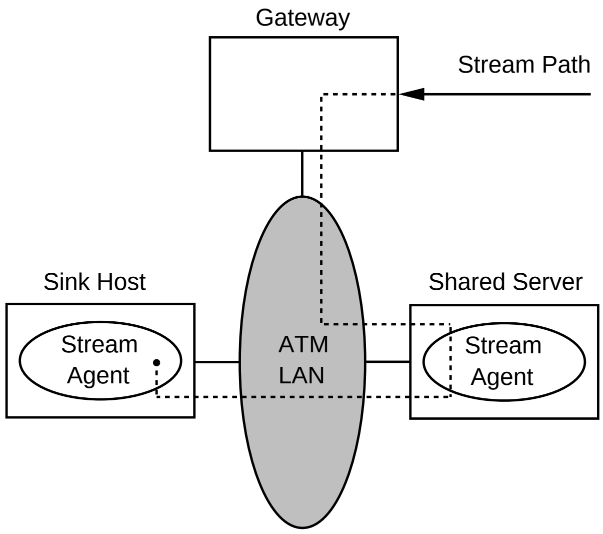


Figure 2.13: Two Stage Synchronisation.

The synchronisation service approach allows an application to balance support for its QOS requirements between the inter-network and destination network. This is compatible with the proposal in [Partridge91], which advocates the use of significant quantities of buffering near the sink to support communications protocols which provide guarantees on delay, but not jitter. A multi-stage approach is proposed in

[Ferrari91], which makes use of gateways along the route of the stream to maintain the same QOS at each stage. This approach reduces the amount of jitter which must be dealt with at the destination, but complicates the operation and design of network relays.

The recommended arrangement is that a *shared server* is made available for networked sink devices in a local ATM environment. This relieves a sink from the need to provide sufficient memory and processing resources to support synchronisation. It also allows the machine running the server to be specialised, for example, by having significant quantities of memory, along with multiple processors and network interfaces. Additionally, it may be allocated an unfair share of network bandwidth and operate a suitable real-time scheduling algorithm. The costs associated with this machine can be offset by the corresponding reduction in sink resources. A feature of a shared server is that it acts as a focal point for streams within a local area and as such facilitates synchronisation between streams. An important issue for successful operation is dealing with the CPU utilisation and scheduling for multiple concurrent stream agents. A drawback of a shared server proposal is the increase in network activity caused by having to route a stream via the server host.

A shared server is not the most suitable arrangement for use with a conventional workstation, because it is difficult to provide a high grade QOS connection to the output device for synchronised data. Even with simple sink devices, any buffering along the path can introduce jitter. In particular, it is likely that synchronisation may be done before decompression in order to conserve memory and bandwidth, in which case the decompression agent must be prepared to offer very good QOS guarantees. Naturally, stream timing information must remain accessible from the decompressed stream in this arrangement.

2.10 Summary

This chapter presented a highly adaptable solution for the support of intra-stream synchronisation. Sources of stream delay both in the network and end-systems were identified. Elastic buffer mechanisms were described with particular emphasis on their use in an ATM environment. Pragmatic solutions to the problems caused by clock drift were discussed. Embedded approaches to synchronisation were criticised and as an alternative the stream agent was introduced. Issues related to the location, design and operation of stream agents were investigated and CPU scheduling requirements discussed. Finally, the synchronisation server was proposed as a stream agent manager and the benefits of operating a shared synchronisation server in a local ATM environment were discussed.

Chapter 3

Inter-Stream Synchronisation

The subject of this chapter is synchronisation between streams of continuous media. Factors which influence its support are described, followed by a detailed discussion of the proposed design.

3.1 Introduction

By definition, a multi-media presentation can comprise data from two or more types of information media. Its co-ordination is based upon the occurrence of certain stimuli, representing events such as the passage of time and user interaction. At the application level this behaviour is commonly described using scripts or time-lines. An important way in which synchronisation manifests itself is between streams of continuous media which are to be presented concurrently. This inter-stream synchronisation differs from behaviour which is dependent on discrete stimuli due to the continuous, time based nature of the requirements. This chapter proposes a design to perform inter-stream synchronisation, while Chapter 4 will discuss non-continuous stream related synchronisation.

The most common inter-stream requirement comes from the fact that video is accompanied by audio for the majority of applications in which it is employed. Typically this relationship is established during a live audio-visual performance, but it may also be introduced artificially, for example, by the addition of a music sound-track to a movie. The aim of synchronisation is to maintain the correspondence between the time dimensions of related streams throughout their presentation. *Skew* is defined as the difference between the stream time systems and is qualified in terms of a maximum bound which is acceptable to the application. Selecting this value is influenced by several factors including data sampling rates, encoding and compression as well as quality related factors like video image resolution. Results of experiments reported in [CCETT88], suggest that skew between audio and video should be bounded to 150ms. Recent experience with the Pandora system indicates that skew greater than 80ms[1] can be noticeable, while a range of -20ms to 40ms between audio and video is suggested in [Leydekkers91].

An important factor in choosing a skew bound can only be provided by the application, as it is a description of the stream content and usage. For instance, a music sound-track is likely to tolerate a greater amount of skew than a sound-track which contains, for example, a gun-shot with accompanying visual effects. Synchronisation between streams of the same media is also important. The bound is usually very small for stereo audio and situations where several video streams provide a multi-angle view of a high motion activity such as sport.

The next section examines source-based synchronisation and concludes that an alternative sink-based approach is considerably more flexible. Section 3.3 discusses the problems of getting streams initially synchronised and leads to the subsequent section which deals with maintaining synchronisation. Section 3.5 is a detailed discussion of the proposed design for inter-stream synchronisation.

3.2 Source-Based Models

Source based operation provides synchronisation at or near the source, before the data of each stream is transmitted. This has the benefit of not having to introduce complexity at the sink, proving attractive for relatively simple devices such as televisions. The fundamental assumption is that subsequent delays are approximately equal for each stream. However, this is not always valid in a distributed system, with the result that a synchronisation relationship which is present at the source may be lost upon arrival at the sink.

3.2.1 Interleaved Communication

An important technique which can be used to ensure that inter-stream relationships are maintained during transport is known as *interleaving*. This works by multiplexing data from each related stream to form an aggregate stream. In essence, this data is grouped to indicate a simultaneous presentation requirement [Sventek87]. This aggregate stream is sent over an order preserving communications channel from source to sink as illustrated in Figure 3.1. This could be implemented using a single

[1] Information supplied by G. J. Stark of the University of Cambridge Computer Laboratory.

transport level virtual circuit, with interleaving performed at a higher protocol level (probably the session layer) or in user space. Alternatively, the transport protocol may support a *bundle* of virtual circuits. Within a bundle, each circuit traverses the same path and the data input sequence is reproduced at the destination. It is proposed in [Dixon91] that this could be implemented using a B-ISDN Virtual Path, which provides a suitable facility to support signalling for a collection of Virtual Channels [CCITT88]. In general however, this type of protocol support is not common due to the complexity which it introduces.

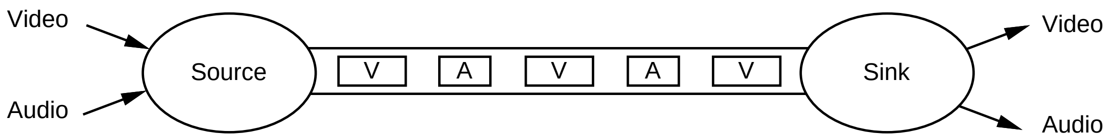


Figure 3.1: Interleaving for Synchronisation.

While interleaving is a simple technique it has several drawbacks which make it unattractive in general terms. A potential conflict exists when multiple streams with different QOS requirements are multiplexed over a single transport connection. For example, less reliability may be acceptable for video but not for text. Thus, it becomes necessary to use the strictest combination of QOS parameters [Leung90], resulting in the inefficient use of network resources. A non-interleaved approach allows per-stream QOS to be selected and can exploit parallelism, giving considerable freedom in the choice of network, route and protocol for each stream.

Interleaving also makes it difficult to dynamically add or remove streams during a presentation and to employ hardware which is media specific. In regard to intra-stream synchronisation, the task of performing jitter smoothing is complicated by having to deal with an interleaved stream. The very act of interleaving is also likely to introduce jitter, through the buffering and delays associated with multiplexing and demultiplexing. Flexibility is reduced for performing inter-stream synchronisation and, as pointed out in [Nicolaou90b], the ability of the application to influence and possibly relax skew constraints is removed.

3.2.2 Source and Sink Constraints

The benefit of interleaving is lost if pre-multiplexing and post-demultiplexing delays can cause a loss of synchronisation. As a consequence, it is effectively required that streams emanate and are multiplexed at a single source node, with an equivalent restriction on the sink. In addition, it is important to control any difference in delays at these nodes, caused for example by resource contention or an elastic buffer. This stipulation that related streams travel between a given source/sink pair is very limiting. There are several possible source to sink arrangements as illustrated in Figure 3.2. The common case, shown in (a), satisfies this constraint, fitting the situation where devices are fully integrated into the workstation architecture. The arrangements in (b),(c) and (d) deal with the use of devices having a network attachment which is distinct from that of a workstation. Examples include dedicated systems like ISLAND telephones [Ades87], as well as the ATM devices of the DAN system [Hayter91].

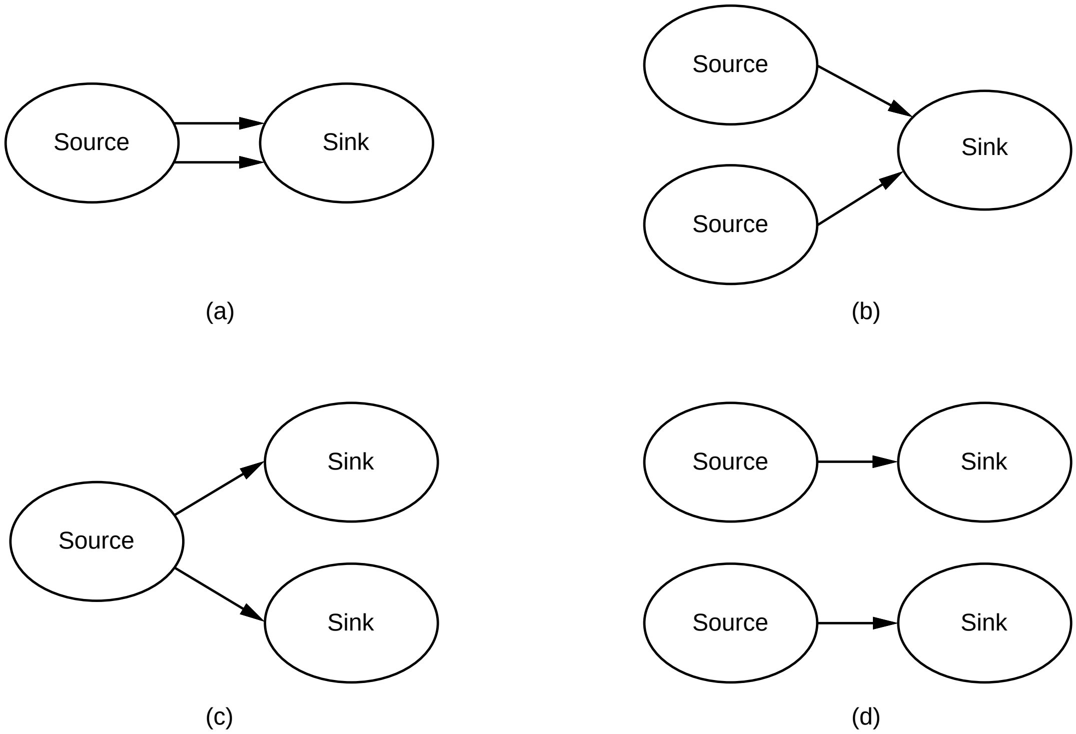


Figure 3.2: Source to Sink Organisations.

Typically, these disjoint devices offer less application control than an integrated solution, but they offer several valuable benefits which assure their continued use. Firstly, they increase independence from the workstation, allowing greater flexibility in selecting a device, its location and portability. For example, the system described in [Rangan91] employs a PC to deal with continuous media, since the DVI hardware being used is not available for the target workstation. Secondly, since they are usually cheaper than a workstation, although more expensive than an integrated solution, the potential for a large scale of production exists. Thirdly, as discussed in Section 2.3, the use of a dedicated host in this way serves to alleviate the workstation of resource contention and management problems. Finally, both the hardware and software of a disjoint device can be made much more reliable than that of a workstation. Thus, it can be arranged that a disjoint telephone may still be used, even during the workstation failure. Therefore, it is concluded that synchronisation must be possible for any source/sink arrangement to support operation in a heterogeneous environment.

3.2.3 Centralised Operation

The flexibility to support multiple sources can be supported by providing a central host which multiplexes those streams converging on it. This has a matching demultiplexer at the other end of the connection, from which streams diverge to their respective destinations as shown in Figure 3.3. This scheme is similar to the use of a central controller to serialise updates to a shared workspace [Sarin85] and to combine individual voice streams for conferencing [Ades87]. In these cases, the amount of communication can also be reduced by eliminating the need for a direct link between each participant.

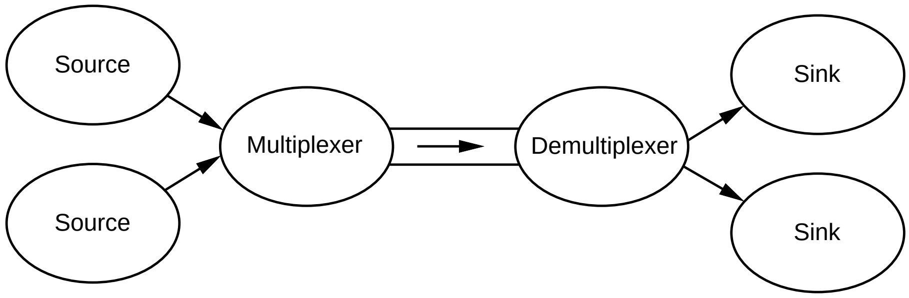


Figure 3.3: Centralised Approach.

However, this approach is inappropriate for inter-stream synchronisation. A loss of synchronisation results if streams encounter a delay difference *en route* to the multiplexer which exceeds skew bounds. This can be resolved using a synchronisation mechanism prior to interleaving. However, this is equivalent in function to that which is necessary at the destination when interleaving is not used. Hence this scheme simply shifts synchronisation to the multiplexer. At the demultiplexer, the skew caused by differences in the onward travel delays are also problematic. For hosts on a LAN, it should be possible to avoid significant delay differences. But this restriction on location is unreasonable and if imposed would yield a system with non-optimal routeing and bandwidth usage. In addition to these drawbacks, contention at the centralised nodes could be significant. For these reasons the use of a centralised organisation is not advocated in this dissertation.

3.2.4 Storage

Interleaving is also used for multi-media storage. In that context, it offsets the physical limitations on device bandwidth and mechanical seeking latencies for storage media such as the CD-ROM. Without interleaving, seeking activities may result in thrashing, causing excessive jitter and a loss of synchronisation.

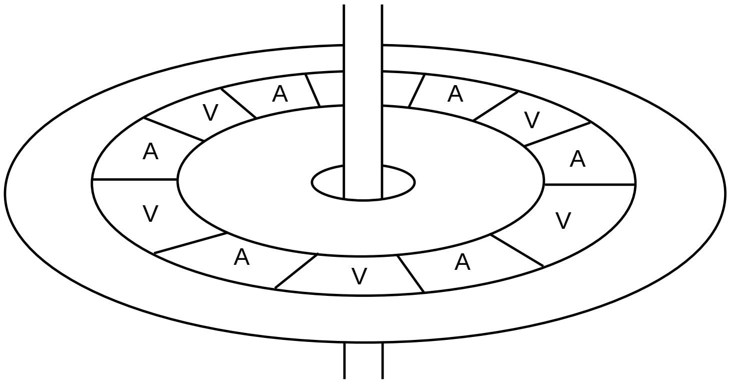


Figure 3.4: Interleaved Disk Layout.

Consequently, it is necessary to carefully plan the layout of data on the disk as illustrated in Figure 3.4. Algorithms which aim to provide an optimal layout for audio on a CD-ROM are given in [Yu89]. The file format used in the DVI system for CD-ROM is composed of a sequence of blocks which may contain data interleaved from a collection of video and audio streams [Luther89]. The DVI Audio/Video Support System (AVSS) also allows still images, application data and real-time *underlay* data to be incorporated into an interleaved file. A similar approach is used in the MPEG standard which aims to be both application and device independent [Szabo91]. Both of these schemes currently use hardware support for interleaving. An alternative to interleaved storage is to employ the method used in the conventional analog Video Cassette Recorder (VCR), where video and associated audio are recorded on individual tracks and played back using physically separate reading heads. In this case, transmission delays are not an issue, but jitter caused by mechanical head drum motion is corrected using a Time Base Corrector (TBC) in high quality machines. This track-based approach is not suitable as a general solution due to the complexity and expense of multi-head disk systems and the inability to scale the scheme for additional streams.

Although the use of interleaved storage is an essential element for current CD technology, the approach is only suitable for pre-recorded material and in the situation where concurrent access is not required. It is restricted by a layout format which is derived from characteristics of the storage device. Thus, for example, files in the DVI system often contain dummy data, known as *padding*, to ensure that an average retrieval rate is maintained and seeking can be avoided. These constraints complicate editing, modifying the presentation and gaining access to individual streams. Similarly, sharing data between several presentations is not possible and requires individual copies to be kept.

Despite developments in CD technology which are likely to significantly decrease seek latency, it is expected that interleaved formats will still be used. This is because they allow synchronisation issues to be encapsulated with the data and dealt with at production time, rather than during presentation. In a more dynamic environment using magnetic disks, streams are more likely to be stored separately as in [Jardetzky92, Hopper90], thus providing considerable flexibility in comparison. Issues related to the support of synchronised retrieval using multiple storage systems for multi-media presentations are outlined in [Sreenan90].

This dissertation encourages the use of a non-interleaved scheme and does not impose constraints on the use of multiple stream sources and sinks. However, it recognises the need to support interleaving and can do so because of the flexibility in a system comprised from stream agents, as discussed in Section 2.5.2. Interleaving is underpinned by viewing it as a particular type of stream format which can be dealt with by a stream agent of an appropriate type.

3.3 Achieving Synchronisation

This section addresses the task of attaining synchronisation without relying on the use of interleaving and when multiple sources and sinks may be involved. Attention is first turned to initial synchronisation in the face of equivalent end to end delays, while Section 3.3.2 looks at the wider problem which becomes evident when this supposition is withdrawn. For reasons of clarity, synchronisation between a pair of streams is considered, although the analysis extends naturally to multiple streams.

3.3.1 Initial Co-Ordination

Achieving synchronisation between streams is determined by control of two variables: the relative real times at which they are started and the subsequent end to end delays which are encountered. If the latter can be bounded to within the desired skew requirements then a synchronisation relationship present at the source devices is reproduced during presentation. This discussion temporarily assumes that individual streams are, in fact, guaranteed equal delays.

3.3.1.1 Live Streams

For live streams, if there is a difference in the real times at which the data began to be captured for each stream, then it should be maintained during the presentation. A desire to commence stream presentation simultaneously can be satisfied either by hiding this difference or using synchronisation to eliminate it. Hiding is achieved by discarding an appropriate amount of data from the earliest stream. This can be performed at either the stream source or sink, but requires access to information telling when the other stream is ready. The situation is illustrated in Figure 3.5, where a single clock is used to stamp the time on each sample as it is captured. Capture on stream A is started at time 2 and followed on stream B at time 4, with the result that presentation nominally begins at times 7 and 9 respectively. If this difference is to be hidden, then either the earliest stream transmission should be at time 4 when capture for both streams has commenced, or presentation should be deferred until time 9 when both streams are arriving at their sinks. The sink based approach has been adopted for the storage facilities of the Pandora system [Hopper90, Jardetzky92], where recording begins only when the audio and related video stream are both being received.

An alternative scheme is to synchronise streams so that their capture begins at approximately the same real time. This could be achieved by pre-scheduling the operations based upon a view of time which is shared between the sources. This common time system could be realised using a global clock or based upon knowledge of source clock differences as described in Section 2.2.4. The fact that related live sources are likely to be in close proximity serves to increase the accuracy which may be achievable in this respect. Commands must be submitted far enough in advance in order to allow the sources to carry out any preparatory work. The sources may then be able to make an advance reservation for associated resources or alternatively, submit allocation requests a safe period ahead of the time, suffering a degree of resource wastage. In the latter case, the possibility of having to deal with time-outs for idle resources must be considered if the chosen period is too lengthy.

A prepare and act arrangement achieves this aim without relying on a common time system [Arons89, Schmandt88]. In this scheme, sources are asked to prepare in advance of the start time, allowing resources to be allocated. The aim is to reduce the latency of the subsequent act command, which in this case is used to initiate data capture. Once again, in order to avoid excessive resource wastage it is recommended that the time between the prepare and act operations is made as small as possible. The accuracy of the synchronisation which can be achieved is determined

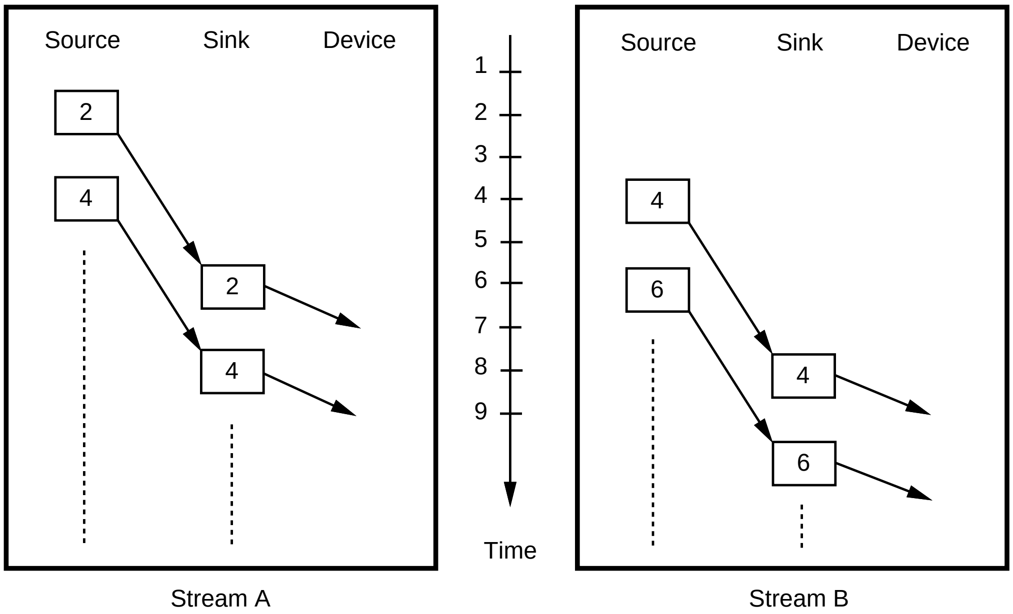


Figure 3.5: Initial Synchronisation for Live Streams.

by the latency of issuing, delivering and reacting to the act command. It may be possible to bound the difference in these latencies for each stream through the use of appropriate QOS arrangements in the network and end-systems, supporting a real-time RPC or message passing protocol. Time-based message passing is employed in several real-time operating systems, including those for the support of process control [Damm89] and computer music [Puckette86]; while a proposal for a real-time RPC protocol is discussed in [Li91]. For a single source, the prepare and act commands could be incorporated into a single atomic operation which would wait for all streams to be primed before commencing synchronised operation.

3.3.1.2 Stored Streams

For streams of stored data the problem is different, in that there may be inaccuracies in starting the retrieval of related streams simultaneously. This problem is also relevant in situations where it is desired to start streams with some intentional time difference between them. Hence, a synchronisation mechanism is necessary in order to ensure that the intended real time relationship between streams is reproduced during play-back. This relationship may either be an inherent attribute of the original recording or created artificially by combining streams which were not captured together. Attempting to meet this aim by simply synchronising related play back commands is prone to inaccuracies, mainly from delay variations in storage facility operation for each stream. This has particular significance when storage devices with slow and/or unpredictable access times are used. Common time system and advance preparation solutions can alleviate this difficulty, or alternatively, sink based buffering can be used to eliminate skew, based on knowledge of the intended real time difference.

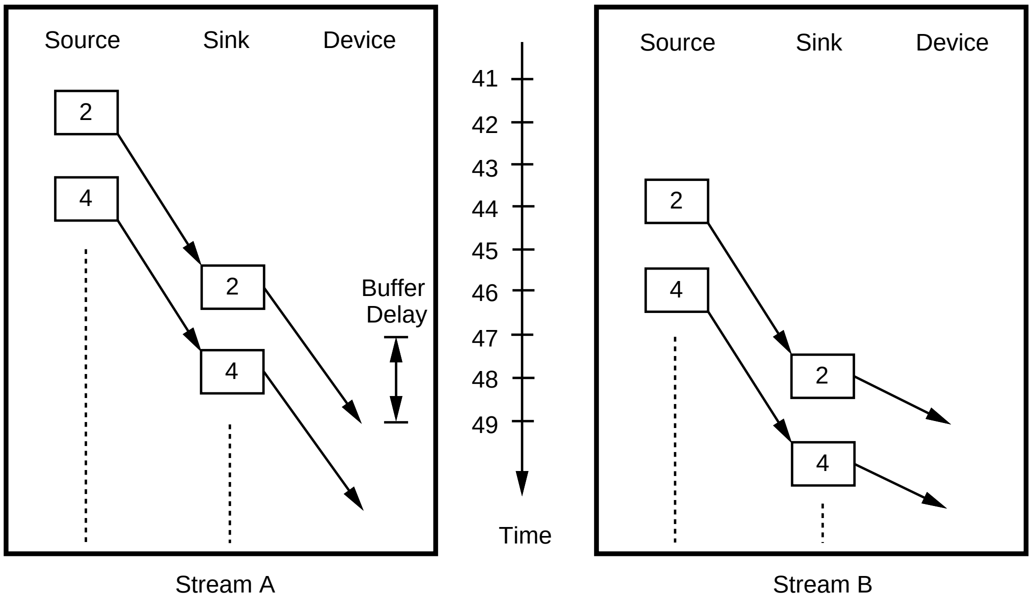


Figure 3.6: Initial Synchronisation for Stored Streams.

An example of this scheme in operation is given in Figure 3.6, where the earlier stream to arrive is delayed until the presentation of both streams can be started simultaneously as directed by the timing information. In this case stream A encounters two units of buffering delay to ensure that presentation of both streams commences at time 49. However care must be taken in using this algorithm to ensure that this delay does not cause the stream end to end delay tolerance to be violated, although for stored streams this problem has less urgency because of the larger latencies which are usually acceptable. The possibility of buffer overflow is also a factor to be considered when trying to eliminate large skew values using this method. In essence, this approach can only be considered when both streams are to be started simultaneously. However, as already mentioned, this will not be the intention if they were captured at different times or are coupled artificially. This relationship can be described explicitly by supplying out-of-band information or alternatively through the use of stream timing information.

There is an alternative to waiting until all related stream are arriving before performing synchronisation and commencing presentation. This operates over an interval after play-back has commenced. It uses a feedback loop to suggest adjustments which are applied in an adaptive manner so that skew is eventually eliminated. Skew is calculated as the related stream begins to arrive, taking account of any intended start difference between streams. If its value exceeds tolerable bounds then it is dealt with by a combination of an immediate buffering delay and/or the subsequent activities of an adaptive synchronisation algorithm.

3.3.2 End to End Delays

A difference in end to end delays for related streams also contributes to skew. With an ATM network and well behaved end-systems the best scenario is where QOS arrangements can be used to make sure that delays are appropriately bounded. In the absence of this support, it may be possible to engineer a system whereby skew is nearly always bounded by a value which is acceptable to applications. To date, the Pandora system has been able to take this approach because of basic features of its design and environment. Operation is within a single ATM LAN which provides low and highly predictable latencies, and avoids the delays associated with relays and different routes. Related streams always travel between a single source/sink pair, as in Figure 3.7; whilst skew caused by the dedicated end-systems is relatively small and effectively bounded. The skew which occurs does not exceed that which can be tolerated by the current set of applications.

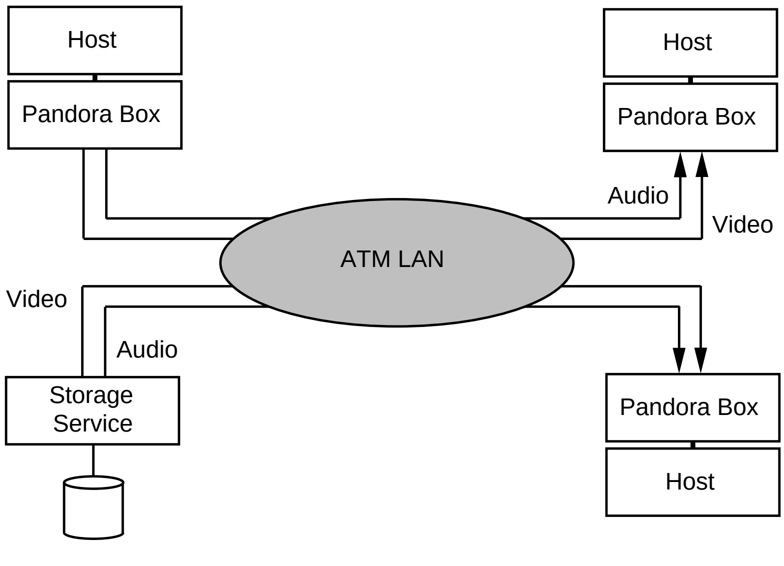


Figure 3.7: Stream Paths in the Pandora System.

3.3.2.1 Origin of Delay Differences

In a more general inter-network setting, supporting a wider range of applications, adopting this approach will not provide an adequate solution. This is mainly due to the greater probability of different stream delays occurring, both in the network and at the end-systems. In particular, the use of multiple sources and sinks for related streams results in different propagation delays. This also increases the scope for using multiple routes and hence incurring different queueing delays as well. In the network, if ATM connectivity is available from end to end, it is likely that skew of the order discussed in Section 3.1 will be the exception. However, end-systems and any intermediate nodes, such as those offering a decompression service, could cause excessive skew unless appropriate guarantees can be negotiated.

Each stream passes through a collection of components in the path from source to sink, accumulating delay at every step. The minimum possible delay is determined not only by the network but also by the end-system and service nodes. These latter sources of delay currently dominate in terms of an integrated system. For example, related streams may travel from source to sink via different subsets of available device handlers, application code and system services such as those for window management and synchronisation. Consequently, the difference in the delay which streams encounter will mainly be determined by the software which handles it, increasing the probability that related streams are skewed upon arrival at the sink device as illustrated in Figure 3.8.

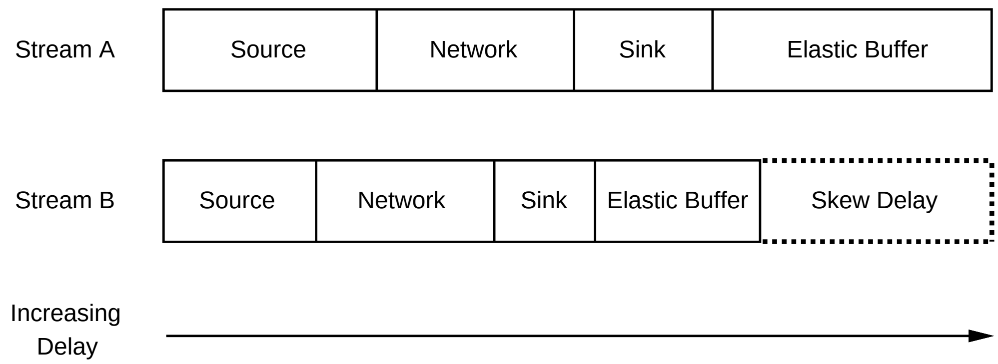


Figure 3.8: End to End Skew.

For example, in the Pandora system, delays from capture to transmission for video can be up to 40ms greater than those for audio[2]. Should the video stream subsequently incur a delay of similar magnitude, then tolerable skew bounds could be

[2]Information supplied by G. J. Stark of the University of Cambridge Computer Laboratory.

violated without having exceeded typical values for overall latency. A similar observation is reported in [Jeffay91], where video is delayed by up to 230ms, mainly because of the pipelined frame processing of DVI hardware.

Of course, it may be possible to avoid excessive skew by arranging comprehensive QOS guarantees. This requires stream delays to be normalised and can be achieved by using additional buffering, possibly distributed amongst the elements in the path of each stream. Thus, data is buffered if, after processing, it is ready to be dispatched at a time which would violate the minimum delay of the negotiated QOS for that element. This is similar to the operation of the protocol described in [Verma90], where each relay node on the network route operates a time ordered buffer for outgoing data, in order to ensure that jitter bounds can be guaranteed. Having suitable QOS is very important for interactive applications which demand that delay, and hence jitter, are low and bounded. This has the effect of curtailing the scope for large differences in end to end delays. If it is not possible to normalise stream delays using QOS arrangements then an alternative is to consider extending the delay experienced by a stream within its elastic buffer.

3.3.2.2 Dealing with Skew

Attempting to deal with skew at the source relies heavily on very reliable QOS support to control end to end delays. A sink based approach allows greater flexibility in adapting to the prevailing circumstances. It can also take advantage of the fact that it will be common for sinks to be physically adjacent in this case, facilitating higher accuracy through more predictable latencies in performing synchronisation.

The determination of a compensating delay for synchronising a stream is influenced by two variables: end to end skew and for stored streams, the skew which may exist in the source start times. It may be possible to assume that the latter variable is eliminated by source and/or application synchronisation. This ensures that streams are dispatched at real times which match the intended relationship between their start times. Hence, end to end skew can be deduced from the difference in arrival times, taking any initial losses into account. It should be noted that undetected losses in this case can lead to violations of stream latency constraints. An alternative approach is to factor out end to end skew by using QOS guarantees, allowing any initial skew to be easily detected.

The favoured solution is to allow the sink to evaluate skew by comparing stream timing information, regardless of its origin. For stored streams, it is important that this information gives the intended difference, if any, between start times, rather than the real times at which retrieval commenced. Based on the arranged QOS it may be possible to derive an estimate for the expected skew upon stream arrival at the sink. The ability of the sink to support initial synchronisation based upon this value can be determined by taking elastic buffer parameters into account. Hence the application must determine a balance between jitter, latency and skew in order to meet its synchronisation needs using an elastic buffer.

3.4 Maintaining Synchronisation

Once the presentation of related streams has commenced, the need to maintain synchronisation is evident. The following sections discuss the possible causes of synchronisation loss, issues in the detection of skew and possible remedies. The general approach which is advocated takes advantage of the predictability of ATM network operation and the high accuracy of modern clock oscillators. In this environment, it is suggested that skew can usually be handled by small scale adjustments to elastic buffers.

3.4.1 Basis

In Section 3.3.1.2 it was explained that the success of the initial skew elimination process is dependent on the ability to employ suitable length elastic buffers. Skew which remains after this point should be removed by making adaptive buffer delay adjustments spread over a subsequent period of time. During the presentation, skew can occur due to relative changes in stream end to end delays, operations which affect the progress of stream time, and exceptional conditions.

Changes in stream delays can result from elastic buffer operation, notably the mechanism for clock drift and dynamic response. Buffer length variations result in an altered relationship between stream time, as derived from time stamps, and the passage of real time. The effects of advancing stream time by skipping, and retarding it by the insertion of a pause are illustrated in Figure 3.9. In addition, deliberate changes in the performance of the network or synchronisation facility can be caused by an application which uses *re-negotiation* to dynamically alter its QOS requirements. For example, it may be desired to reduce the QOS for a video stream when the associated workstation window is obscured, thereby allowing better resource management and perhaps even cost savings. The existence of a re-negotiation facility at the OSI transport level is advised in [Berrocal92]. However, analytical work in [Crosby92] questions the benefits of re-negotiation over the static approach, where a network QOS is arranged only when a connection is instantiated. A conclusion of that work is that re-negotiation is not useful when the ratio of source peak to line rate is small. Hence, the use of re-negotiation in a LAN is not recommended, since under-utilisation is common in that context. Further research is needed to determine its value in a wider area context.

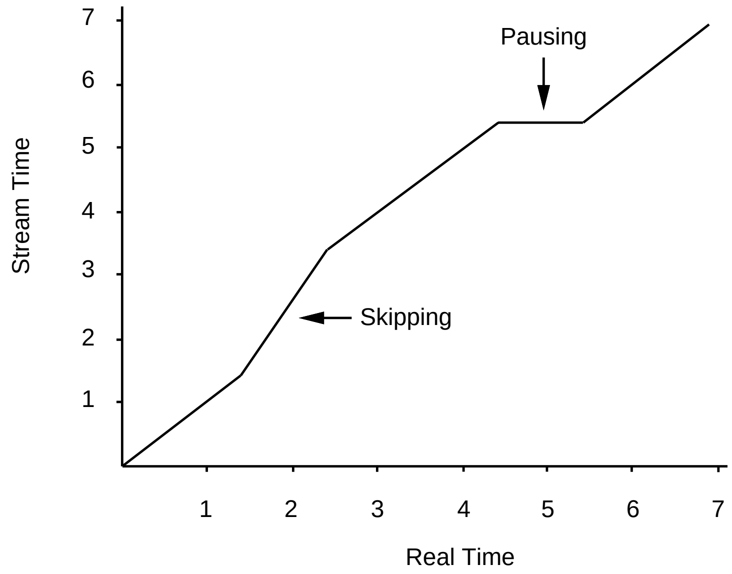


Figure 3.9: Alterations to Stream Time.

Stream operations can also result in excessive skew. This happens when the rate, position or direction of stream time is intentionally modified. Similarly, the failure of a stream is viewed as a cause of increasing skew.

3.4.2 Approach

The most straightforward method for skew detection assumes that source clocks operate with a known frequency difference, preferably zero. This allows stream timing information to be used for accurate inter-stream comparisons. Alternatively, it is possible to factor out frequency differences by using a common logical clock to generate time stamps, providing a reliable expression of the ongoing real time relationship of samples within and between streams.

Using a single source but separate sinks, the possibility of the latter operating at unequal and variable frequencies arises. These circumstances may be dealt with by making adjustments at the sink which either cause it to converge to the source frequency or equivalently maintain phase synchronisation as described in Section 2.4. Therefore the correspondence between time stamps is not disturbed in this situation.

If sources do not share a common frequency the situation is complicated because the continuous real time relationship between stream data is not given directly by the time stamps. Skew which occurs in this case can only be determined with knowledge of the frequency differences and variations. By having each sink adapt to the rate of the source but keep track of its deviance from a clock which it shares with the other sinks, the frequency difference between the sources can be derived. This can be used in combination with the values for time stamps at the point of initial synchronisation to work out the real time relationship between the stream time scales.

If time stamps are stored along with the stream, this algorithm loses its appeal, because a mismatch between the original source clock rates may not be reproduced unless it is explicitly noted during the recording. Thus, operation without a common frequency can lead to skew, if recorded stream time stamps incorporate elements of drift which are not reproduced during play-back. It may be possible to measure the frequency difference between the source clocks and that at the storage facility. But if related streams are stored at different sites, this requires a common time reference at the sinks in order to allow rate comparisons in the manner described above. Alternatively, if a common frequency is available at the sources, the real time correspondence between data from different streams is present in the recording. Hence, during play-back, a shared clock at the storage locations can then make it possible to ensure that skew does not occur, although the clock used for play-back may in fact operate at a different frequency to that used to generate the original time stamps. If play-back is from storage sites operating at different frequencies then this becomes apparent in the light of comparing time stamps after sinks adjust to their respective source frequencies.

Thus it is observed that clocks with approximately equal frequencies are desirable between sources throughout the presentation lifetime. This scheme also encompasses the situation where streams are later dispatched with reference to clocks which operate at dissimilar frequencies, such as when streams are retrieved from separate storage locations. Unless the frequency differences between all source clocks are minimised, the process of combining streams which were recorded at different occasions in real time is prone to the occurrence of skew which is not evident from examination of the original time stamps.

If skew is detected it may be possible to remedy the situation by altering the buffer length in steps. This may be unattractive if it would result in exceeding the bounds on length. Hence, it may be decided that the bounds are to be changed in combination with a lower or re-negotiated QOS. For stored streams the use of slightly different frequencies at the source results in skew. For streams which are related, the association between the frequency of the stream time systems must be defined. They will typically be the same, meaning that a discrepancy is purely due to different play-back clocks, eventually leading to a synchronisation loss. The judicious use of rate control may also be considered, in that it may be possible to make subtle regulations to the source rate of dispatch. The options of making elastic buffer adjustments as well as re-negotiating QOS are also available in this case.

3.5 Design Issues

In the previous sections the problems of performing synchronisation between streams were discussed and the merits of possible solutions examined. Building upon these foundations, the issues which are relevant to the design of a facility to support inter-stream synchronisation are now discussed.

3.5.1 Comparing Streams

Comparisons between streams are performed using their stream times as introduced in Section 2.6, which are made accessible externally by the associated stream agents. Stream time is represented by the value of a logical clock which is maintained internally by a stream agent and is used to determine the mapping of sample presentation to real time. The mapping may be non-linear due to synchronisation related adjustments to the elastic buffer as previously illustrated in Figure 3.9. This approach is also used for synchronisation involving a stream agent which handles an interleaved stream, made possible by the shared notion of time which, by definition, must exist between the original streams in that type of arrangement.

The use of time for inter-stream comparisons has a number of benefits over the use of other information. Stream specific information such as frame boundaries requires that the translations to a common reference can be easily executed. This would necessitate knowledge of the progress information from each possible type of stream agent interface. Stream time provides an inherently simpler abstraction and already exists within each stream agent for intra-stream synchronisation purposes. In addition it is not affected by changes in the granularity of a stream which complicate schemes where particular data samples are identified. The importance of providing other information about a stream's progress is important for event-based synchronisation and will be discussed in Chapter 4.

3.5.2 Performing Adjustments

After comparing stream times it may be found that the required skew bounds have been violated. It may even be possible to predict this situation by examining the current state of elastic buffers in combination with previously observed trends. Therefore an adjustment must be performed, in that the trailing stream must be advanced and/or the leading stream forced to regress. This decision process is explained in Section 3.5.3. A stream can be brought forward or retarded by making a phase and/or frequency adjustment to the stream time. As described previously, phase adjustments are achieved by skipping data or causing the stream time system to pause temporarily. This is illustrated in Figure 3.10, where the skewed situation in (a) is corrected by arranging for stream A to pause and stream B to skip so as to narrow the gap. In addition, for stored streams it may be possible to perform rate adjustments at the source. For a stream agent to operate in this way, it must be provided with direct access to the source for this purpose.

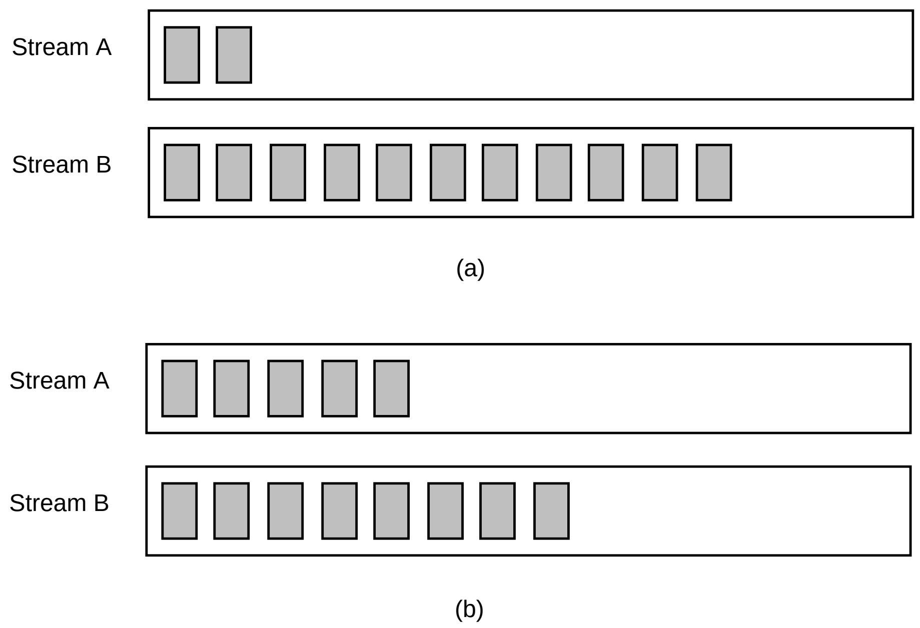


Figure 3.10: Performing Elastic Buffer Adjustments.

The ability to manipulate the buffer for phase adjustments is limited primarily by the restrictions on elastic buffer length. In addition, it may not be possible to perform skipping without a noticeable degradation in presentation quality. This decision must be made by the appropriate stream agent which is in a position to be aware of the effect of losses at the time of the adjustment. This must be determined with respect to the recent loss history and perhaps content related information such as the current degree of motion or occurrence of silence. Although application requirements for skew and latency are likely to remain relatively static in the future, advances in networks and protocols will allow reductions in transmission delay and jitter. As a result, the delay imposed by sink buffering may be allowed to increase, widening the scope for dealing with skew.

For live streams, the occurrence of skew is likely to derive from unsuitable QOS arrangements. Consequently, a longer term solution requires some form of re-negotiation or the relaxation of existing demands. Thus, for example, if skew results because of increased delays from the activities of a dynamic response mechanism (see Section 2.2.3), an application may wish to arrange for less network and source jitter, or alternatively to instruct the stream agent that greater losses or increased skew is tolerable. For a stored stream skew may also result from the use of sources which operate at different clock frequencies. In this case a longer term solution is to negotiate with the source to make a frequency modification. The stream agent must, of course, align itself with the new frequency so as to avoid a synchronisation loss due to buffer overflow or underflow. These adjustments are limited by the rate of the actual sink device and influenced by the need to ensure that presentation quality is not unduly affected.

In the process of performing these adjustments it is important to limit any jitter which is caused. This derives from the fact that samples which are presented after an adjustment, incur a different latency in comparison with those before it. The important issue here is the rate of change of jitter, since the long term variation is not likely to be of great consequence to the presentation quality. Hence the rate of performing an adjustment to the stream time must be tempered such that the rate of change of jitter is acceptable.

Therefore, one should note that the ability to perform an adjustment at a given time is best determined by the relevant stream agent. This is because it is highly dependent on the current buffer state, QOS requirements and relationship with the source.

3.5.3 Organisation

For the purposes of inter-stream synchronisation, stream agents are organised into *groups*, representing a collection of streams for which application skew requirements are specified. There are several possibilities for organising stream agents within a group so as to co-ordinate making stream adjustment decisions.

The problem can be compared to that of clock synchronisation in a distributed system, but it has a subtle difference. The latter aims to ensure that clocks show the correct time with a known degree of accuracy. For stream synchronisation, it is desired to limit skew between time systems, but without any hard notion of a correct time and allowing the existence of offsets between clock values. However, there are similarities which justify a brief examination of traditional approaches to clock synchronisation. These can be categorised into algorithms which are centrally controlled or distributed.

Centrally controlled algorithms appear in two forms: master/slave and client/server. In its simplest form, a master broadcasts its own time value to each of the slaves, which in turn make any necessary adjustment to their own clocks. Alternatively, the master may poll each of the slaves and subsequently supply a correction term to each of them. In the client/server approach, the server provides a facility which will either supply the client with the correct time or determine a correction term on its behalf. In the latter case a client supplies its own time value to the server in a request to be synchronised, while a reply is used to return an adjustment term. Both algorithms can be employed in a hierarchical configuration, with the accuracy rating of nodes, known as the *stratum*, increasing towards the root [Mills89]. Distributed algorithms operate by having each host determine the required update to its own clock after receiving the values of other clocks and taking account of estimates of their correctness. Their main benefit over the centrally controlled techniques is increased fault tolerance, gained at the expense of a greater amount of communication.

In many cases these schemes produce a correction term which requires a clock to adjust backwards as implemented in [Gusella83]. This can have undesirable consequences by making time appear to run in reverse [Lamport78], prompting a solution which spreads an adjustment over the next re-synchronisation interval [Kopetz87, Mills89]. A similar restriction applies in relation to inter-stream synchronisation, where a stream can be advanced, slowed down or paused, but backwards motion is unacceptable unless it is a feature of the presentation. Even in that case, reverse motion in the longer term is only possible for stored streams. Hence, forcing streams to converge by making reverse adjustments is entirely unsuitable.

The approach adopted here is equivalent to the centrally controlled algorithm for clock synchronisation. In contrast to a distributed approach this simplifies making comparisons between stream times and provides a focal point for dealing with situations where stream agents are unable to adjust as requested. This type of exceptional condition does not exist for clock synchronisation. For inter-stream synchronisation there is considerable scope for adapting to this occurrence, including, for example, the possibility of delaying streams to wait for one which is lagging behind. For this reason, a strict master/slave arrangement is also deemed unsuitable.

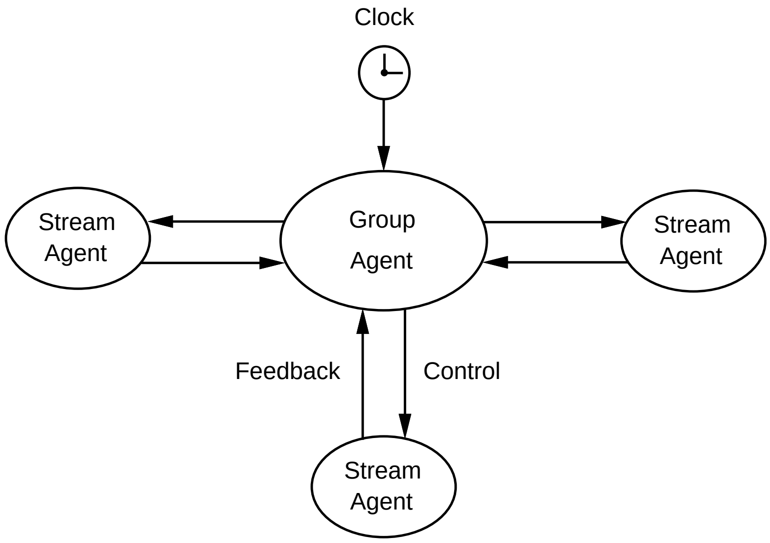


Figure 3.11: Group Agent Organisation.

In the design being proposed here for inter-stream synchronisation, the *group agent* is responsible for central group control. A group agent is illustrated in Figure 3.11 and functions by attempting to maintain its member streams within application skew bounds. These bounds are specified in terms of the maximum acceptable drift between stream time and the *presentation time* which is maintained by the group agent. The advance of presentation time can be driven either by real time at the group agent node or by coupling it to the stream time of a nominated *primary stream*. Selecting a primary stream indicates its relative importance, in that the other secondary streams should be adjusted in *preference* to maintain synchronisation. This functionality could be used, for example, to reproduce the hardware based algorithm of [Szabo91], where video output is dictated by the related audio in order to minimise audible interference. Failure of a primary stream could be handled in one of several ways:

- default to the use of real time;
- select a new primary stream based on priorities;
- consult the application;
- examine the QOS for each stream.

There are two alternatives for describing skew requirements in this organisation. An all encompassing value can be used, which specifies the maximum tolerable skew with respect to presentation time for every stream in the group. Alternatively, a method with greater flexibility allows this value to be individually provided for each stream. Further research and experience with application requirements is necessary if one is to determine whether this additional functionality is necessary. In some cases the use of a presentation time as a basis for these specifications may be insufficient. For example, in a group containing a video stream and two stereo audio streams, the skew requirements will typically be tighter between each audio stream than for those between each audio stream and the video. The use of a group skew bound necessitates the strictest requirement, hence losing the ability to handle the video in a more relaxed fashion. Even using per-stream bounds, it is not possible to dynamically balance the streams around the presentation time. This is exemplified in Figure 3.12, where a 100ms skew bound between streams, translates to fixed bounds of 50ms with presentation time. It may be useful to extend the design to accept a collection of desired skew constraints for each stream. Thus it becomes necessary to prioritise the constraints if it may be impossible to satisfy them all. It may also be advantageous to use a hierarchical arrangement for managing these requirements, comparable to that based on stratum ratings for clock synchronisation algorithms. This would require group agents to export their presentation time for use in higher level synchronisation decisions, in the same manner as stream agents make their stream times available to a group agent.

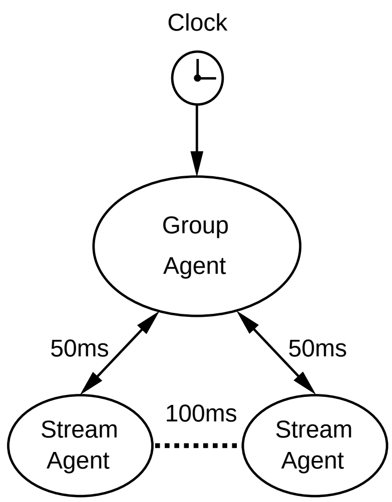


Figure 3.12: Per-Stream Skew Bounds.

3.5.4 Operation

The group agent is instantiated with the default assumption that real time is to be used to drive presentation. As stream agents are created they are attached to the group and their skew tolerance is recorded. It is also possible at this stage to indicate primary stream status. Any known or intended difference between the source clocks of streams is also supplied. This may be given in terms of a particular stream, but must be capable of interpretation relative to a common time reference.

As explained in Section 3.3.2.2, the main difficulty in performing initial synchronisation is that it is unreasonable to simply defer taking any action until all streams have started to arrive and minimum elastic buffer requirements are satisfied. The proposed algorithm commences presentation when one of the streams within the group fulfils its minimum elastic buffer requirements. This is based on the guiding assumption that later streams have probably encountered longer delays, thus facilitating the task of retarding the first stream to have started by maximising the scope for adjusting it. The time at which the first stream starts is dispatched to each of the other stream agents which can use it in determining the initial skew and making a decision on whether they should start as soon as their minimum buffer requirements are satisfied or a pre-start delay after that point.

For the purpose of maintaining synchronisation, each stream agent supplies information on its commencement to the group agent. The group agent records initial information which allows it to take account of original differences to normalise related stream times for comparison purposes. In reply the group agent provides each stream agent with details of how it should report its progress. This is done at intervals which are chosen so that skew requirements should not be compromised in the intervening period, except in unpredictable circumstances. Over time, a comparison of these progress times is used to determine suggested alterations to the behaviour of individual stream agents. An appropriate rate matching algorithm using phase adjustment has been described for computer music [Schmidt85]. In that system it is used to synchronise the playback of a stream of stored samples, in Musical Instrument Digital Interface (MIDI) format, with the operation of a tape recorder, which makes progress information available in the form of SMPTE[3] time codes. A mathematical description of a similar algorithm is described in [Anderson91a]. The important features of these algorithms are to avoid unnecessary adjustments and minimise the effects of phase alterations. In particular, as explained in Section 3.5.2, the stream agent is responsible for carrying out phase adjustments while

[3]Society for Motion Picture and Television Engineers (SMPTE) time codes provide the hour, minute, second and frame within second of each video frame and have traditionally been used to facilitate editing.

observing limits on the rate of change of jitter.

The response of the group agent to the arrival of stream agent timing information contains a target stream time which the stream agent must attempt to have reached by the next reporting point. Inability to achieve this value is reported back as part of the next interaction, at which point the group agent must take alternative action to maintain inter-stream synchronisation. This may take the form of making balanced adjustments to other stream agents or modifying the presentation rate. This decision process may be optimised if each stream agent provides an estimate of its current slack for making adjustments, based on factors such as its current buffer state and sample loss history. It is also possible to request that the application should intervene by relaxing stream or group agent operating requirements, or re-negotiating QOS arrangements. Further research is required to determine if it would be feasible to endow the group agent with the power to take these actions directly on behalf of the application, based perhaps on some concept of priority between streams, along with the importance of latency versus skew bounds for each stream.

3.5.5 System Support

For comparison purposes the group agent must be able to estimate the timeliness of information as reported by each stream agent. This is given significance by the fact that these may exist at different hosts. Specifically, it is desirable to allow stream agents to run either at the sink or on a synchronisation server host. If latencies for reporting are significant, it is necessary for the group agent to extrapolate current state, perhaps based on historical information about the stream. The era to which the stream information refers can only be determined with reference to a shared view of time. Given the high probability of stream agents existing in a physically local area, this could be satisfied by working with delay estimates or using a global clock. Attention must also be given to ensure that delays in deciding on an adjustment and arranging for stream agent activity are not such as to allow skew bounds to be exceeded. Internally the group agent is structured on a thread per stream basis and scheduling requirements could be influenced in regard to the strictness of the associated skew bound.

3.5.6 Group Manipulation

Finally, an issue in the group agent design which must be considered is how stream and group operations are handled. A group is instantiated by the synchronisation service on behalf of a client application. Operations which influence the group as a whole must be performed by the group agent on behalf of the application. These include synchronised stop and resume, as well as rate changes for stored streams, all of which are carried out in co-operation with the sources. If low latency support for command delivery and execution is unavailable then the group agent must rely on pre-scheduling and advance preparation techniques. Individual changes to a stream which occur without the knowledge of the group agent, can manifest themselves as exceptions, resulting in a loss of synchronisation with the group and a notification being dispatched to the application. For example, a requirement to increase the stream rate in combination with a revised skew bound makes it necessary to explicitly inform the group agent of the impending action. The general situation is that elements of a group will be operated on collectively rather than individually for presentation purposes.

3.6 Summary

The subject of this chapter was inter-stream synchronisation. The use of the source based technique known as interleaving was initially discussed. The approach has several drawbacks which make it unsuitable as a general mechanism, however it has some importance and can be supported by treating it as a particular stream type. Solutions were proposed to deal with the problems of achieving synchronisation in the face of skew caused by variations in stream start times and subsequent delay differences. The need for a scheme to maintain inter-stream synchronisation throughout a presentation was justified. Detailed issues of the proposed design were examined, notably in regard to the operation of group agents in their role of performing synchronisation.

Chapter 4

Event-Based Synchronisation

The use of discrete stream related events for synchronisation is the subject of this chapter. A design extension to support this function is discussed.

4.1 Introduction

Systems for multi-media synchronisation aim to provide a high degree of application control, allowing presentation to a user in a co-ordinated and well defined fashion. This is achieved by acquiring knowledge of events which occur asynchronously, but act as stimuli to influence presentation management. A common example is a user interaction, which if accepted by a window system is announced to the relevant application for handling. Similarly, notification of the passage of time or the state of I/O devices may be accomplished directly using an operating system mechanism for signalling. Events are used in combination with application supplied guidelines to manage the flow of presentation. In addition, knowledge of events may be recorded and made available for future examination.

Support for continuous media events is particularly important because it is common for streams to travel on paths which often do not involve direct application handling and manipulation of the data. Instead, a connection is set up through components between the source and sink before presentation commences, making the asynchronous nature of stream transfer apparent. The following section describes the use of different event types for synchronisation with a stream. Section 4.3 discusses the use of stream agents for event detection. Their suitability is justified by taking account of physical location and the considerable freedom to support a range of event detection mechanisms. The supporting role of the design in achieving synchronisation is the subject of Section 4.4. This is followed in Section 4.5 by an analysis of event notification and the manner in which it is influenced by real-time support.

4.2 Continuous Media Events

Useful information about a stream takes several forms as exemplified in Figure 4.1. Events which indicate the start and finish of stream presentation are used to support higher level simultaneous and sequential synchronisation respectively. Many applications require access to events which are associated with stream progress. These may be associated with stream time as introduced in Chapter 2, or perhaps expressed with respect to the boundaries of such defined data units as voice phrases or video frames. This type of functionality is necessary to support, for example, the coupling of subtitles to a video stream at presentation time, and the use of a graphical indicator to dynamically draw attention to a point of interest in a stream. It is also desirable to allow events to be identified with *markers* supplied with the stream. These may be used to convey application specific information, for example, the results of pre-processing a stream in search of video scene changes. Finally, there may be an interest in events which are derived from analysis of the stream contents at presentation time. This could be used, for example, to monitor the occurrence of audio silence or wait for a given image pattern.

Time is important when dealing with stream related events. Although an event refers to something that happens at a particular point in time, it is associated with the presentation of data which itself has a defined temporal position and duration within the stream. Hence, the time associated with the event occurrence is an integral attribute in its use for real-time synchronisation. Applications may use this information to guide synchronisation decisions, particularly in cases where it is important to know the elapsed time since the event.

4.3 Event Detection

Event detection is achieved by gaining knowledge of the state and/or contents of a stream. The physical location where this task is carried out and the actual mechanisms which are employed are now discussed.

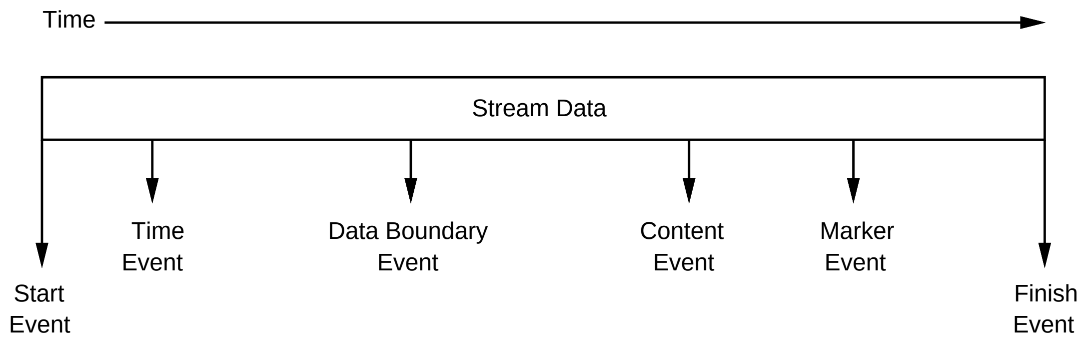


Figure 4.1: Continuous Media Events.

4.3.1 Location

The events just described are explicitly associated with data presentation. They can be tied directly to the relevant output device by using device driver feedback, thus indicating that the data has actually been presented. The alternative is to employ events which are of a predictive nature in terms of presentation. In this way, stream events can be used to refer to stream processing at a location which is between the source and sink. It is proposed here to support events which describe a stream as it is removed from the elastic buffer of the stream agent. These are predictive in the sense that they describe the state of stream data before it is actually presented. This choice of location is influenced by the requirements of the detection activity, as well as the reliability and accuracy of events in their relationship to the subsequent presentation of the data.

The alternatives to using a stream agent require an increase in the physical distance between the point of detection and that of presentation, impairing the usefulness of events for accurately predicting presentation. This is because it becomes more difficult to accurately estimate the latency between the event and actual presentation. An important component of this latency is time spent in the elastic buffer of the stream agent. In some cases it may be acceptable to rely on an estimate of this delay, ignoring dynamic adjustments to the elastic buffer. Reliability issues are also a concern, in that the data to which an event refers may be lost or discarded by the time of its predicted presentation, making it important to allow for synchronisation activities such as skipping.

In a system which does not allow the determination of accurate latencies it is impossible to use a source-based scheme to enable *precise* synchronisation. Despite this fact it should be noted that it may be desirable to make the source responsible for generating events which are not directly related to the stream presentation. Thus, for example, a file server can be expected to support events which identify the start and end of *retrieval* for a given stream, as provided in [Rangan91]. Similarly, it may be prepared to offer information on its progress within the play-back of a continuous media file. Some applications may decide to use these events to provide a crude substitute, allowing an approximate view to be formed of the presentation activities happening across the network.

4.3.2 Mechanisms

In the proposed design, the task of detecting stream related event occurrences is delegated to individual stream agents. Not only are these agents suitably located, but they allow the activity of detection to be carried out in a stream dependent manner. In the future, a standard mechanism for specifying and detecting events may emerge, but presently this flexibility is an important design feature, given the range of techniques currently being employed. In this section the general issues of event detection are discussed, although in practice the ability to detect events is tied to the type of stream agent. Thus, for example, event detection which requires knowledge of audio silence will only be available if the particular agent can either analyse the stream data or deal with explicit silence indicators.

The start of a stream is linked with the first issue of stream data from the elastic buffer. Explicit identification is necessary if multiple streams are allowed to sequentially share this buffer or there is a leading blank sequence such as may exist in a file copied from a conventional pre-recorded video cassette. Reliability issues must be considered in these cases, with appropriate time outs and default actions put in place to deal with the loss of event information.

Observing the end of a stream is more complex. One approach is to simply infer that a stream has terminated when a time out period elapses during which no stream data arrived. However, in the case where a variable sample rate is used this algorithm is unsuitable. Even for fixed rate streams the approach is prone to produce errors, due to intentional pauses, unexpected delays and temporary failures which accidentally cause such a time out period to be exceeded. Selecting a large time out period reduces the accuracy of the event detection mechanism in this case, because of excessive latency in reporting the event.

An alternative scheme takes closure of the associated communications channel to indicate that the stream has finished. In this case the latency between the presentation of the last stream data and the channel closure is problematic and could be substantial, especially in the case of stored streams where data flow will terminate when the end of the file has been reached. The use of multiple streams which are transmitted sequentially over a single channel is not possible using this scheme as it can only detect a single finish event. For stored streams an algorithm could be devised which operates using a decreasing or predefined range of sample or packet numbers. However, a general solution requires that the source supply an explicit *tag* to identify the stream end. This is generated dynamically, also making the approach suitable for dealing with live streams, where a maximum length value is not available in advance. The possibility that a tag may be lost in transit is a serious problem in this case and must be incorporated into the algorithm, while premature termination of the channel must also be handled appropriately.

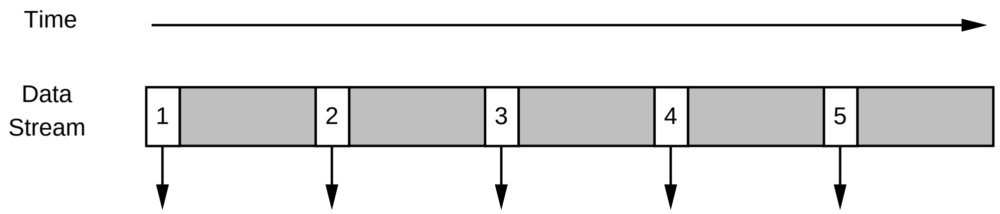


Figure 4.2: Embedded Stream Events.

Detecting progress events within a stream requires access to information which describes the stream representation. In the case of time and data boundary events this information is typically embedded within the stream in an in-band fashion as illustrated in Figure 4.2. Markers can also be embedded in a stream as shown in Figure 4.3, or alternatively, a distinct interleaved marker stream may be used as in Figure 4.4. This has the benefits of not having to modify the original stream and allowing easy modification of the set of markers if required. Marker events can also be derived by indexing them in a pre-supplied table using the information contained in timing events. Finally, content related events require comprehension of the actual encoding of the data or alternatively must rely on explicit marker events.

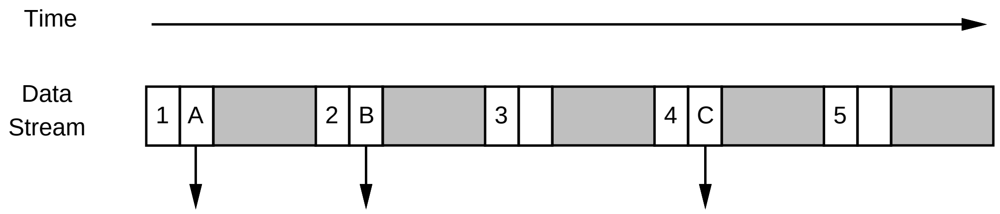


Figure 4.3: Embedded Stream Markers.

An example is the interleaved underlay stream of the DVI system [Luther89] which contains data to be delivered to the application in synchronisation with stream presentation. This is in addition to DVI support for embedded time stamps [Ripley89] and an application supplied *hook routine* which, if installed, is invoked by the support system for each frame of video which is processed.

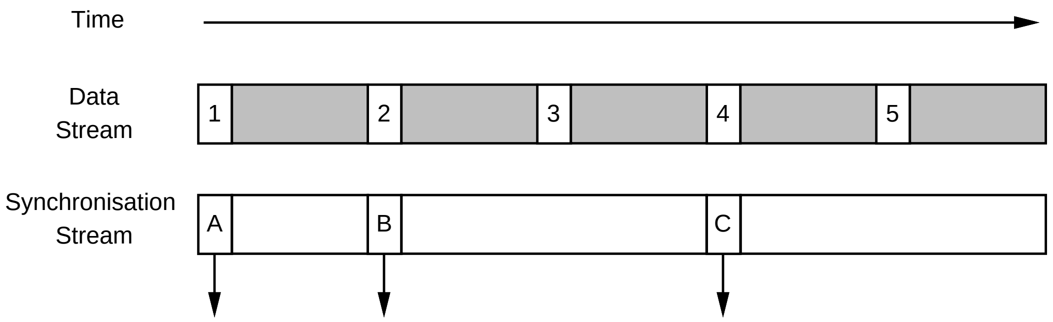


Figure 4.4: Markers in a Synchronisation Stream.

The use of such a variety of techniques for conveying stream events strengthens the argument for using typed stream agents for event detection. This conclusion is reinforced when the use of stored streams is considered. Source-based detection would require a filing system to possess extensive knowledge of each possible stream format and deal with the additional overhead of event detection activities. Many events can only be determined by examining the contents of the actual data stream, requiring that details of the encoding and compression techniques are also accessible. In the particular case of filing systems it appears reasonable that file servers are unlikely to be in a position to fully comprehend stream formats and their content. Depending on the format which is used it may be impossible to detect data boundaries while the stream is in a compressed state. For example, the use of inter-frame video compression reduces the granularity at which individual frames can be identified. Operation at the stream agent proves more attractive in this case. Thus, the choice of the typed stream agent of Chapter 2 is enhanced as the optimum location at which to execute event detection.

4.4 Performing Synchronisation

For inter-stream synchronisation, a stream agent is obliged to make its current state available to the appropriate group agent where it is compared with that of related group members. The nature of this information is the same as that given by the events being considered here and it is even possible to use a single mechanism for notification purposes. It should be noted, however, that the use of the events described in this chapter is quite different. For inter-stream synchronisation, there is a requirement to dispatch stream time events in a frequent and timely fashion to a single client, the group agent. General event-based information is expected to be required with less regularity, in some cases to multiple clients and with more relaxed time constraints. In addition, the function of inter-stream synchronisation is well defined, whereas in general, events may be used in a wide range of ways to achieve other forms of synchronisation.

The proposed design is guided by the need for great flexibility in the use of stream events. Rather than attempting to perform event-based synchronisation within the agents of the synchronisation service, support is provided for supplying event notifications to any interested parties, allowing synchronisation activities to take place externally. An alternative is to attempt to program synchronisation constraints into the service, possibly using some form of script language or combination of event/action pairings. In this way event occurrences are monitored and used to drive a synchronisation machine having a predefined pattern of behaviour. However, the complexity of this scheme is considerable and further research is necessary to determine its feasibility. The proposed design does not preclude the construction of such a facility outside the synchronisation service, with operation based upon event notifications as they are supplied by stream agents. Indeed, a simple scheme of this form is discussed will be described in Section 4.5.2. As it stands, the design imposes no constraints on the manner in which event notifications are employed once they are dispatched.

Another possible approach is to control synchronisation by expressing the requirements within the stream itself. Chapter 6 will describe two related research projects in which this is adopted. In the first project (described in Section 6.1.2.4), information in a separate synchronisation stream identifies specific data items and indicates the manner in which they should be presented. The other project, described in section 6.1.2.5, allows control over the processing order of presentation requests, both within and between streams. This may be compared to the scheme in [Gifford88], where the order in which asynchronous RPCs are dispatched over a group of channels may be reproduced at the sink. These schemes are designed for situations where related streams have the same destination, hence simplifying the task of synchronisation.

The use of an event-based scheme provides a relatively simple solution and shifts the emphasis for ensuring non-continuous synchronisation to the application. Related research for application support in event-based systems will be outlined in Section 6.1.1.

4.5 Event Notification

The role of a synchronisation server is to arrange that its agents detect the occurrence of events and issue corresponding notifications, so that higher level synchronisation constraints can be satisfied. This section discusses the organisation of an event notification system and the technique employed for conveying information to interested clients.

4.5.1 Organisation

There are two possible approaches for organising the system for event notification. Event notifications can be dispatched directly to interested clients either by the detecting agent or via an *event service*. In either case some mechanism for filtering or selecting interesting events is necessary in order to avoid potentially swamping a client with a deluge of irrelevant information.

The arrangement which involves an event service is illustrated in Figure 4.5. Group and stream agents inform the service of all events and it dispatches them to clients which have registered an interest. An event service effectively operates as an intelligent filter and redistribution point for event notifications, thereby avoiding the need to increase agent functionality to deal with multiple clients and their specific requirements. It can also reduce agent load by allowing them to only make a single notification for each event occurrence. As an extension, it is possible to consider augmenting event service operation to deal with a general range of other related events.

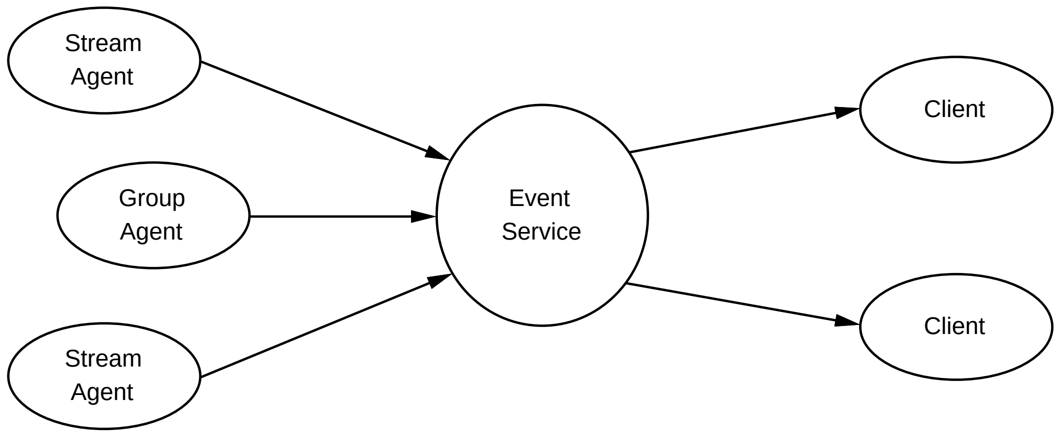


Figure 4.5: Event Service Approach.

However, a scheme involving an event service in not adopted in the proposed design for a number of reasons. Firstly, events related to a given stream are commonly only of interest to a single client. Secondly, the task of event reporting in most instances has strict time constraints, the satisfaction of which is complicated by the use of an indirect notification path. Thirdly, the filtering benefit of an event service can be effectively reproduced directly within an agent. An agent is not typically required to notify every event occurrence, and discrimination can be carried out using relatively simple guidelines based on event types and desired frequency.

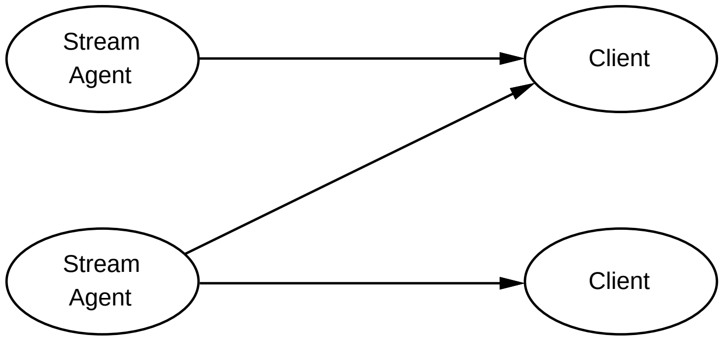


Figure 4.6: Direct Notification.

The adopted approach to event notification imposes a direct relationship between each agent within a synchronisation service and interested clients as illustrated in Figure 4.6. This requires applications to explicitly register with the particular agent in order to request that events associated with the stream are subsequently notified. This registration interface allows high level filtering based on the selection of available event types. Consequently, an agent generates a notification for an event only if a registered client exists. Additional parameters allow further discrimination based upon the frequency and/or granularity of interest in an event sequence. Thus, for example, an application may express an interest in video frame events with a granularity parameter specifying every tenth presented frame. The interface must also support dynamic alteration and removal of client interests in events.

Events are typed based on the selection described in Section 4.2. In some cases an event is singular such as at the start or finish of a stream. The common case is that a sequence of the same type of event occurs during a presentation, making it necessary to augment the event information with an instance identification. An example is an event tied to the presentation of a video frame, which to be useful must allow the position of the frame within the stream to be determined. Thus each event has a type and an instance. Event types are further categorised into basic and supplementary classes. Basic events are those which are always supported. These are the start and finish types and events derived from stream time advances. Other types are supplementary and are supported at the discretion of the particular stream agent.

Event notification is a two stage process. Within each agent a local synchronisation mechanism is used to signal the occurrence of events to those components which are responsible for interacting with clients. The subsequent task of making an external notification requires the use of an RPC or message passing protocol. Research in [Nicolaou90b] suggests that an RPC based call-back mechanism is most appropriate in this respect. Hence, a client provides the agent with the identification of a remote procedure which is to be invoked when events of a particular type occur. Multi-cast support may be beneficial when several clients express an interest in the same events.

4.5.2 Real-Time Support

In most cases there are client requirements which demand that events get detected, reported and acted upon within specified real-time constraints. These exist to support higher level synchronisation and are typically less critical than those associated with inter-stream synchronisation. This is a very complex problem and requires further research in order to determine its general feasibility. A solution must effectively arrange for suitable scheduling arrangements at the agent and client, in addition to time-constrained communication support for notification. The former is likely to be complicated by the sporadic nature of event occurrences, while the latter requires considerable advances to be made for real-time message passing and RPC systems.

In terms of the synchronisation service, the scheduling of threads which are associated with event notification must be strictly controlled, as otherwise it may become extremely difficult to satisfy other synchronisation needs. The application must balance its requirements in these regards, taking into account its capabilities and those of the involved agents. For example, it may be prudent to reduce the frequency of event notifications in order to allow the intra-stream synchronisation QOS to be maintained at a satisfactory level. In particular, the scheduling of event notification threads within an agent may result in excessive jitter as viewed by the output thread. Reactions of this form could potentially be supplied as instructions to a programmable stream agent. Alternatively, congestion at the application host may lead to the adoption of the short term approach of discarding those events which cannot be processed. It should be noted that the form of QOS guarantees required for event notifications may not be the same as those associated with the handling of streams. This is because in many cases the former will demand high reliability as well as delay bound guarantees. An example is where a video frame event may be employed by an application to perform a graphics overlay, demanding a low and bounded latency for the notification. In addition, the notification should be delivered without loss or corruption unless the application has implemented a suitable recovery mechanism.

In order to minimise problems due to undesirable notification latencies a possible approach is for applications to employ the use of *proxy handlers* which can be located very close to the point at which events occur. Such a handler would process notifications on behalf of an application using advance information, which has the effect of relieving the application burden. This could take the form of the application supplied, time critical call-backs in the CD-I system [Meissner91], which can be installed at runtime for execution on an embedded operating system with real-time characteristics. In essence, the use of a proxy handler arrangement is advisable where synchronisation requirements can be expressed in a relatively straightforward fashion and timeliness constraints are tightly bounded. An example of this scenario is the co-ordination of a set of graphics overlays with frames of a related video stream as illustrated in Figure 4.7. The accuracy of a handler's response to an event notification can be increased by having it prepare in advance, in this case by pre-fetching graphics data to avoid additional unexpected delays. Preemptive activities of this nature must be performed in a predictive manner, based on the analysis of related event notifications prior to the target. For example, graphics data to match video frame 100 could be fetched when the notification for event 90 is received. Determining the requirements for guiding advance preparation is a complex issue requiring further research.

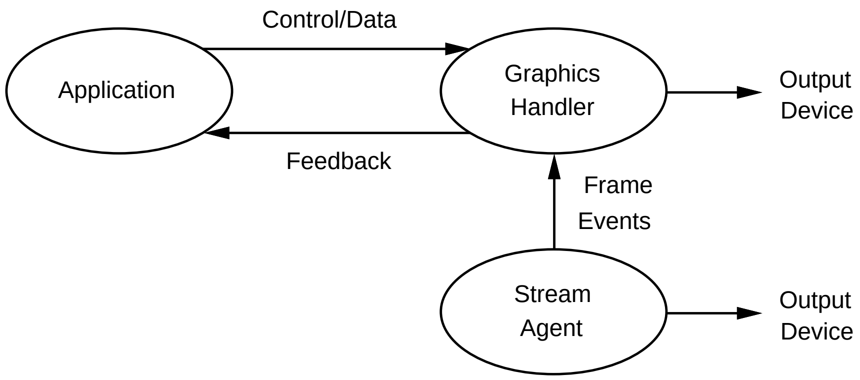


Figure 4.7: Use of a Proxy Handler.

The concept of programming synchronisation requirements in this way may be compared with the proposal in [Staehli91], where filing system retrieval events are used in combination with an application supplied script to drive the retrieval order of continuous media files. This general approach is only suitable when synchronisation requirements can be expressed in a relatively simple manner. It offers the additional benefit of allowing future client requirements to be analysed and perhaps employed in combination with some form of resource reservation.

Another requirement is to allow the receiver of an event notification to determine the time at which the associated event actually occurred. This information is essential because the temporal attribute of an event is likely to be employed as an integral part of the higher level synchronisation process. The use of a time constrained communications facility is one solution to this problem or alternatively a common time system may be considered. Unlike the situation for inter-agent communication it is more probable that this communication will occur over a relatively wide area, thus making the provision of a global clock or accurate propagation delay knowledge more complex. The use of proxy handlers can be used again to ameliorate the situation.

4.6 Summary

Synchronisation using discrete stream related events was discussed in this chapter. The general use of information about asynchronous event occurrences for presentation management was outlined. The form of an event and its manifestations were described, along with proposals regarding the manner and location of event detection activities. It was argued that events are best detected within stream agents and made available for external use in performing synchronisation. In the proposed design, event notification is achieved by having interested clients register their interests with appropriate agents of the synchronisation service. The use of proxy handlers which can perform simple synchronisation functions on behalf of an application was advocated. These handlers are particularly beneficial for event-based synchronisation which must operate under strict time constraints.

Chapter 5

Experimental Evaluation

This chapter describes an evaluation of the proposed synchronisation facility, based upon experiments with an implemented prototype. Chapter 6 completes the evaluation process by allowing a comparison with closely related research.

5.1 Introduction

This chapter describes the use of experimental techniques to evaluate the proposed design. The evaluation was performed using a prototype synchronisation server, operating in an ATM inter-network environment. A set of experiments were carried out, which confirm the feasibility of the approach and illustrate the type of problems which are encountered when performing synchronisation in a user space server. Conclusions are drawn regarding the supporting role of the operating system kernel. Prior to describing the prototype in Section 5.3, an outline of the local systems environment is given in Section 5.2. The programme of experiments is described in Section 5.4.

5.2 Local Systems Environment

This section presents relevant background information about the environment used for development and execution of the prototype. A summary of the elements and organisation of the system is first given, followed by a technical abstract in Section 5.2.2, describing the supporting micro-kernel.

5.2.1 Synopsis

The environment used for development and experimentation consists of a collection of heterogeneous machines, which are physically distributed over two separate sites in a metropolitan area. Workstations running the UNIX operating system have Ethernet access and provide support for development, as well as a platform for executing applications. Several of these can support multi-media applications by using a directly attached Pandora Box [Hopper90] as described in Chapter 1. This allows audio and video data to be transferred over a CFR and can be controlled by communicating directly with a process running on the workstation.

Machines used experimentally run the WANDA micro-kernel [Dixon91], which is described in the next section. Most of these machines have access to a CFR, while a small subset are additionally equipped with interfaces for Ethernet and/or the Cambridge Backbone Network (CBN) [Greaves90]. The CBN is a fibre optic based, Metropolitan Area Network (MAN), which currently operates at 500 Mb/s. Originally designed to serve as a backbone for CFR interconnection, it has a slotted ring access mechanism and ATM properties. The use of this network for continuous media traffic undergoes experimental analysis in [Dixon91].

The main communication protocol is the Multi-Service Network Level (MSNL), defined within the Multi-Service Network Architecture (MSNA) [McAuley89]. MSNL is implemented within WANDA, Pandora and most locally available versions of UNIX, supporting unreliable communication based upon the use of lightweight virtual circuits in an inter-network environment. At a higher level, an implementation of the Advanced Networked Systems Architecture (ANSA) [ANSA89], known as the *Testbench*, provides RPC capabilities.

5.2.2 WANDA

WANDA is a micro-kernel developed locally to support systems research. Its main intended use is to provide a lightweight platform for *running* user programs. These are developed in a workstation environment and for execution may be *spliced* on to the kernel prior to booting, or dynamically loaded by using ANSA RPC to interact with a user space server.

A WANDA process, known as a *domain*, is an address space which is shared amongst a collection of independently scheduled threads. Depending on the kernel configuration and hardware support, domains may be backed by a virtual memory subsystem, or alternatively, operation may be in *single address space* mode. The latter offers performance improvements by eliminating superfluous data copying and context switch overheads. This is mainly designed for situations in which a machine runs as a dedicated server, reducing the need for the protection offered by virtual memory support.

Scheduling decisions are made by the kernel based upon the (statically assigned) priorities of runnable threads. Preemption occurs if a thread with higher priority than the currently running thread becomes available. Thread switching is optimised by arranging for a thread which blocks to *spin* within the scheduler based on its wake up condition, thus avoiding a change of context in the case where it is resumed prior to other activity. The kernel itself is preemptable and contains several background threads, such as those for monitoring device operation. Basic synchronisation between threads in a domain is achieved using traditional semaphore operations. A semaphore wait may be qualified, based on a time out value specified in terms of either relative or absolute time.

The kernel supports network and device access using I/O buffers as mentioned in Section 2.5.1. An I/O buffer is a contiguous region of physical memory which, if allocated, is *mapped* with read and write access into the address space of the owning domain. In regard to Interprocess Communication (IPC), such a buffer is effectively detached from the domain once the data it contains is sent. Conversely, upon receipt, the kernel places newly arrived data in an I/O buffer and attaches it to the target domain. This scheme has the benefit of avoiding data copying between user and kernel spaces for these purposes.

The IPC interface is derived from the Berkeley UNIX socket abstraction, with synchronous operations for handling connections and performing data transport. Support is provided for MSNL as a protocol domain and is used by the ANSA Testbench which runs in user space. An interesting feature of the implementation supports efficient gateway operation by allowing two MSNL connections to be *joined* from user space. Subsequent cell forwarding takes place in the network interrupt handlers.

The author modified WANDA in order to increase the accuracy and granularity of kernel timing facilities. In particular, the standard version of the kernel allows all interrupts to be disabled at selected points, to promote efficiency and simplicity of operation. This has the side effect of causing the software maintained clock to drift, because of clock interrupts which go unnoticed while in this state. This problem is particularly acute during periods of high levels of network activity, such as when dealing with a video stream, potentially leading to a loss of intra-stream synchronisation as discussed in Section 2.4.

The implemented solution is to ensure that the clock interrupt is never disabled or masked, except when executing the clock interrupt handler itself. A subtle problem arises, because in addition to maintaining the time, this handler examines the queue of threads which are blocked expecting a timed wake up. A thread is woken by placing it on the runnable queue, perhaps causing a reschedule operation on exiting the interrupt handler. In order to maintain the integrity of kernel data structures and prevent the possibility of rescheduling during a lower priority interrupt, a flag is used to indicate periods when the timer queue should not be examined in this manner. Hence, the handler proceeds with updating the time, but defers other processing until dealing with a subsequent clock interrupt during which normal conditions prevail.

A further kernel modification was made to support high granularity timing, necessary when dealing with continuous media. The standard version requires the clock interrupt handler to maintain system time in seconds and microseconds. This has the overhead of having to check for microsecond overflow as well as the need to use two values when examining the timer queue. Kernel timing was changed to operate in terms of clock *ticks* rather than real time units. This yields a reduction in the minimum clock interrupt processing latency and simplifies timer queue processing. Consequently, this makes it possible to offset the additional processing overheads of operating at the relatively high clock period of 1ms. A drawback of the scheme is the increased latency involved in reading the time from user space, because of having to convert from ticks to system time. Additionally, it shortens the period of operation before wrap around occurs for system maintained time.

5.3 Prototype Implementation

The prototype synchronisation server is implemented as a user space program which executes with the support of WANDA. Using the terminology introduced in Chapter 2, it operates using the *remote* approach, where synchronisation is provided as a shared service for a local area ATM environment, in this case based on the CFR. The server incorporates all synchronisation activity and executes stream and group agents within the confines of a single domain. In comparison with a multiple domain approach, this simplified implementation and increases the efficiency of agent interactions. The prototype executes on a dedicated machine running in single address space mode, ensuring that server operation is faithfully observed. Interaction with the server is performed using ANSA RPC from a workstation based control program.

Two basic types of stream agent currently exist, supporting synchronisation for Pandora audio and video. Pandora transmits stream data over MSNL in a sequence of messages, each known as a *segment*. Within a stream, segments are of equal size and are transmitted at regular intervals. The format of a segment is illustrated in

Figure 5.1. Of particular relevance for synchronisation are the sequence number and time stamp fields of the basic segment header. Sequence numbers allow losses to be detected, while time stamps indicate the temporal position of segment data within a stream. Audio time stamps are generated using a clock on the audio card within a Pandora Box, while a physically separate time source on the video card provides stamps for video segments. Audio segments contain 4ms worth of data captured at 8000 Hz and with 8 bits per sample, requiring a fixed length packet size of 96 bytes after rounding up for transmission in 32 byte CFR cells. The size of a video segment is dependent on selected stream parameters which define the image dimensions and optional use of DPCM compression. Pandora may package larger images into more than one segment.

Code for each stream agent is present statically within the server as a library. An instance is created and destroyed in response to RPC requests from the controlling application. In particular, during creation the threads which comprise a stream agent are instantiated. A group agent exists only while there are members within its group. Hence it is created when a newly created stream agent indicates a desire to join a group which does not exist at that point. Each stream agent maintains state information about its operation and parameters which influence its performance. These parameters may be supplied when creating the agent or based on default values. The prototype interface requires values to be supplied for:

- stream agent type;
- stream destination;
- minimum elastic buffer length;
- maximum elastic buffer length;
- input thread priority;
- output thread priority;
- tolerable output jitter.

In addition, for a stream in a group relationship, a group identifier must be supplied, along with information giving the offset (if any) of the time stamp clock. The latter is used to support synchronisation between streams where sources have different clocks, by comparing offsets taken at the same point in real time. In regard to group management an indication that the stream is to be treated as a primary stream may also be given at this point. Alternatively, an upper bound on tolerable skew with respect to the presentation time must be supplied.

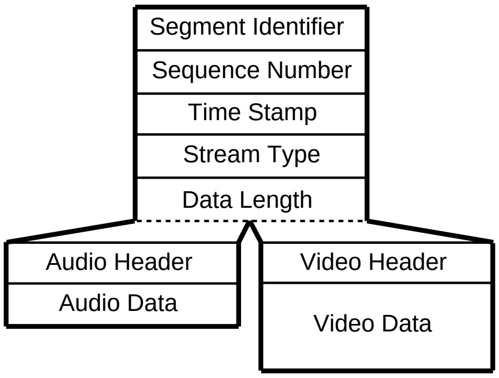


Figure 5.1: Pandora Segment Format.

The implemented stream agents operate a static elastic buffer mechanism (see Section 2.2.2). Therefore, the target buffer length (initialised from the minimum length parameter) is not changed except to accommodate the inter-stream synchronisation algorithm. Consequently, the parameter which specifies the maximum buffer length is only employed in that situation. Naturally, the actual memory usage of the buffer is dependent on the jitter of the segments which it contains. The target length can increase or decrease by a small amount due to a frequency offset between clocks, but this quantity is bounded for each stream by the operation of a clock drift thread. This makes adjustments to the target length in order to maintain phase synchronisation with the source as discussed in Section 2.4.

A stream agent is structured according to the design presented in Section 2.6. The elastic buffer is implemented as a linked list of elements, each of which contains a reference to the WANDA I/O buffer allocated for the associated segment. Segment header contents can be read by an agent without having to copy the contents of the related I/O buffer. Similarly, the segment can later be transmitted without having to transfer its contents via user space, hence maximising agent throughout. This arrangement is illustrated in Figure 5.2 and is comparable to the use of a buffer module in the proposal for a distributed stream agent in Section 2.8.1. Note that operations to add and remove segments from the elastic buffer do not involve copying segment data because of this feature.

The input thread receives incoming stream data, records any losses which are detected and inserts segments into the elastic buffer. It has a producer/consumer relationship with the output thread, which is responsible for removing segments from the elastic buffer for dispatch to the destination. After data has started to arrive, the output thread is woken and arranges with the kernel to postpone further operation until the minimum buffer complement is reached. It then enters its

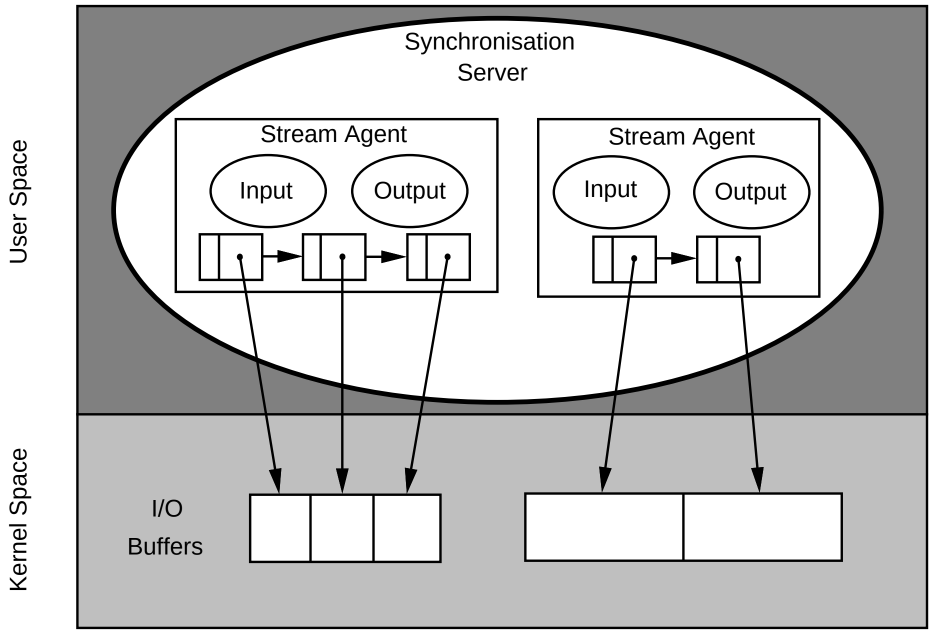


Figure 5.2: Elastic Buffer Implementation.

main operational loop unless involved in a group, in which case it firstly executes the algorithm for initial inter-stream synchronisation described below. Within the loop, it removes a segment from the elastic buffer and calculates the target play out time. This value is then compared with the current and stream start times in order to determine if sending the segment would exceed tolerable jitter bounds. If so, the segment is discarded and execution continues with the next buffer element; otherwise, a timed wake up at the target transmission time is arranged. The ability to sleep until a specified real time, facilitates dealing with streams, by factoring out unexpected variations in the output thread handling latencies for each segment.

The algorithm for initial inter-stream synchronisation operates as described in Section 3.5.4. The first stream in a group to reach its minimum complement of elastic buffer elements commences operation and its actual start time and initial time stamp are recorded. Once this has happened, other members of the group make a synchronisation decision upon reaching their minimum buffer length, resulting in a timed delay or entry to its main loop. For each stream, a note is made of the value of any initial skew which is present after this algorithm has been executed. In any case, a stream does not intentionally violate its buffer length bounds.

The group agent attempts to bound any skew which exists after the presentation has started and/or which occurs during subsequent operation. It performs this duty by making regular checks on the stream times of group members. A decision to adjust these values, by forcing an agent to skip or pause, is based upon the measured skew values and knowledge of elastic buffer length bounds. These adjustments are imposed by modifying the recorded stream start times, thereby causing the related output thread to react accordingly, by discarding samples which it deems to be late, or requesting an extended timed delay before sending the next segment. This activity is constrained by the allowed frequency and granularity of adjustments, which are selected so that changes in segment jitter occur at a acceptable rate.

The prototype supports basic event based synchronisation. This is implemented within the video stream agent and is based upon the dispatch of video frames. Clients may use ANSA RPC to register an interest in monitoring these events at a requested level of granularity. Event occurrences are dealt with internally by using semaphores to synchronise between an output thread and those which handle event notification for each registered client. Notification is subsequently performed by using ANSA to inform the interested client of the frame which has been transmitted to the destination.

5.4 Experimental Programme

The programme of experiments was carried out in the ATM inter-network environment of Figure 5.3. Each stream originates at a source on the service CFR and travels over a fixed route to the synchronisation server, which is attached to a private CFR. This route takes a stream over the CBN and through two identical gateways, which manifest themselves as points of contention, hence producing jitter. These run on a pair of 68040 based WANDA machines and operate in cell forwarding mode as described in Section 5.2.2. Cell queueing within a gateway is performed using a FIFO mechanism, for which experimental results are presented in [Dixon91].

The source of a stream is either a Pandora Box or the Continuous Media File Server (CMFS) [Jardetzky92]. The CMFS allows Pandora streams to be stored and retrieved in real time and runs on a 68030 based WANDA machine, equipped with a set of conventional hard disks. To support related streams, it will honour requests to synchronise the start of record and play back operations, using the techniques discussed in Section 3.3.1.

The synchronisation server also runs on a 68030 based machine with 8 MBytes of RAM and network interfaces for the Ethernet and CFR. After synchronisation, a stream is transferred over the local ring for presentation on a Pandora Box, or to a measurement server for unintrusive experimental observation. As a result of the properties of the CFR (previously described in Chapter 1), the jitter which occurs

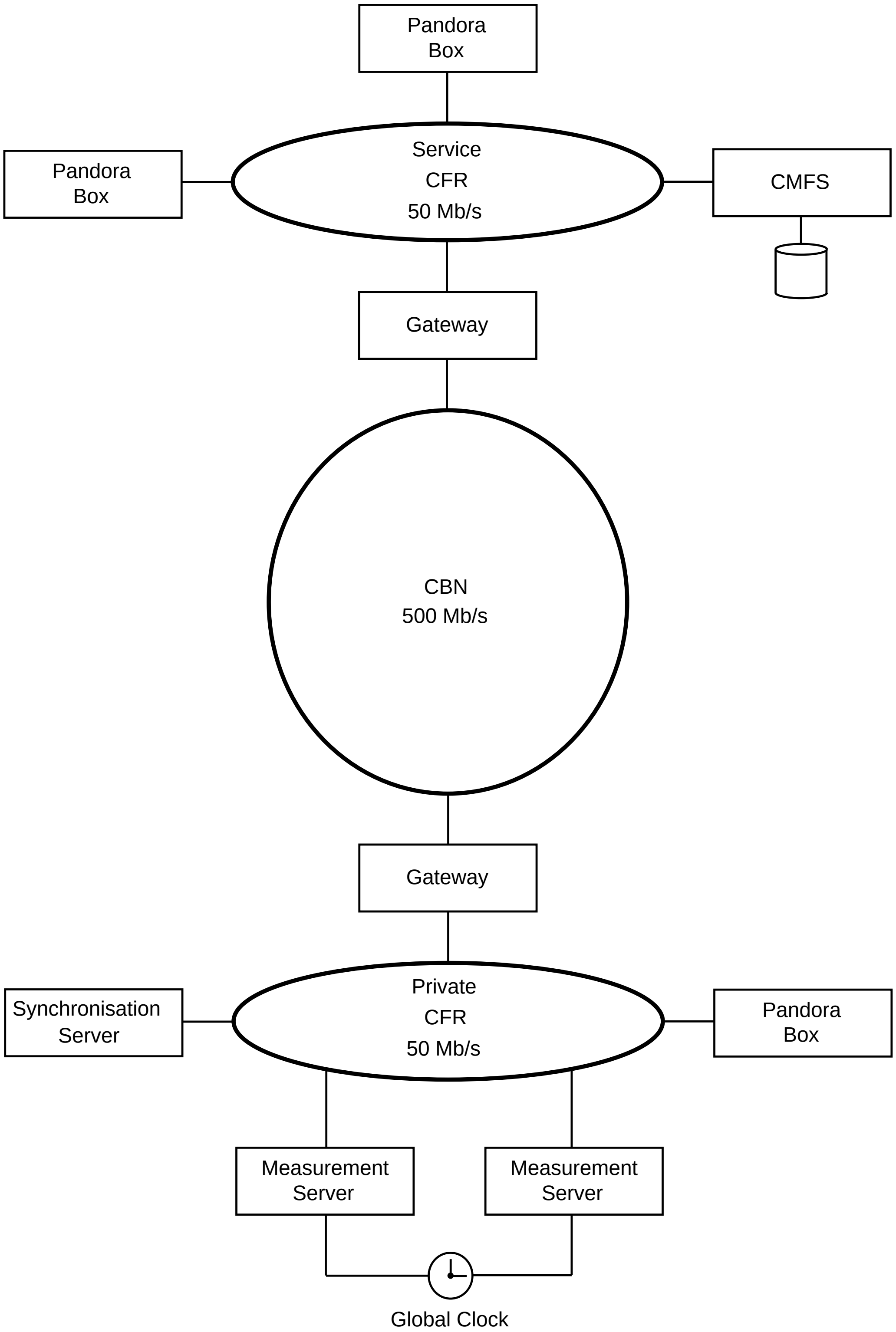


Figure 5.3: Experimental Configuration.

between the synchronisation and measurement servers is both low and bounded throughout the programme of experiments.

The measurement server is a WANDA application which is executed on a pair of 68020 based machines. It operates by recording segment information, including local arrival times, which is subsequently uploaded to a workstation for analysis. The synchronisation server copies a small amount of state information into the data portion of each segment for this purpose, allowing its operation to be closely monitored. Each of the measurement machines has a memory mapped counter card[1], driven by a shared 4 MHz oscillator, providing a global clock with microsecond precision.

This clock allows synchronisation between streams to be easily compared, and increases the accuracy of arrival time measurements, by the ability to disable on-board clock interrupts throughout each experiment. Consequently, the receiver is the only runnable thread, ensuring a small and bounded delay between segment arrival and execution being resumed. This value is not fixed, due to the overhead of interrupt handling for a management cell sent out by the monitor station once a second. Occasionally this may occur between segment arrival and reading the global clock from user space, in which case the measurement accumulates a value of approximately 75μs.

Stream Type	Size (bytes)	Period (ms)	Bandwidth (Kb/s)
Audio	96	4	192
Video	1888	40	377.6

Table 5.1: Pandora Stream Details.

Numerical details of the Pandora audio and video streams used in the experiments are given in Table 5.1. Unless otherwise stated it is to be assumed that video streams were used at their full rate, i.e. 25 frames per second. Each experiment was run for the duration of 50000 audio samples and repeated in order to establish the results. In the histograms which follow, the interval over which samples are grouped is 250μs, with the exception of Section 5.4.2, where this value is decreased to 25μs so that relatively small variations in inter-arrival times can be highlighted. The results express segment inter-arrival times in terms of the maximum and centile values, where the nth centile is defined as the minimum interval in which n% of the sample distribution occurs. All timings are given in microseconds unless indicated otherwise.

[1] Designed and built by M. J. Dixon at the University of Cambridge Computer Laboratory.

5.4.1 Initial Measurements

The aim of the initial two experiments is to illustrate the behaviour of the target environment under defined loading conditions. The results describe the nature of the problem being tackled and allow selection of a suitable elastic buffer length for use in subsequent experiments.

Experiment 1 involved two audio and two full rate video streams which were generated by a Pandora Box on the Service CFR. The inter-arrival times of segments were recorded for both an audio and video stream as they reached their respective measurement servers. The remaining two streams provided contention, traversing the same route to a Pandora Box on the private CFR. The distributions are shown in Figure 5.4.

Experiment 2 dealt with streams which were transmitted to the measurement servers by the CMFS, from its location on the service CFR. These has been recorded prior to the experiment. The distributions are given in Figure 5.5 for an audio and full rate video stream. During the experiment the filing system was simultaneously recording an audio and video stream (at 5 frames per second) from a Pandora Box on its local CFR.

The configurations used in these experiments were employed consistently in the main three phases of experiments which appear in the following sections. Hence, the total number of streams and their inter-network routes remain the same, except that henceforth an audio and/or video stream is re-directed to pass through the synchronisation server.

5.4.2 Single Stream Operation

The set of experiments which are discussed in this section examined the intra-stream performance of the synchronisation server when handling a single stream of audio or video. The aim is to characterise the influence of selected scheduling priorities for the input and output threads. Throughout the experiments, the WANDA priority structure was viewed in terms of an 8 level priority scheme as illustrated in Figure 5.6. In this arrangement, a thread at level 0 has the minimum priority and can be preempted by those assigned a greater priority, i.e up to the maximum value of 7. Within this organisation, there exist a number of *background* threads which execute during the experiments at levels 2 and 3. These include a kernel thread which examines the observed state of the CFR every 15 seconds, as well as those belonging to ANSA which perform various housekeeping duties. Note that throughout all of the experiments, the clock drift thread for each stream was assigned the

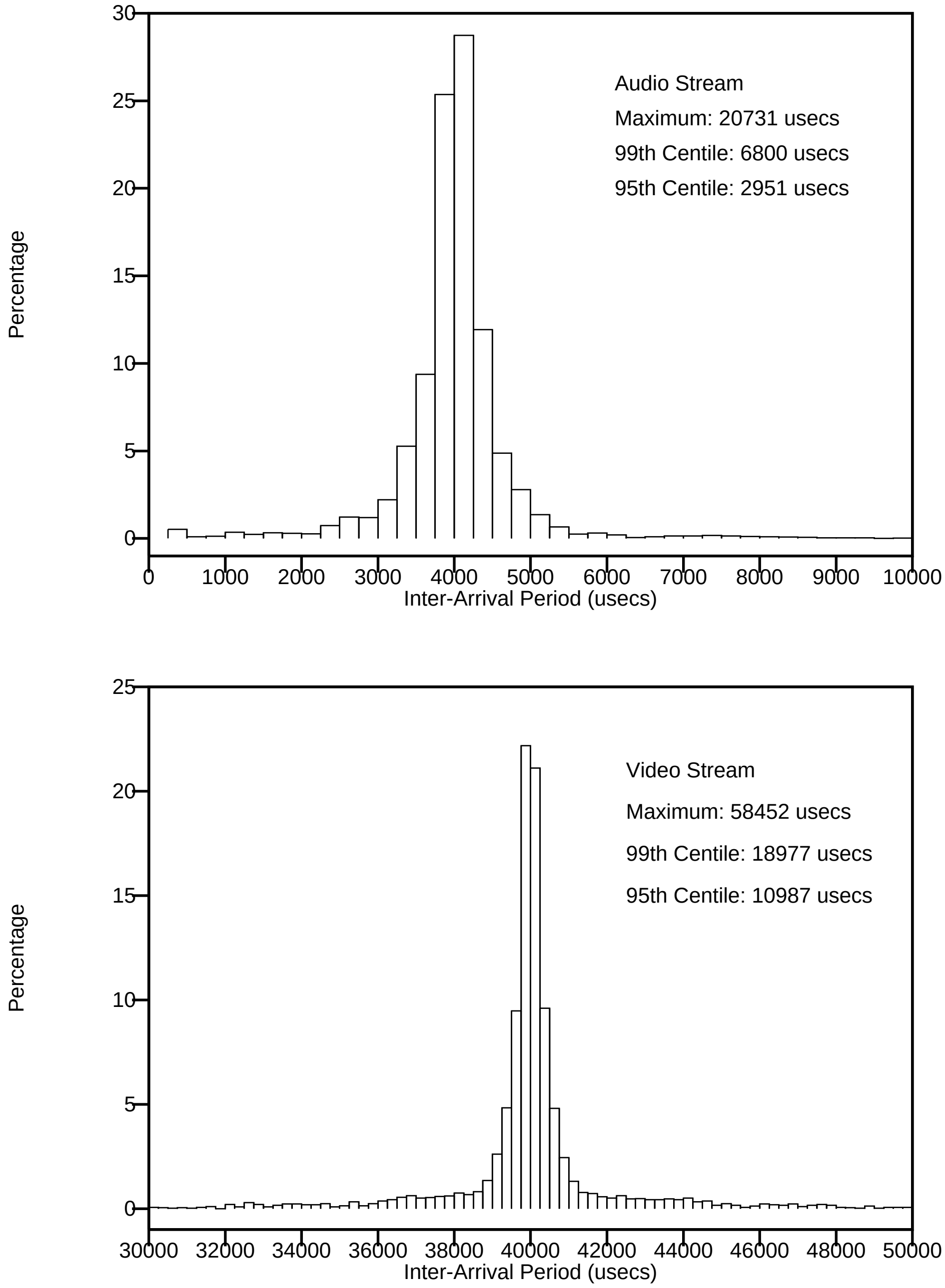
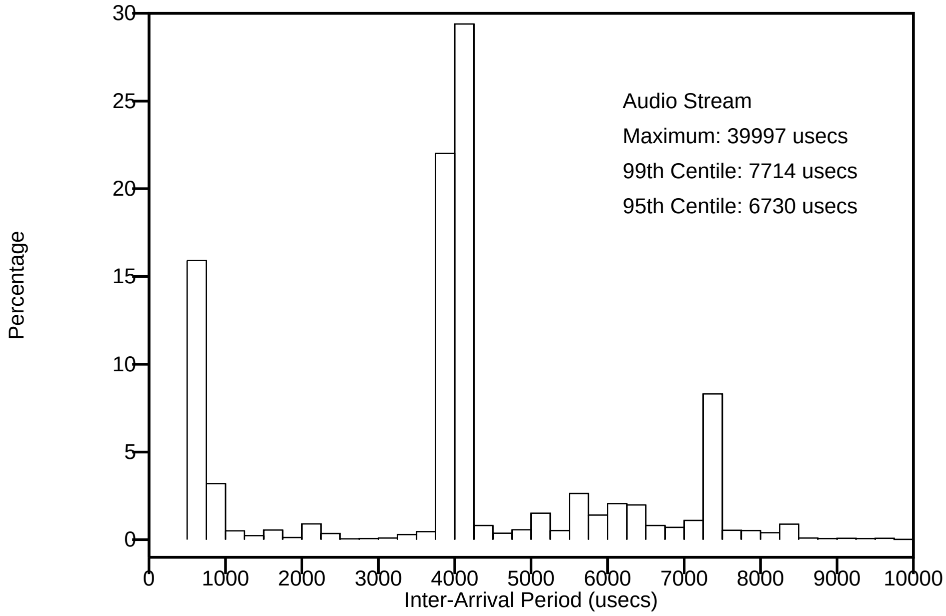


Figure 5.4: Distributions for Experiment 1.



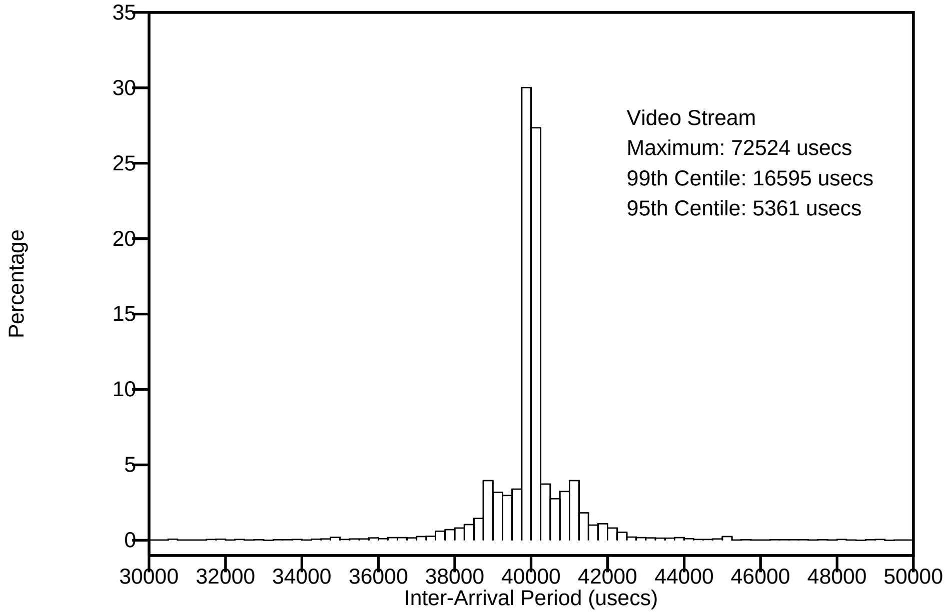


Figure 5.5: Distributions for Experiment 2.

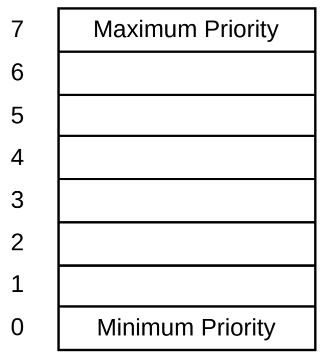


Figure 5.6: Thread Scheduling Priority Levels.

same priority as its related input thread.

In **Experiment 3**, the distribution for an audio stream travelling via the synchronisation server was measured. The input and output threads both ran at priority level 1 and hence could be preempted by the background threads, but not by each other. This was repeated in **Experiment 4** using a video stream instead of an audio stream. The distributions are presented in Figure 5.7. As stated in the previous section, the total number of active streams in these two experiments (and those which follow) was four, providing consistency with the results of the initial experiments. In the distributions which are presented in this and subsequent sections, the measurements from Experiment 1 are shown in outline form in order to ease the task of making comparisons. The improvements are clearly evident from the distributions of Experiments 3 and 4.

The default buffer length for the experimental programme was set to 10ms, selected after taking account of expected jitter and the scope for a small amount of drift before an adjustment is made to deal with a frequency offset. Based on the results of Section 5.4.1, one would expect this length to be sufficient for satisfactory jitter smoothing, i.e. without having to throw away a significant number of segments which are deemed to be late. In these experiments, the number of such excessively late samples for each stream was insignificant, consistent with the initial measurements.

The chosen length also accommodates the two additional sources of jitter which are not present in the initial experiments, because the influence of the synchronisation server was not in evidence. When a CFR source node transmits a cell, it may be informed on the return of the slot that the destination was unable to accept it. This is usually because the previous cell which arrived is still waiting to be removed from the destination's receive FIFO (which can hold a maximum of one cell). If this occurs, the CFR hardware will repeat the transmission attempt. After

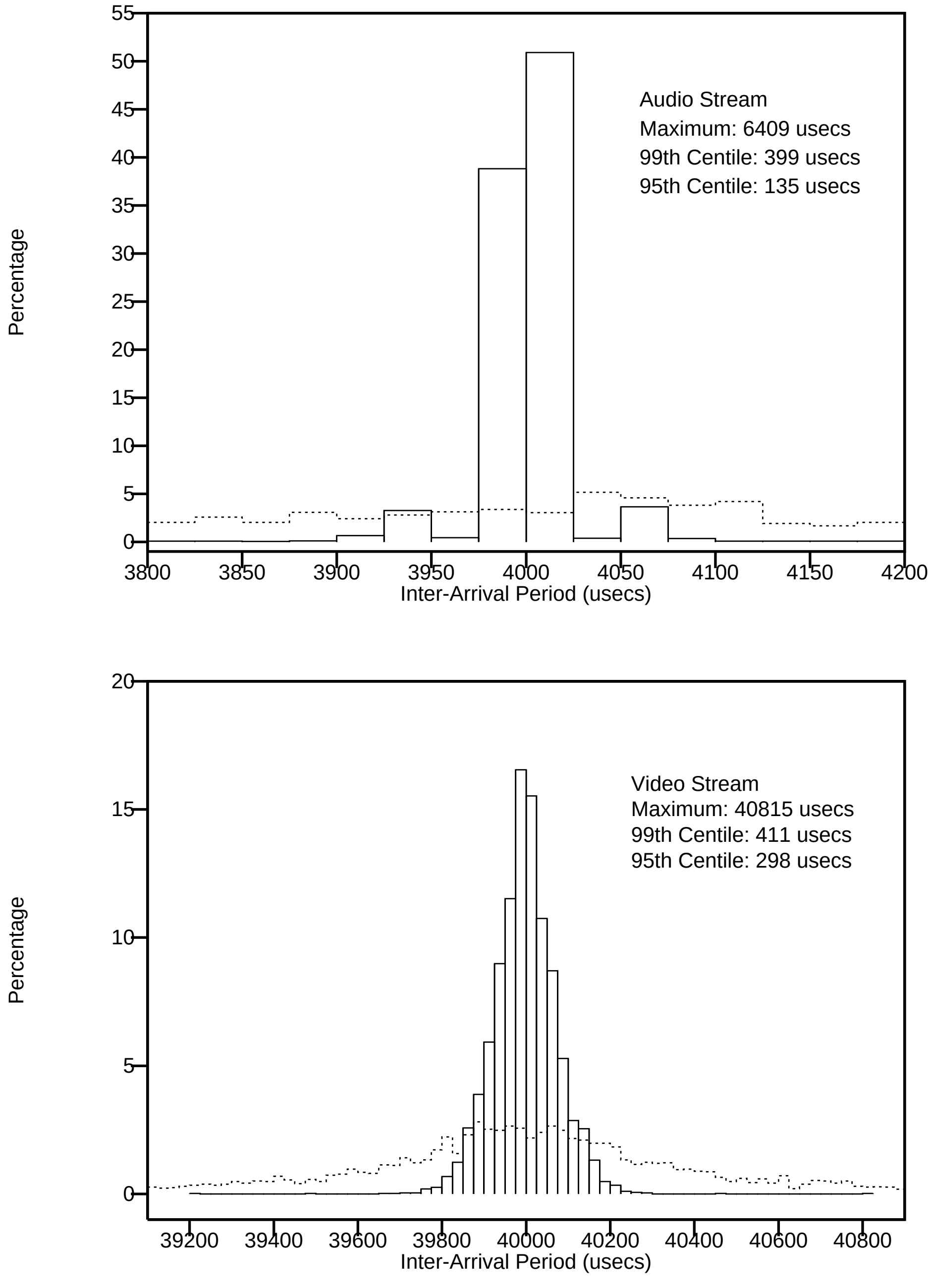


Figure 5.7: Distributions for Experiments 3 & 4.

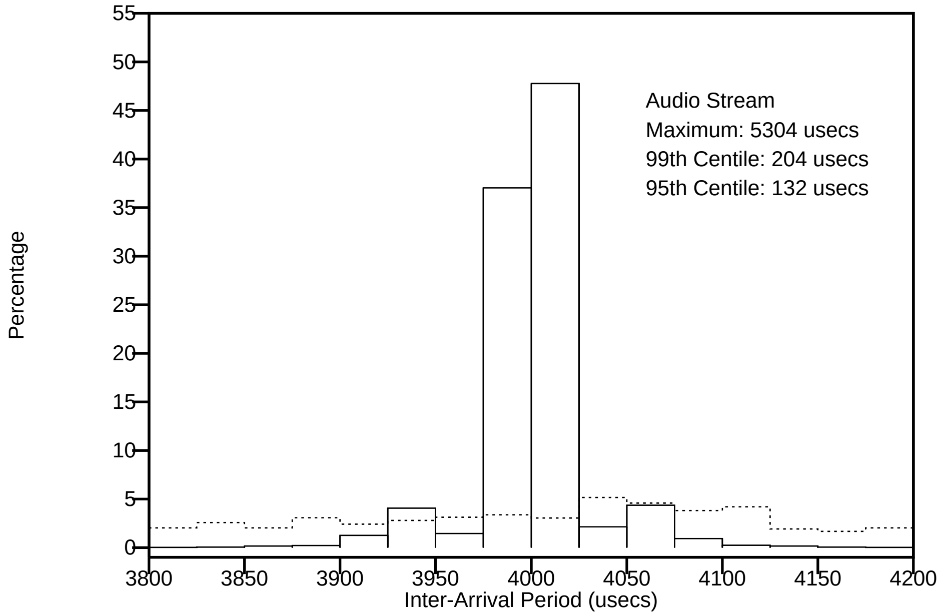
a pre-determined number of such retries, the sender gives up and is said to have received a TOG (Thrown On Ground) for the cell. At this point the CFR device driver can be configured to repeat this transmission process up to a preset upper limit. With the experimental apparatus as described, TOGs are most likely to occur when the gateway on the private CFR is forwarding cells to the synchronisation server. The probability that the latter will cause a greater number of TOGs than the measurement servers is due to the fact that it must additionally:

- deal with transmit as well as receive CFR interrupts;
- handle multiple streams simultaneously;
- service clock interrupts (at a high priority).

This leads to additional jitter because of retries which are made by the gateway node and the ensuing increase in the size of its cell queue. Once the maximum number of (hardware and software) retries for a cell have been exceeded it is discarded, with the result that the sink protocols will consider the containing segment to be lost. A more drastic origin of cell loss occurs when severe contention causes the statically allocated cell queue at the gateway node to overflow. These losses were minimised in the experimental configuration by tuning the retry values used at the gateway node. Hence, unless stated otherwise, it can be assumed throughout the experiments that such losses either do not occur or are negligible. The implications of having a single cell CFR FIFO are also discussed in [McAuley89, Dixon91].

Another cause of jitter for an incoming segment is due to the latency variations in scheduling the related input thread. Unsuitable scheduling at this point can result in the segment being placed in the elastic buffer after it is due to be dispatched. Consequently, when the output thread is scheduled the segment will be discarded for having excessive jitter. In the extreme case, a build up of arriving segments will eventually overflow the capacity of the kernel I/O buffers if the input thread is not allocated sufficient CPU resources. In the prototype, this situation can also occur for the output thread, as each segment in the elastic buffer occupies the kernel I/O buffer in which it was placed upon arrival. The default value for tolerable jitter in the experiments was set to $500\mu s$, avoiding losses due to relatively small jitter excesses. Thus, a segment which exceeds its target dispatch time by up to this value is viewed by the output thread as being on time. Note that this bound can be violated by variations between the time of checking that a segment is on time and its actual transmission. Appropriate scheduling of the output thread can minimise this problem.

In **Experiment 5** the synchronisation server performance for an audio thread was measured again, but this time the stream handling threads were assigned a pri-



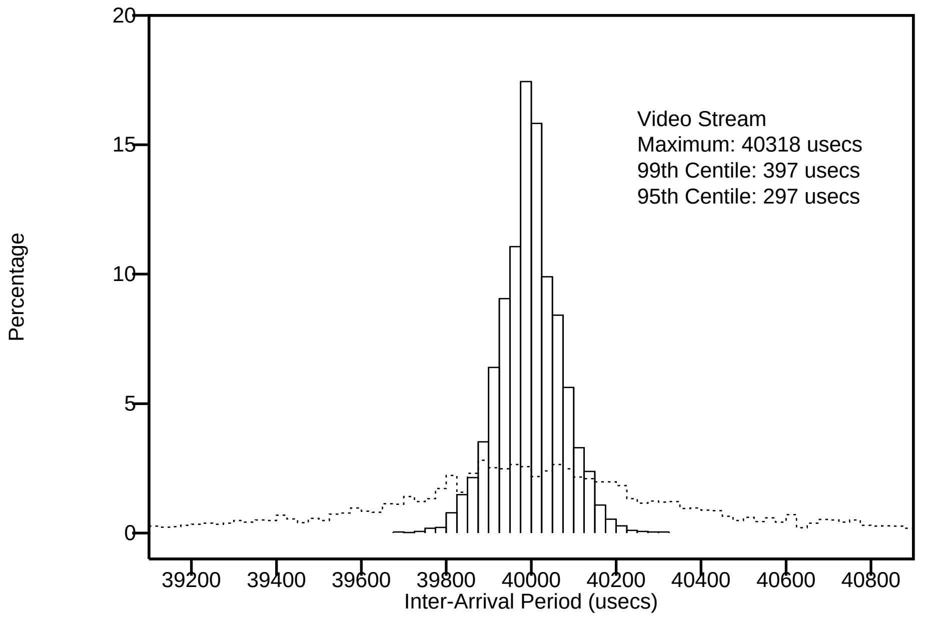


Figure 5.8: Distributions for Experiments 5 & 6.

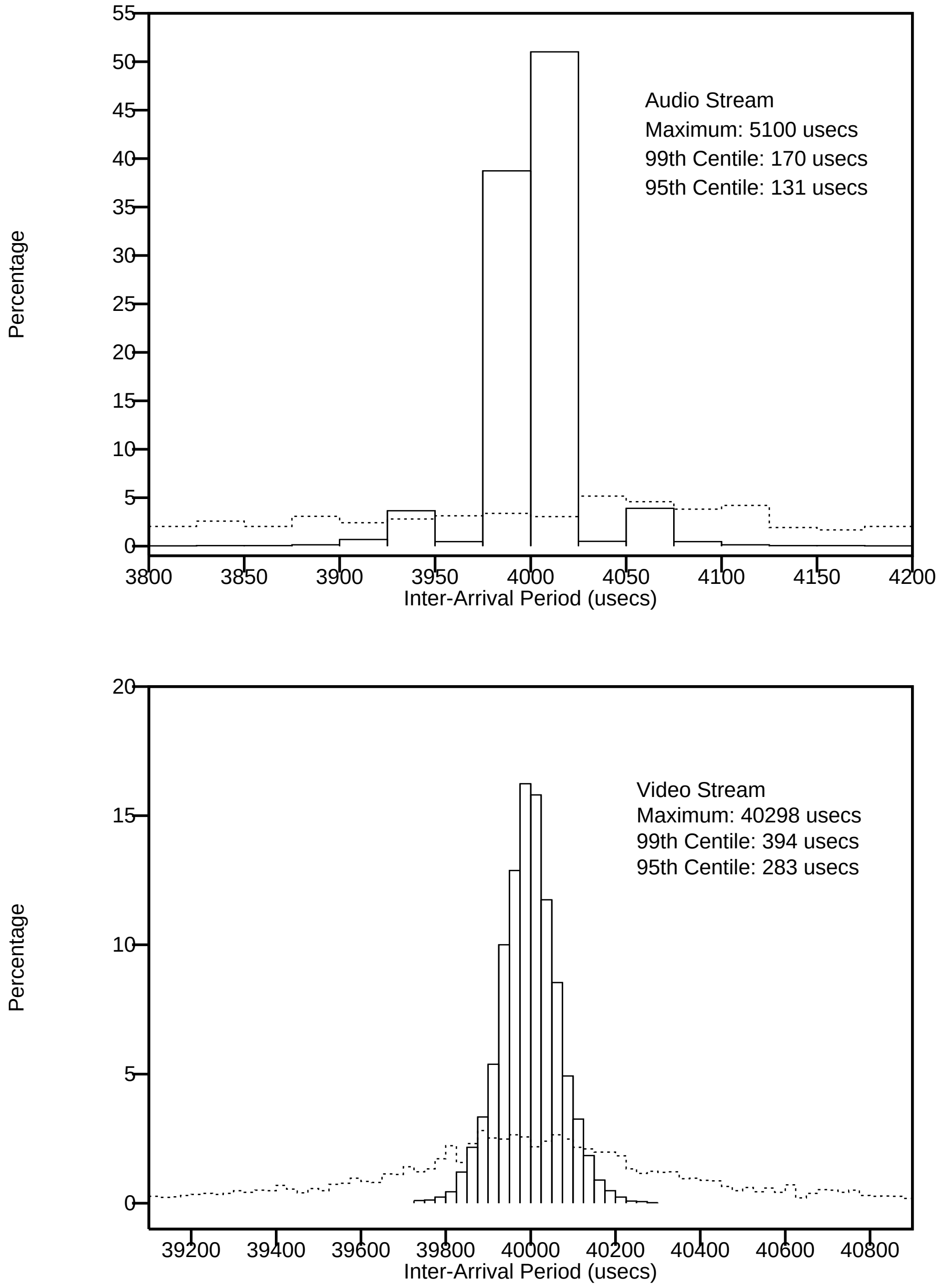


Figure 5.9: Distributions for Experiments 7 & 8.

ority of 4. Hence, they preempted the background threads whenever they became runnable. **Experiment 6** repeated this experiment for a video stream. As expected, the inter-arrival distribution appears relatively unchanged (see Figure 5.8), but an improvement is evident upon examining the results. This was augmented further in **Experiment 7** and **Experiment 8**, by executing the output thread at priority 5, one level above the input thread. This ensured that the process of dispatching a segment was not delayed by the activities of any other threads.

A notable feature of the results for each of the audio experiments is that the maximum inter-arrival time is large in comparison to the overall distribution. This happens when the input thread is resumed later than the time it requested and is due to an implementation feature of the kernel timing facilities. This defers processing of the sleeping thread queue if it is unsafe to do so, as described in Section 5.2.2. When this occurs the queue is not processed until the time of the next clock interrupt at the earliest, in this case 1ms later. Measurements for the experiments in this section show that the resultant latency exceeds the tolerable jitter for less than 3 segments in every 10000.

5.4.3 Multiple Stream Operation

The collection of experiments in this section aim to evaluate the performance of the server when performing intra-stream synchronisation for an audio and video stream simultaneously. Investigations regarding the use of priority were continued from the previous section, while subsequent experiments concentrated on the shortcomings of the kernel timing facilities.

5.4.3.1 Scheduling Priorities

The combinations of priorities which were selected for the next set of experiments are summarised in Table 5.2. The results should be viewed in combination with the details of operational performance for each stream as given in Table 5.3. The quantity represented by α in that table, is the maximum recorded percentage of segments which were discarded by the appropriate output thread because they were excessively late. The column entries denoted by β provide the maximum recorded percentage of segments which were excessively delayed because the output thread was woken later than the target time it requested. These values have been rounded up to the nearest 10th of a percent.

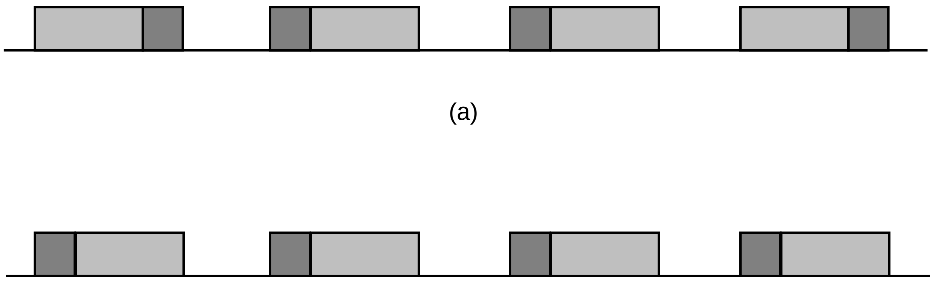
The combination of **Experiment 9** and **Experiment 10** confirm that giving a higher priority to output threads is also beneficial in a multiple stream situation.

Experiment	Audio		Video	
	Input	Output	Input	Output
9	4	4	4	4
10	4	5	4	5
11	4	6	4	5

Table 5.2: Thread Scheduling Priorities for Experiments 9 to 11.

Figure 5.11 shows the performance when the input and output threads for both audio and video were executed at priority level 4. Note the expected improvement, evident from Figure 5.12, when priority level 5 is used for both of the output threads.

In **Experiment 11**, the audio output thread was assigned a priority of 6, meaning that it would preempt the video output thread as well as both input threads. Surprisingly, the resulting distributions showed benefits for video as well as audio. This can be understood by considering the transmission sequences as exemplified in Figure 5.10. In (a), the situation from Experiment 10 is shown, where it can be seen that the order in which a waiting audio and video segments are transmitted is not fixed. Meanwhile, in (b), a waiting audio segment is always transmitted ahead of waiting video, thus reducing the variation in delay encountered by the video thread after being woken up. The increase in the number of video segments which encounter this additional delay also accounts for the small increase in its β value.

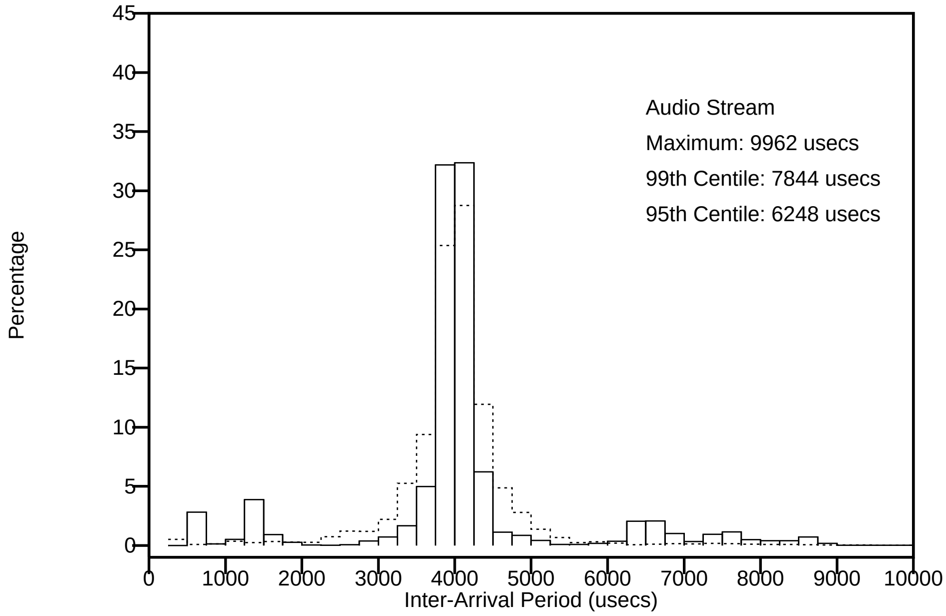


(a)

(b)

Figure 5.10: Effect of Priority on Transmission Sequence.

Of particular concern with these results are the significant values of β for both the audio and video streams. In particular it is clear that timer queue processing is deferred on some occasions during several successive clock interrupts, as evidenced by the maximum inter-arrival times. The next section will address this problem.



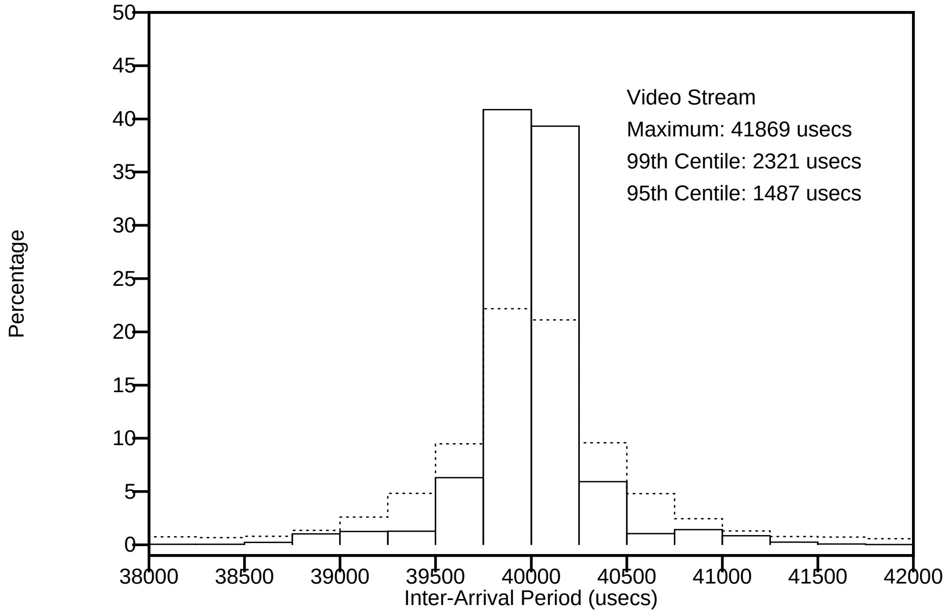


Figure 5.11: Distributions for Experiment 9.

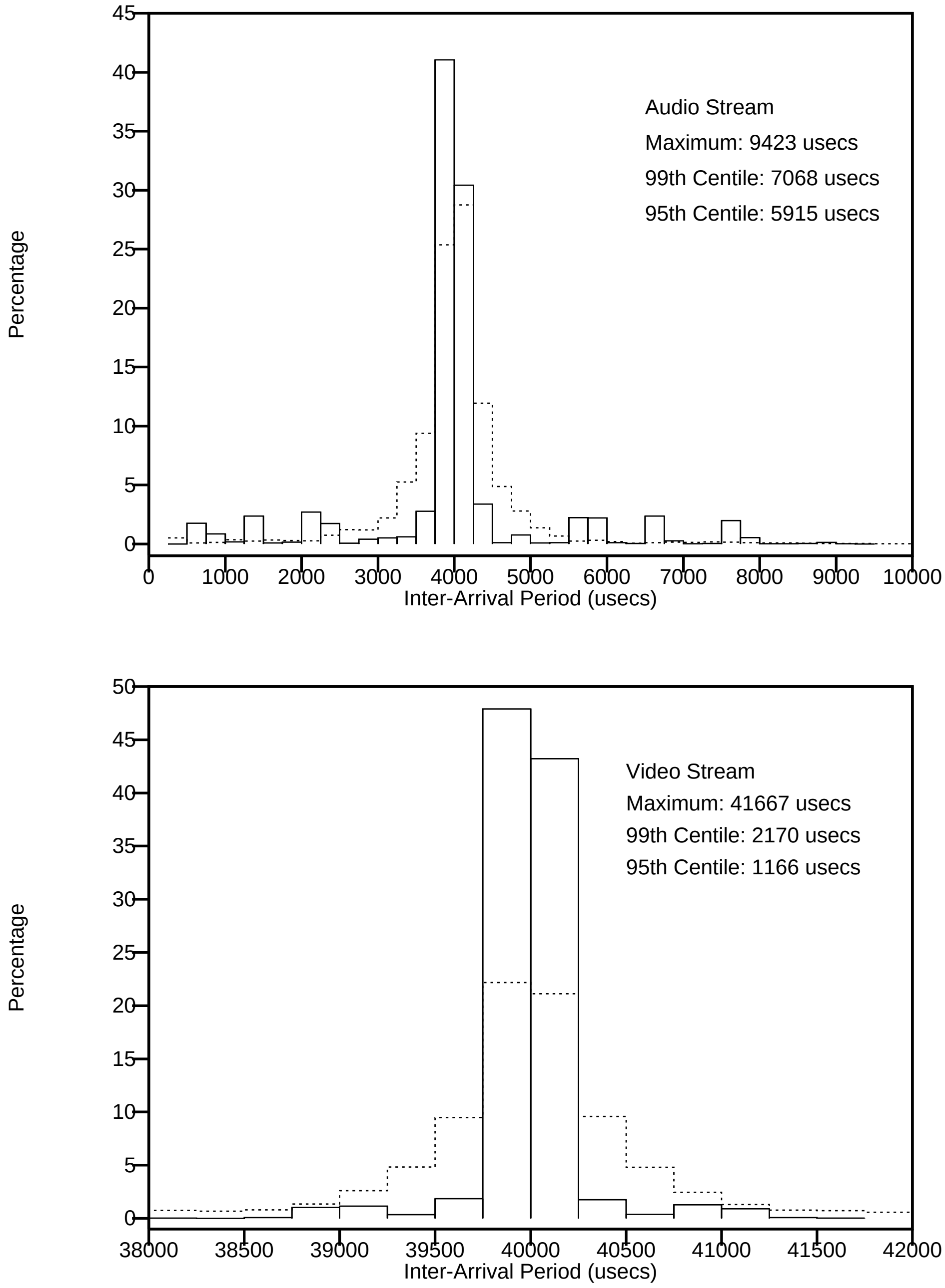


Figure 5.12: Distributions for Experiment 10.

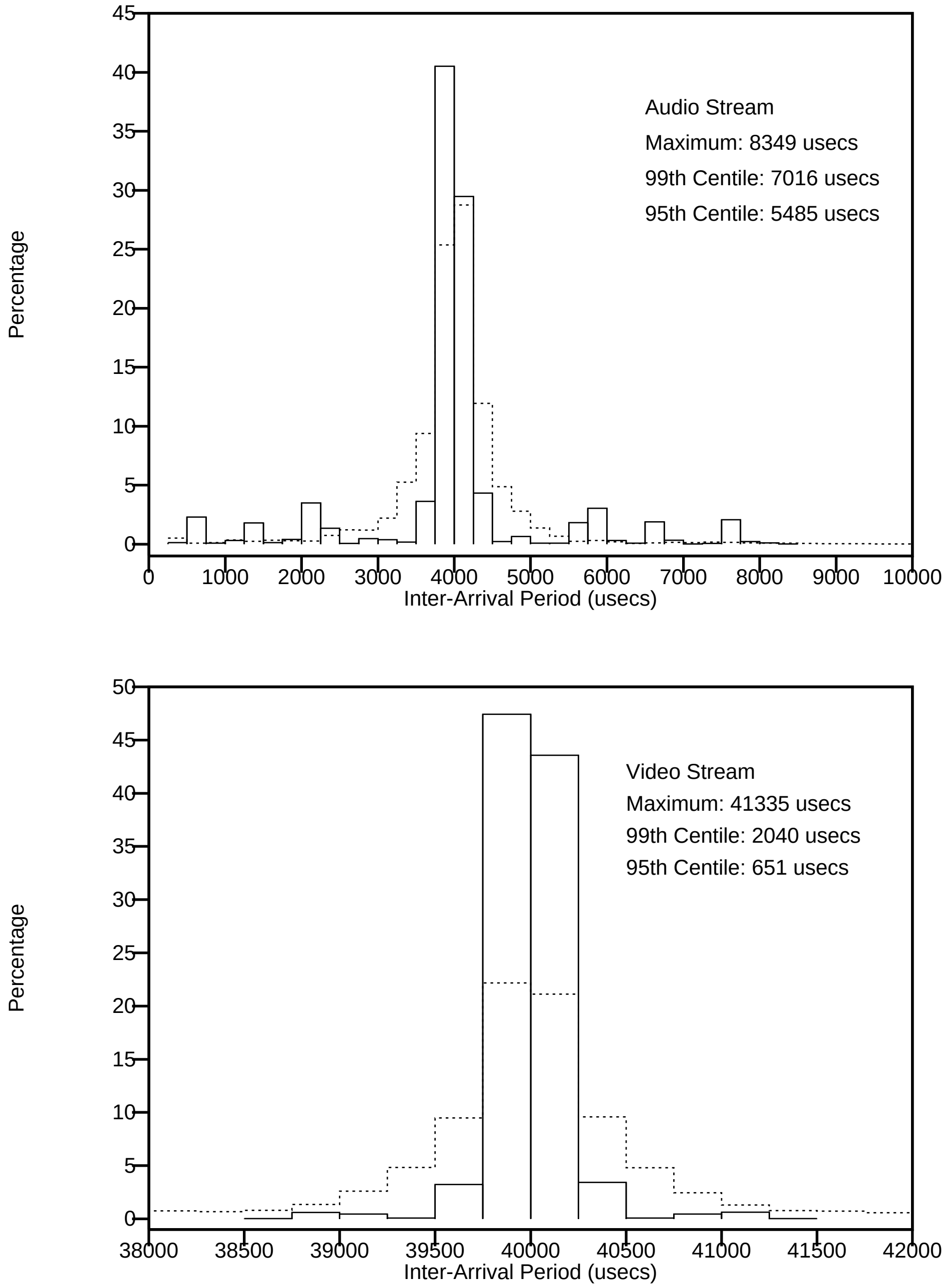


Figure 5.13: Distributions for Experiment 11.

Experiment	Audio		Video	
	α (%)	β (%)	α (%)	β (%)
9	1.8	11.7	0.3	17.4
10	0.2	10.9	0.1	24.4
11	0.1	5.2	0.1	25.0

Table 5.3: Operational Results for Experiments 9 to 11.

5.4.3.2 Kernel Timing Support

The experiments in this section aim to characterise the source and extent of the problem which caused output threads to be woken excessively late.

In **Experiment 12** the stream agent was modified so that it discarded, rather than dispatched, those segments for which it was woken excessively late, i.e those which contributed to the value of β. While the ensuing losses would not be generally acceptable (particularly for audio), the result confirms that the behaviour of the timing facility has a severe influence on the observed performance of the synchronisation server.

There are a number of solutions to this problem. Increasing the frequency of clock interrupts would allow a reduction in values of β. However, this results in an increase in the software overheads of dealing with clock interrupts, which in turn detracts from the servicing of lower priority interrupts. In particular, this would cause a sending node to produce a greater number of CFR TOGs when transmitting to the synchronisation server. A more suitable solution is to process the timer queue as soon as possible after the clock interrupt has occurred. This could be implemented when lowering the flag for deferred timing, by checking whether an interrupt occurred since the flag was raised. If so, the queue can be processed immediately or perhaps by a high priority kernel thread. In the longer term, the need to provide accurate timing facilities without impinging on stream throughput requires further attention. The problem could be ameliorated by suitable hardware support to maintain the time and provide a programmable wakeup interrupt.

Experiment	Audio		Video	
	α (%)	β (%)	α (%)	β (%)
13	0.1	2.6	0.1	0.4

Table 5.4: Operational Results for Experiment 13.

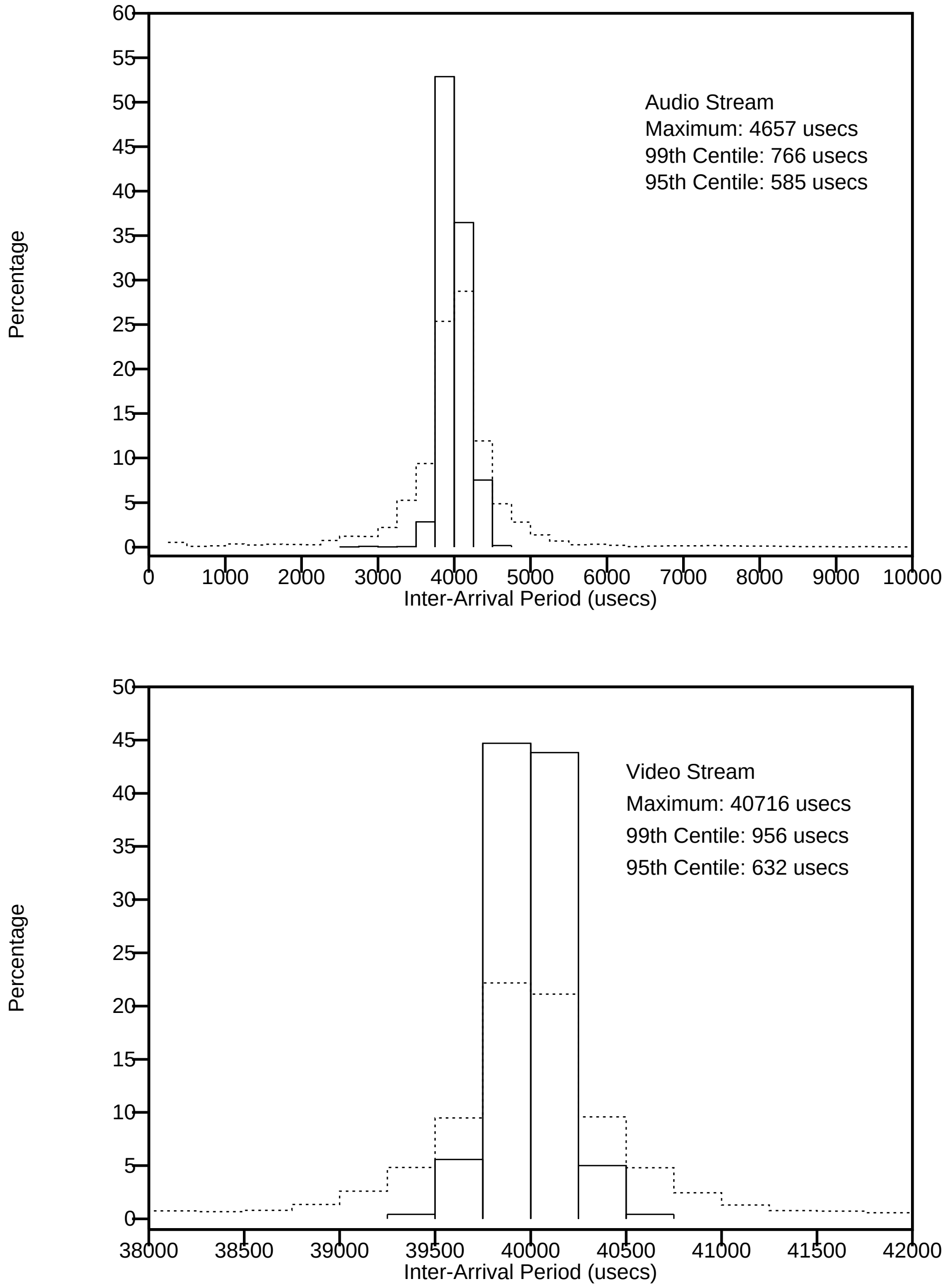


Figure 5.14: Distributions for Experiment 12.

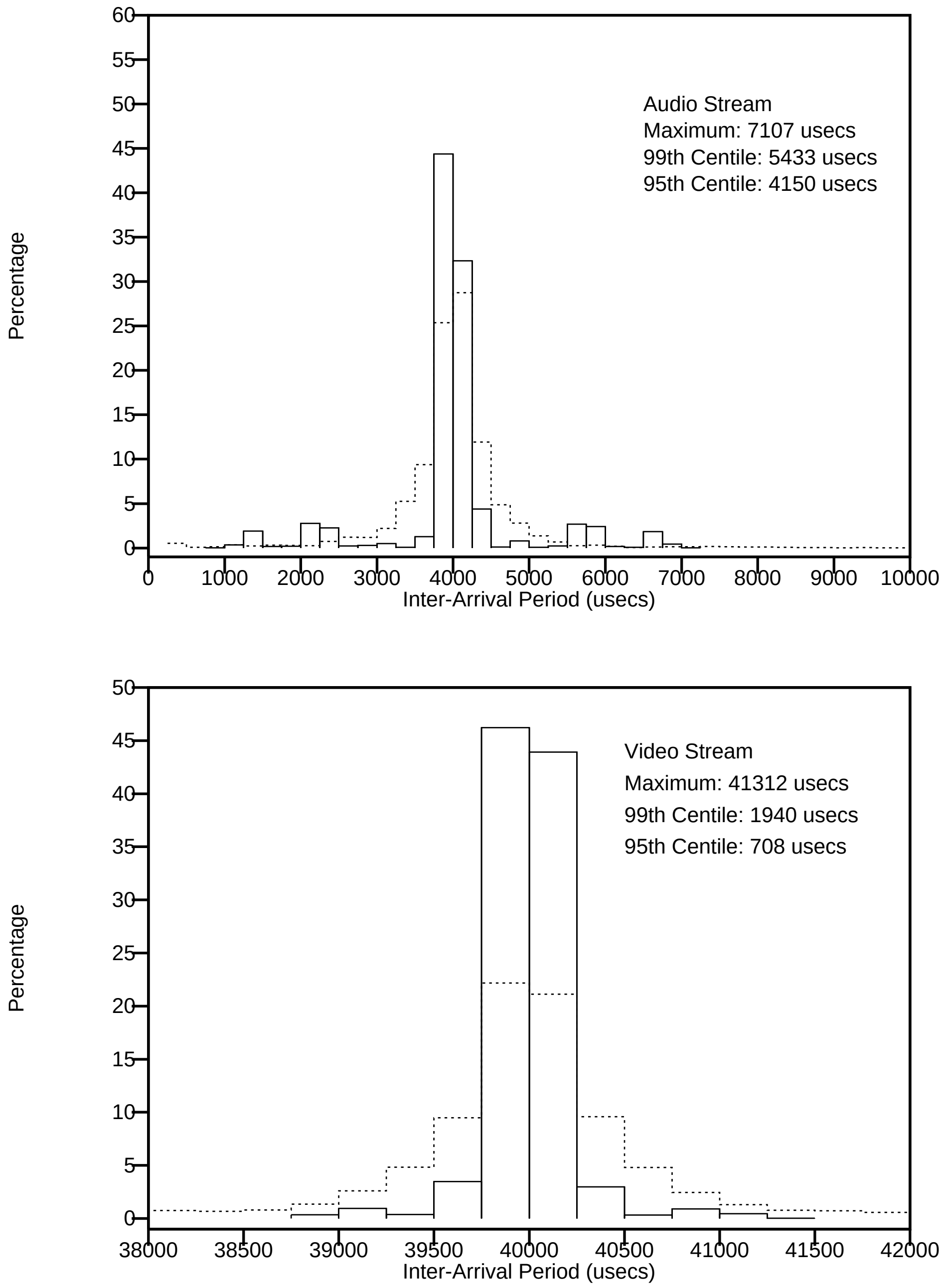


Figure 5.15: Distributions for Experiment 13.

Experiment 13 repeated Experiment 11 to show how the value of β (and hence discarded segments) can be traded against increased jitter tolerance. However, in this case, the tolerable jitter was changed to 1500μs from the default value of 500μs. The result and distribution (provided in Figure 5.15) should be compared with Figure 5.13 to observe the improved performance. The operational results for this experiment are given in Table 5.4 and show a substantial reduction in β for both audio and video. Naturally, this was achieved at the expense of a wider distribution of inter-arrival times than those of Experiment 12.

5.4.4 Inter-Stream Operation

Experiment 14 showed the operation of the inter-stream synchronisation algorithm. For this experiment the CMFS was instructed to record (and synchronise) an audio and video stream from a Pandora Box on its local CFR. These streams were later played back to the synchronisation server by issuing separate commands to the CMFS. This introduced initial skew which resulted in the video being presented approximately 1000ms ahead of the related audio.

Both stream agents were configured with a minimum buffer length of 10ms and a maximum value of 2000ms. The time stamps of the first recorded segment in each stream were provided to the synchronisation server, allowing the group agent thread to map each stream to a common presentation time. This thread was executed at priority level 3, and operated by comparing the streams every 1000ms and making an 80ms adjustment when it detected skew to be in excess of 80ms. The frequency and value of these adjustments were selected so that the effects of the algorithm would be highlighted in the measurements. In practice, a choice of smaller values is more likely in order to bound large variations in inter-transmission times and hence jitter.

Figure 5.16 clearly shows the initial skew and the convergence caused by the operation of the group agent. The enlarged view (in Figure 5.17) shows the video stream being forced to pause in a stepwise manner in order to synchronise with the audio stream. This is achieved between 12 and 13 seconds into the presentation, being consistent with the expected time to eliminate the initial skew using the selected adjustment parameters.

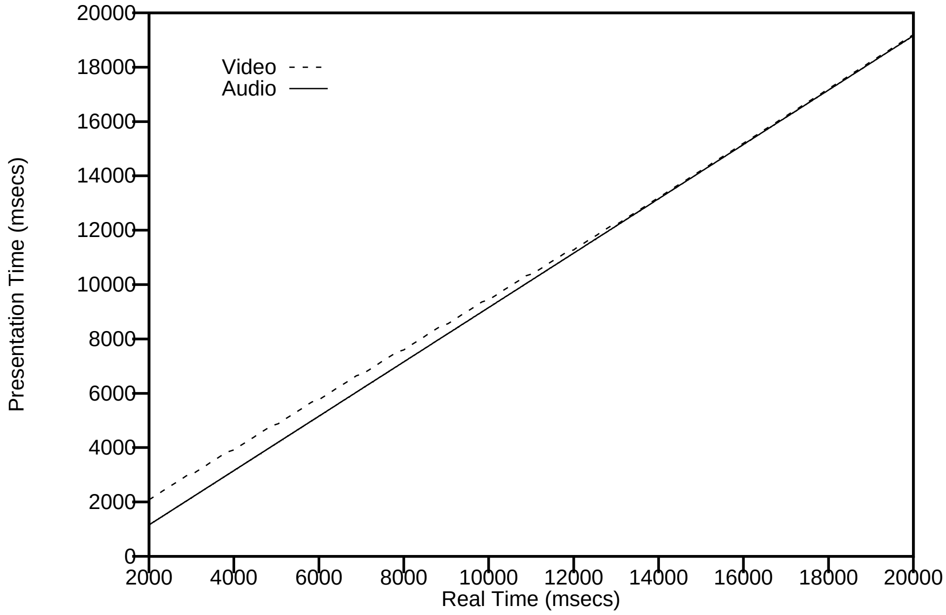


Figure 5.16: Result of Experiment 14.

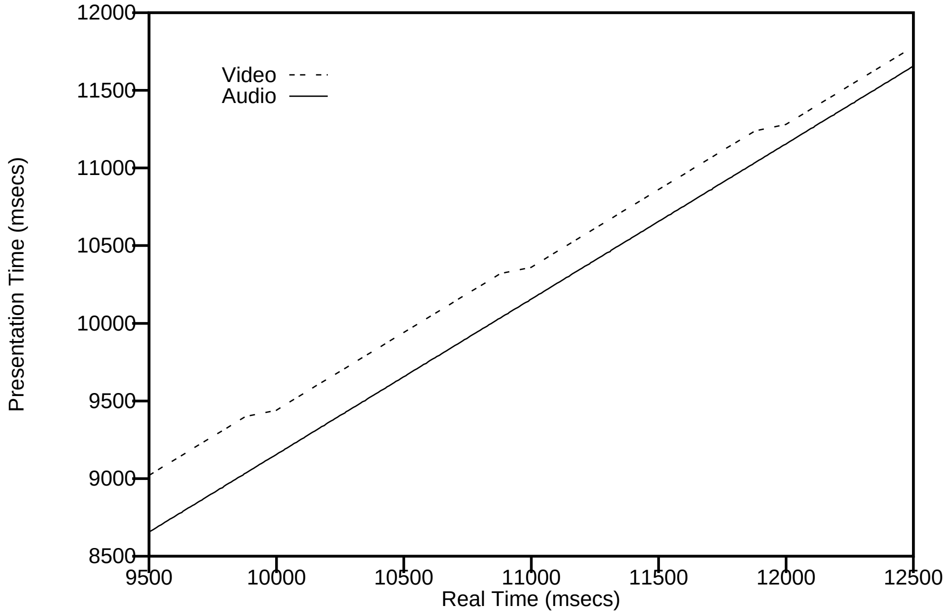


Figure 5.17: Result of Experiment 14 (Detail).

5.5 Summary

This chapter provided an experimental evaluation of a prototype synchronisation server. An overview of the target environment and kernel were first given, followed by a description of the implementation. The programme of experiments concentrated on examining the performance of the prototype when performing intra-stream synchronisation. In particular, the influence of the choice of scheduling priorities was investigated and conclusions were drawn. Problems from unsuitable kernel timing facilities were identified and a number of solutions were proposed. Finally, the ability of the group agent to perform inter-stream synchronisation in a satisfactory manner was clearly shown.

Chapter 6

Comparative Evaluation

This chapter describes recent research for the design and support of systems for synchronisation. A critique of closely related work is presented, allowing a comparative evaluation of the proposed design. In combination with Chapter 5, this completes the process of evaluation.

6.1 Related Research

This section presents an analysis of related research, concentrating on systems which perform synchronisation, rather than the higher level task of requirements specification. A general overview is given in Section 6.1.1, followed in Section 6.1.2 by a detailed examination of five projects which are closely related to the work of this dissertation.

6.1.1 Overview

Related research can be viewed within the framework illustrated in Figure 6.1. Operating below the specification level, a number of projects concentrate on the semantics of operating system and programming language primitives to support synchronisation. This work is motivated by the need to acknowledge the importance of time constrained synchronisation in presentations which involve continuous media.

In [Hazard91], existing work on formalisms for *reactive* systems is used as a basis for programming time constrained synchronisation. A reactive system is defined as one which interacts in an ongoing manner with its environment, expressed in terms of input and output event occurrences. This is in contrast with the more conventional

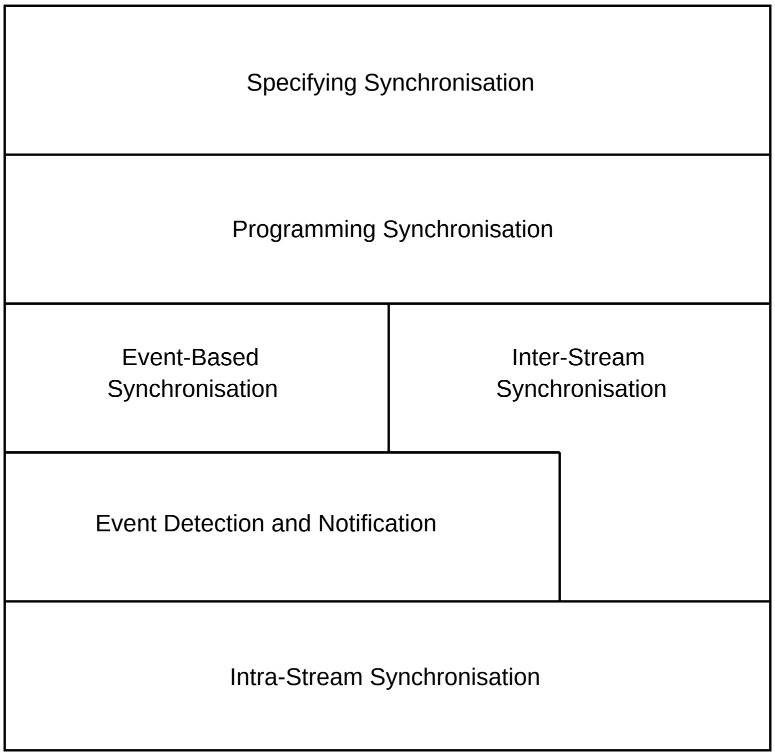


Figure 6.1: Synchronisation Framework.

view of computation, in which a set of output data is produced from supplied inputs. An extended example is given, describing the use of a synchronous language, incorporating temporal constructs which are used in combination with event notifications. It is suggested that the approach can handle inter-stream synchronisation within the event-based framework. The issues of employing a synchronous language in an distributed system, notably ANSA [ANSA89], are considered. It is proposed that multi-media applications should embody a *reactive kernel* to handle all synchronisation and real-time aspects of operation, thereby allowing the synchronous properties of the programming language to be supported.

Conventional synchronisation primitives are reviewed by [Steinmetz90], for use in the context of multi-media. The concept of *restricted blocking* is proposed as a semantic extension in order to allow the desired behaviour of a waiting stream to be explicitly directed. Thus, for example, when a video stream is blocked, it may be deemed suitable to display the most recently presented image until the stream is resumed. A further extension is described in which time constrained stream synchronisation is supported by allowing bounds on the minimum, maximum and average delays between event occurrences to be supplied. This is exemplified in Figure 6.2, where minimum and maximum constraints are given for the delay of the event occurrence on stream B in relation to that on stream A.

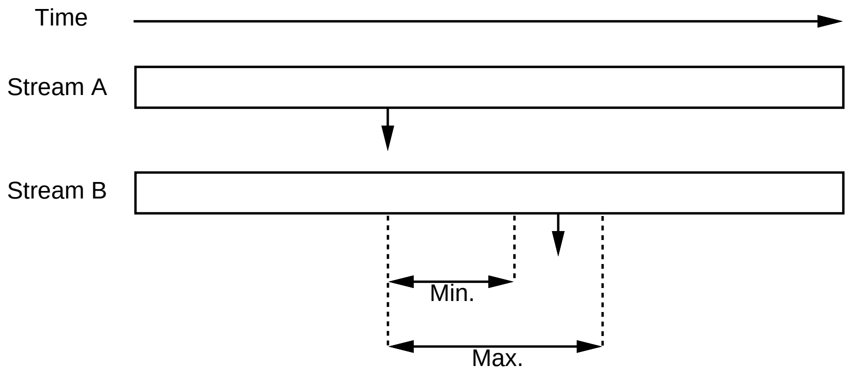


Figure 6.2: Time Constrained Event Synchronisation.

The availability of event information provides a highly adaptable foundation for higher level synchronisation. This is illustrated by the work reported in [Nicolaou90b], which defines an architecture with support for notification of stream related events. The proposal consists of a two-level event model, in which a stream handler performs a mapping between physical and logical *synchronisation frames*. A physical synchronisation frame is stream dependent, and the application can select the appropriate granularity of the mapping to suit its needs. Each logical synchronisation frame results in an RPC call back to the application, which may be used to perform synchronisation expressed in terms of programming language synchronisation primitives.

The system described in [Little90a] directly executes a high level statement of application synchronisation requirements. Synchronisation is specified using a logic of temporal intervals and takes the form of a timed Petri Net [Little90b]. These specifications are stored in a database, from which they are extracted and executed at presentation time, relying on the use of a communications protocol which provides highly predictable end to end delays for each stream [Little91]. Event information is obtained using logical synchronisation points within each stream, based on the proposal from [Nicolaou90b] described above.

A common theme in each of these projects is the important role of event notifications. The need to provide a mechanism which is flexible, yet temporally constrained, is highlighted.

6.1.2 Closely Related Projects

The following sections describe the five related research projects which are most closely related to the work of this dissertation. A critical examination of each project allows a comparative evaluation of the proposed design. The discussion deals only with functional issues, since experimental information is not available for these systems.

6.1.2.1 ACME I/O Server

Abstractions for Continuous Media Extensions (ACME) [Anderson91b] is a workstation server for stream I/O, similar in style to the X Window System. Implemented as an ordinary UNIX process, it handles streams in user space under control of the window system, and transfers them using the standard Transmission Control Protocol (TCP). ACME is multi-threaded, with threads assigned to handle network and device I/O for each stream. These are scheduled preemptively within the server, using an EDF algorithm. A thread inherits its deadline from the time stamp of the data it is currently handling.

Inter-stream synchronisation is supported using a *Logical Time System* (LTS) to group related streams. An LTS is operated by ACME based upon the following application supplied parameters:

- N: per-stream initial buffer length;
- T: per-stream logical time offset;
- S: maximum acceptable skew;
- M: identifier for master stream.

Initially, output from an LTS is postponed until the contents of each stream buffer has reached its minimum value of N bytes. It is intended that this value should be selected so that the expected jitter of the incoming stream can be accommodated.

Once presentation has begun, time stamps for each stream are converted to a common scale using T, thus allowing skew to be detected. In the prototype, the advance of this scale is driven by a nominated master and it is used by each slave to decide if it needs to skip or pause to minimise skew. This decision uses a rate matching algorithm [Anderson91a], which is executed every time data on a stream is about to be presented. If S is exceeded, all stream output is paused, with the objective of allowing the offending connections to catch up, after which normal operation resumes. Within ACME, a stream buffer is allowed to overflow or underflow as long as this does not cause excessive skew.

Event based synchronisation is supported by making reference to a specified LTS time value. Upon reaching this time, it can be arranged to have an event handling thread report back to the client or invoke a selected window system operation. These threads do not have deadlines, but instead are scheduled on a best-effort basis within the server. Other server operations allow the current value of an LTS to be read and its constituent streams to be stopped.

The ACME approach is designed to perform synchronisation for streams which can be routed through a server located at their common destination. Knowledge of stream formats is embedded in the server, allowing timing information to be easily extracted, but complicating the introduction of new stream types and alternative event-based facilities. The lack of appropriate support provided by UNIX, hampers the operation of the prototype server. In particular, data presentation is driven in response to UNIX signals, the delivery of which can be highly unpredictable in terms of delay. The duration of skip and pause operations in the server is matched to that of the signal frequency for the corresponding device. Consequently, audio is made to suffer large adjustments of approximately 32ms. The initial implementation was prone to noticeable timing errors and data loss which occurred in the presence of other concurrent activity in the system [Govindan91]. A technique whereby kernel scheduling could be influenced by the priorities of threads existing in user-space was proposed to remedy the situation.

It is unclear whether ACME operation takes sufficient account of stream QOS requirements. The unbounded waiting period to achieve initial synchronisation can lead to excessive delays and/or buffer overflow. If the latter condition occurs, the prototype server sends an exception message to the client [Anderson89a], relying on it to implement some form of flow control. In addition, during operation of an LTS, it seems that skip and pause adjustments are made without regard for the length of a slave's buffer. Decreasing the size below N bytes increases the possibility of excessive jitter, while at the other extreme, the acceptable end to end delay for the stream can be violated. It is reported in [Govindan91], that the delay in using ACME exceeds that which is acceptable to a telephone conversation client.

6.1.2.2 MODE

Multimedia Objects in a Distributed Environment (MODE) [Blakowski91] is designed to support the specification and performance of multi-media presentations.

A graphical editor is used to describe a presentation in terms of the desired spatial layout and synchronisation relationships between a collection of *media objects*. These relationships take the form of a list of *synchronisation points*, each of which represents a coupling of *reference points* within two or more objects. In other words, a synchronisation constraint will be satisfied if the related objects have reached these reference points at the same occasion during the presentation. Examples of reference points include the start and end of displaying a video frame or still image. In addition, a special timer object has reference points at 1ms intervals, which allow synchronisation along a time-line if desired. Synchronisation points are manipulated using the editor, which can display objects along a time axis, indicating their expected durations.

The synchronisation requirements are stored by the editor in text form and passed to a *synchroniser* for execution. This program runs at the node where presentation is to take place, and manages both intra-object and inter-object synchronisation. The former aims to ensure the periodic presentation of, for example, frames within a video object. Within the synchroniser, a thread is assigned to perform this task for each stream, with a scheduling priority associated with the delay sensitivity of the object being handled. These threads interact using a signalling mechanism to achieve inter-object synchronisation. When a thread reaches a reference point which is associated with a specified synchronisation point, it issues a signal to each of the other threads which are supposed to synchronise at that point. It then waits until it receives a reply from each of these threads, at which time it may decide to extend its delay so that they can catch up. They can facilitate this process by accelerating up to the synchronisation point after receiving the signal. This is achieved by temporarily increasing the speed of presentation or skipping the intervening data.

This basic approach of using stream handling threads to interact for inter-stream synchronisation reasons is similar to the design presented in this dissertation. However, there are a number of aspects of the MODE system which manifest themselves as operating restrictions. By specifying the synchronisation requirements in advance of the presentation, it is not clear how additional objects can be dynamically introduced, unless there is some mechanism for informing the executing threads of new synchronisation partners (and possibly additional synchronisation points). Since the signals are used only within the synchroniser it is not possible to simply latch on to an already running presentation. The provision of externally visible signals would allow an event-based approach, gaining the benefit of synchronisation with non-data objects, such as application programs.

The mechanism for implementing synchronisation points is particularly complex. Each thread must be aware of the complete list of synchronisation points with which it is associated. In many cases it should be possible to describe these iteratively, especially when unbounded streams are involved. The set of such points can be altered before the presentation by taking resource availability for synchronisation into account, but once presentation has commenced it appears that this flexibility is removed. This approach makes it difficult to adapt the synchronisation requirements based on prevailing conditions. For example, it may be useful to allow synchronisation points to be skipped during periods of excessive network delays or when operating the presentation in fast-forward mode. This would allow the synchronisation accuracy to be reduced, so that the task of the synchroniser could be lightened. The use of fixed reference points, rather than time, complicates the situation by not allowing an acceptable tolerance for inter-object synchronisation to be expressed. Such a value can be used to considerably ease the task of synchronisation, by allowing the application to indicate its ability to put up with a controlled degree of inaccuracy. Although the system can support objects with an unpredictable duration it is not described how, if dealing with live streams, any initial skew can be detected and removed.

The organisation used for inter-stream synchronisation has a number of drawbacks. It is unclear whether the adjustments performed by threads can be co-ordinated to ensure that QOS requirements are not violated. In particular, it appears that the first thread to reach a given synchronisation point may have to perform an unbounded wait, resulting in a sudden increase in its values for end to end delay and buffer occupation. Similarly, MODE seems not to control the jitter and losses caused by accelerating the other partners in a synchronisation point. These problems are exaggerated by the inability to apply gradual synchronisation, since it appears that progress is halted until synchronisation is achieved. Extending the system to support a concept of stream preference would also be beneficial to allow additional control over these adjustments. The requirement that a signalling thread must block waiting for replies to its signals will not scale well if the system is ever extended for operation over multiple sinks. The latency and reliability of signals and replies, along with the possibility of stream failures, are significant in this respect.

6.1.2.3 Orchestration

Orchestration is a protocol architecture for continuous media synchronisation in a distributed system [Coulson91]. It is described using OSI terminology and is said to exist at a level corresponding to the Upper Layer Architecture (ULA), i.e. the application, presentation and session layers.

From the application viewpoint, synchronisation is initiated by making a request to a High Level Orchestrator (HLO). For each group of related streams, this creates a HLO agent which manages synchronisation from its position on a selected orchestration node. This node is defined to be the source or sink which is common to the greatest number of related streams. In the initial implementation, the streams are required to have either a single source or sink, making this choice trivial. Subsequent application control of orchestration activities is achieved by allowing direct interaction with this agent.

Below this level, orchestration is represented by a Low Level Orchestrator (LLO). A LLO exists on the source and sink node of each orchestrated stream and may interact with other LLOs using per-stream control circuits. These provide guaranteed bandwidth for out of band signalling, and allow predictable source/sink interactions for control of a rate-based transport protocol [Berrocal92]. For each stream, the HLO agent and its local LLO interact at fixed regular periods, allowing skew to be monitored with reference to the clock at the orchestrating node. Rate targets are set for each stream by the agent, which the LLO makes a best effort attempt to have reached before the next reporting point. This may be achieved by skipping buffered data or using rate control to pause the source. Failure to meet a target is a problem which must be dealt with by the agent using an application specified action. In particular, the use of QOS re-negotiation is strongly suggested.

Initial synchronisation is achieved by making judicious use of the transport protocol to allow sink buffers to be primed, in advance of the start of presentation. Other operations allow a group of streams to be stopped, and members to be dynamically added and removed. Event based synchronisation is supported by using a special event field in each packet, the contents of which can be compared by the appropriate LLO with application supplied values. It is suggested that higher level synchronisation can be implemented using a programmable manager which executes an application supplied script based on such event occurrences.

The system appears not to deal explicitly with intra-stream synchronisation, instead relying on suitable QOS support from the transport level [Campbell92a]. In addition, support is only provided for stored streams, thus minimising problems from the significant end to end delays which could be introduced using the rate-based transport protocol, notably at the start of a presentation. As discussed in Section 2.5.1, the effectiveness of performing synchronisation in a protocol stack, as in this system, is dependent of the ability to control the behaviour of subsequent application processing so that streams remain synchronised.

6.1.2.4 OSI Extensions

Support for inter-stream synchronisation within an OSI context is discussed in [Shepherd89]. It is argued that this function is best provided at the transport layer within the receiver's protocol stack. The use of a *synchronisation marker* scheme is initially considered, in which data items to be presented simultaneously are identified by inserting a marker in each stream at the appropriate point. The receiver uses buffering to delay stream data until the corresponding marker has arrived for each related stream, at which point it releases the synchronised data for higher level processing.

A more sophisticated approach which uses a *synchronisation channel* is then described. The information in this channel is carried separately from the stream data and is used to specify the synchronisation requirements. This is done by making reference to points within the streams and indicating whether the associated data items are to be synchronised for simultaneous, sequential or independent presentation. These relationships correspond with the existing proposals for use with elements of multi-media documents [Postel80]. It is suggested that streams should be synchronised initially and that data points could be identified in terms of either the byte offset or packet sequence number.

The manner in which synchronisation could be used as a transport layer service is described. It is suggested that the synchronisation channel interface should allow a time value to be associated with each synchronisation request. For simultaneous presentation this defines the maximum slippage between streams; while for sequential and independent presentation, it limits the allowable pause and the period over which presentation must occur, respectively. A detailed example is given for a service based on the use of synchronisation markers. The need to detect exceptional conditions is highlighted, notably in regard to lost, re-ordered or duplicated markers. In addition, a time-out condition is asserted if corresponding markers are not received within a certain time after arrival of the first such marker. Finally, it is maintained that the proposals can deal with live streams and situations which involve multiple sources.

Many of the comments made in regard to the use of reference points for inter-stream synchronisation in MODE (in Section 6.1.2.2) are also applicable to these proposals. In particular, the possibility that buffering may cause tolerable values for end to end delay (or memory occupation) to be exceeded, does not appear to be handled. It is possible that the proposed time-out mechanism could be extended to incorporate this additional functionality. As remarked in regard to Orchestration in Section 6.1.2.3, operating within the protocol stack requires attention to be paid to QOS aspects of subsequent stream processing, both in the ULA and applications. Similarly, by embedding synchronisation deep in the protocol stack, it complicates the introduction of new stream formats if so desired.

6.1.2.5 X Synchronisation Extension

The X synchronisation extension [Glauert91] aims to allow X Window System applications to co-ordinate data presentation. In the basic X protocol for client/server interaction, data to be presented is transferred within a stream of asynchronous requests to the server. The unpredictable nature of delays between the issue and execution of these requests was the motivation for work on the synchronisation extension, which places particular emphasis on the support of applications which use animation. The extension supports synchronisation by providing a set of integer *counters* which are maintained by the server. A client achieves synchronisation by inserting special operations at appropriate points in the request stream. These operations specify a counter variable and the target value which it must reach before further processing of requests for that stream should be allowed to continue. During this period the server blocks the stream and buffers requests which arrive on it.

Synchronisation within a stream is supported by a special time counter, which is updated by the server at 1ms intervals. This allows the client to insert a controlled delay between operations to display, for example, individual frames of animation. Inter-stream synchronisation can be achieved using counters which are created and advanced directly in response to client requests. This could be used to implement a master/slave arrangement by having the progress of the slave stream driven by counter increments made each time a presentation request is completed for the master. Deadlock situations can be broken by having a stream synchronise using the time counter in combination with a target counter, allowing a time-out mechanism to be implemented.

Event-based synchronisation is supported by the use of an *alarm* facility. This allows a client to register an interest in a counter reaching a supplied value or over an equally spaced set of such values. An alarm goes off when a value of interest is reached, resulting in the dispatch of an asynchronous notification to the registered client. A notification of this form is also used in the general case where synchronisation occurs later than desired. This is detected when a counter value exceeds the target by a client defined *threshold*. The notification tells the client that the server was unable to satisfy its synchronisation request in a timely manner, usually because it was experiencing overload.

Similar in fashion to ACME (see Section 6.1.2.1), this scheme deals with fixed format streams which can be routed through an application-level server at their common destination. Unlike related systems, a stream in this format contains information which controls the operation of the server, for presentation as well as synchronisation purposes. In practice, this demands the support of a reliable transport protocol. Consequently, counter update requests do not get lost or corrupted, avoiding time-outs and the excessive delays in stream processing which could otherwise occur. However, the price of this reliability in terms of protocol retransmissions will itself be the cause of extra delays. In addition, because of the interleaved nature of control requests and presentable data, it is not possible to trade a degree of unreliability for decreased delay if so desired. The discussion in Section 3.2.1 considered these issues in detail.

The extension does not attempt to detect or deal with jitter, presenting data as accurately as it can on a best-effort basis. However, it is possible for a client to minimise jitter by setting up a pseudo elastic buffer within the server. This can be achieved simply by inserting an appropriate delay between the arrival and presentation of the first data on a stream. A greater degree of control over any residual jitter could be achieved by offering support for stream priorities.

A frequency offset may exist between the client and server, eventually causing the server synchronisation delays to increase or decrease. This situation can be managed by having the client make a phase or frequency adjustment (see Section 2.4). In particular, a phase adjustment may be easily achieved by using a synchronisation request to skip or pause with respect to the time counter. Note that this request is only processed after waiting its turn behind previous stream requests which may be waiting in the server buffer. Detecting a frequency offset places a burden on the server to dispatch alarm calls for increments to the time counter. This process is prone to inaccuracies due to unpredictable end to end latencies for these messages.

For inter-stream synchronisation, the onus is on the clients to co-ordinate the relationship between counter updates in advance of the presentation. This is necessary to allow rate matching and the elimination of any initial skew which may occur. This scheme presents a problem when dealing with live streams, unless the expected difference in their actual start times can be enforced using algorithms of the form described in Section 3.6. During synchronisation, thresholds may be used to determine if a master stream is running ahead of the slave. However, there appears to be no equivalent mechanism to ascertain the amount of time for which the slave is forced to wait for the master. This is important to allow the client to take action so that tolerable end to end delays for the slave are not continually violated.

6.2 Operating System Support

The satisfactory performance of synchronisation hinges on the supporting role of the operating system and network. Work on ATM networking, multi-service protocols and communications QOS was described in Chapters 1 & 2. Operating system designers have also been forced to reconsider the temporal aspects of data processing, in the light of applications which need to handle large volumes of data for I/O purposes. Movement of this data within an operating system can be supported by techniques of the form described in Section 2.5.1. However, the time-constrained nature of dealing with continuous media is of greater relevance in the present context. The general trend has been to use a foundation based upon existing work for conventional real-time systems [Audsley90]. However, this work is mainly orientated towards situations where there is considerable information about resource requirements and availability. In particular, such an approach is necessary to adequately support hard real-time systems, where failure to operate within specified time constraints can have catastrophic consequences.

A number of systems aim to reproduce this type of arrangement, by dedicating resources or allowing an application to negotiate a guarantee based on reservations made in advance. The use of a dedicated co-processor is advocated in [Bulterman91] as a solution to this problem, with particular emphasis on the support of synchronisation. Predictable performance, coupled with the benefit of running standard UNIX on the central processor are cited as the main benefits. The work described in [Anderson89b] uses a resource model (implemented in the DASH kernel) which allows applications to reserve resources in an end to end manner. This model is based on a workload characterisation and set of performance requirements. Within a resource, scheduling is based on deadlines which are assigned individually to messages within a stream. A similar approach, implemented for the YARTOS kernel, is described in [Jeffay91]. This provides a guarantee on the latency of message receipt and processing, by executing a scheduling algorithm prior to running the application.

An extension to the Mach kernel for time constrained signal handling is discussed in [Nakajima91]. This is used in combination with a preemptive deadline scheduling algorithm to support multi-media device access from user space. Related work provided a real-time version of the Mach kernel [Tokuda90]. This is a comprehensive re-design of the facilities for thread scheduling, synchronisation and memory management, in order to increase predictability. In contrast to the approach of providing guarantees to applications, the work described in [Hanko91b] is based on the premise that time constraint violation will remain a common occurrence in a workstation operating system. In these situations it is suggested that the system should be designed to either signal the affected applications or follow guidelines which they define in advance. Thus, a policy of graceful degradation is clearly encouraged.

6.3 Summary

This chapter described recent research for synchronisation. Initially, a general overview placed the work of this dissertation in context. This was followed by a detailed examination of five closely related projects, allowing a comparative evaluation of the proposed design. Finally, a commentary on recent related work for operating system scheduling was provided.

Chapter 7

Conclusion

This dissertation has addressed the issue of synchronisation for integrated continuous media. A detailed analysis of the problems and requirements has yielded a set of recommendations which have been adopted in the proposed design.

The design is orientated towards application support in a distributed system where it is desired to process continuous media in an integrated fashion. In such an environment, applications have heterogeneous requirements which are not obvious, making it important not to impose preconditions which require functionality to be traded for operational simplicity. This view is in contrast with the majority of previous work, which has typically provided basic synchronisation features in the context of an embedded system which was assumed to be well behaved. In the present design, application requirements for synchronisation are allowed to be much broader and demand that attention is paid to the general management of system resources. A conclusion is that this level of flexibility is conveniently provided in the form of a service which performs synchronisation on behalf of applications. Thus, an application determines it requirements and negotiates an appropriate synchronisation QOS with the service. The ability of the service to operate as requested is influenced by network QOS and relies upon support from the operating system, notably in regard to CPU scheduling.

A prototype implementation of the design exists as a multi-threaded server running in user space. It operates on live and stored Pandora streams, which were shown to require synchronisation after traversing an ATM inter-network. The prototype has allowed the feasibility of providing synchronisation as a service to be demonstrated in practice. The influence of thread scheduling and kernel timer support on server performance have been determined experimentally. In addition, the ability of the server to perform synchronisation between streams has been clearly displayed. This set of experiments has been used, in combination with a detailed criticism of closely related work, to evaluate the proposed design.

The objective of the research has been satisfied, in that the role and position of synchronisation within the system architecture have been understood and explained. In addition, conclusions have been drawn regarding the influence of kernel support on the ability to perform synchronisation. Service level support for handling continuous media will ultimately be determined by advances in operating system design. In particular, future research is needed to allow temporal requirements to be given sufficient respect when scheduling communications and processing in an integrated continuous media system.

Bibliography

[Abate89] J. E. Abate et al. *AT&T's New Approach to the Synchronisation of Telecommunication Networks.* IEEE Communications Magazine, pages 35–45, April 1989. (p 13)

[Ades86] S. Ades et al. *Protocols for Real Time Voice Communication on a Packet Local Network.* In Proceedings of the IEEE International Conference on Communications (ICC '86), Toronto, pages 525–530, June 1986. (pp 8, 12)

[Ades87] S. Ades. *An Architecture for Integrated Services on the Local Area Network.* Ph.D. thesis, Computer Laboratory, University of Cambridge, January 1987. Published as Technical Report No. 114. (pp 11, 17, 36, 37)

[Almeida91] N. Almeida et al. *End-to-End Synchronisation in Packet Switched Networks.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (pp 17, 18, 27)

[Anderson86] D. P. Anderson and R. Kuivila. *Accurately Timed Generation of Discrete Musical Events.* Computer Music Journal, 10(3):48–56, 1986. (p 20)

[Anderson89a] D. P. Anderson et al. *Integrated Digital Continuous Media: A Framework based on MACH, X11, and TCP/IP.* Technical Report UCB/CSD 90/566, Computer Science Division, University of California, Berkeley, March 1989. (p 103)

[Anderson89b] D. P. Anderson et al. *Support for Continuous Media in the DASH System.* Technical Report UCB/CSD 89/537, Computer Science Division, University of California, Berkeley, October 1989. (pp 22, 110)

[Anderson90] D. P. Anderson and R. Kuivila. *A System for Computer Music Performance.* ACM Transactions on Computer Systems, 8(1):56–82, February 1990. (p 20)

[Anderson91a] D. P. Anderson and G. Homsy. *A Continuous Media I/O Server and its Synchronisation Mechanism.* IEEE Computer, pages 51–57, October 1991. (pp 55, 102)

[Anderson91b] D. P. Anderson and G. Homsy. *Synchronisation Policies and Mechanisms in a Continuous Media I/O Server.* Technical Report UCB/CSD 91/617, Computer Science Division, University of California, Berkeley, February 1991. (p 102)

[ANSA89] *The ANSA Testbench Reference Manual.* Poseidon House, Castle Park, Cambridge, CB3 0RD, March 1989. (pp 71, 100)

[Arons89] B. Arons. *The VOX Audio Server.* In Proceedings of the 2nd IEEE Comsoc International Multimedia Communications Workshop (Multimedia '89), Ottawa, April 1989. (p 40)

[Audsley90] N. Audsley and A. Burns. *Real-Time System Scheduling.* Technical Report YCS 134(1990), University of York, 1990. (p 110)

[Berrocal92] J. Berrocal et al. *Assessment of the Synchronisation Functions.* Technical Report OSI95/LANC/C2/TR/P/V1, Computing Department, Lancaster University, January 1992. Published within ESPRIT Project OSI 95 on High Performance OSI Protocols with Multimedia Support on HSLANs and B-ISDN. (pp 46, 106)

[Blair90] G. S. Blair et al. *Distributed Systems Support for Heterogeneous Multimedia Environments.* Position Paper, Computing Department, Lancaster University, November 1990. (p 16)

[Blair92] G. S. Blair et al. *The Role of Operating Systems in Object-Oriented Distributed Multimedia Platforms.* Technical Report, Computing Department, Lancaster University, May 1992. (p 22)

[Blakowski91] G. Blakowski et al. *Tools for Specifying and Executing Synchronised Multimedia Presentations.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 103)

[Bulterman91] D. C. A. Bulterman and R. van Liere. *Multimedia Synchronisation and UNIX*. In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (pp 28, 110)

[Campbell92a] A. Campbell et al. *A Continuous Media Transport and Orchestration Service*. In Proceedings of ACM SIGCOMM '92, Maryland, pages 99–110, August 1992. (p 106)

[Campbell92b] A. Campbell et al. *Distributed Multimedia Communication System Requirements*. Technical Report OSI95/ELIN-1/C/V1 - Draft Version, Alcatel, January 1992. Published within ESPRIT Project OSI 95 on High Performance OSI Protocols with Multimedia Support on HSLANs and B-ISDN. (pp 8, 11)

[CCETT88] CCETT (Centre Commun d'Études de Télédiffusion et Télécommunications), France. *Rapport final du groupe A.F.N.O.R. Z60R-76 sur la normalisation de la représentation et des protocoles pour l'éxchange des applications audiovisuelles interactives (AVI)*, 1988. (p 34)

[CCITT88] CCITT Study Group XVIII Draft Recommendation I.121. *Broadband Aspects of ISDN*, February 1988. (pp 2, 35)

[Chen89] T. M. Chen et al. *Dynamic Priority Protocols for Packet Voice*. IEEE Journal on Selected Areas in Communications, 7(5):632–643, June 1989. (p 13)

[Cochennec85] J. Y. Cochennec et al. *Asynchronous Time-Division Networks: Terminal Synchronisation for Video and Sound Signals*. In Proceedings of IEEE GlobeCom '85, pages 791–794, December 1985. (p 17)

[Coulson91] G. Coulson et al. *Protocol Support for Distributed Multimedia Applications*. In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 105)

[Crosby92] S. A. Crosby. *In Call Renegotiation of Traffic Parameters*. Internal Report, Computer Laboratory, University of Cambridge, 1992. Submitted to IEEE INFOCOM '93. (p 46)

[Damm89] A. Damm et al. *The Real-Time Operating System of MARS.* ACM Operating Systems Review, 23(3):141–157, July 1989. (pp 14, 41)

[De Prycker87] M. De Prycker et al. *Terminal Synchronisation in Asynchronous Networks.* In Proceedings of the International Conference on Communication (ICC '87), pages 800–807, June 1987. (p 17)

[Dixon91] M. J. Dixon. *System Support for Multi-Service Traffic.* Ph.D. thesis, Computer Laboratory, University of Cambridge, September 1991. Published as Technical Report No. 245. (pp 11, 35, 71, 77, 85)

[Ferrari90a] D. Ferrari. *Client Requirements for Real-Time Communication Services.* Internet RFC-1193, November 1990. (p 8)

[Ferrari90b] 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. (p 24)

[Ferrari91] D. Ferrari. *Design and Applications of a Delay Jitter Control Scheme for Packet-Switching Internetworks.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 32)

[Fujikawa91] K. Fujikawa et al. *Multimedia Presentation System Harmony with Temporal and Active Media.* In Proceedings of the USENIX Summer '91 Conference, Nashville, pages 75–93, June 1991. (p 4)

[Gifford88] D. K. Gifford and N. Glasser. *Remote Pipes and Procedures for Efficient Distributed Communication.* ACM Transactions on Computer Systems, 6(3):258–283, August 1988. (p 64)

[Glauert91] T. Glauert and J. Gettys. *X Synchronisation Extension.* Overview of version 2.0 from the X Window System (version 11, release 5) documentation, 1991. (p 108)

[Govindan91] R. Govindan and D. P. Anderson. *Scheduling and IPC Mechanisms for Continuous Media.* Technical Report UCB/CSD 91/622, Computer Science Division, University of California, Berkeley, March 1991. Also appears in the Proceedings of the

13th ACM Symposium on Operating Systems Principles, 1991. (pp 27, 103)

[Greaves90] D. Greaves et al. *The Cambridge Backbone Ring.* In Proceedings of IEEE INFOCOM '90, San Francisco, 1990. (p 71)

[Gusella83] R. Gusella and S. Zatti. *TEMPO - A Network Time Controller for a Distributed Berkeley UNIX System.* Technical Report UCB/CSD 83/163, Computer Science Division, University of California, Berkeley, December 1983. (pp 13, 52)

[Hanko91a] J. G. Hanko et al. *Integrated Multimedia at Sun Microsystems.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 22)

[Hanko91b] J. G. Hanko et al. *Workstation Support for Time-Critical Applications.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 110)

[Hayter91] M. Hayter and D. McAuley. *The Desk Area Network.* Technical Report 228, Computer Laboratory, University of Cambridge, May 1991. Published in ACM Operating Systems Review, 25(4), October 1991. (pp 30, 36)

[Hazard91] L. Hazard et al. *Notes on Architectural Support for Distributed Multimedia Applications.* Technical Report ESPRIT ISA Project Number 2267, CNET (Centre National d'Études des Télécommunications), Issy les Moulineaux, France, March 1991. (p 99)

[Hodges89] M. E. Hodges et al. *A Construction Set for Multimedia Applications.* IEEE Software, pages 37–43, January 1989. (p 4)

[Hopper88] A. Hopper and R. M. Needham. *The Cambridge Fast Ring Networking System.* IEEE Transactions on Computers, 37(10):1214–1223, October 1988. (p 3)

[Hopper90] A. Hopper. *Pandora - An Experimental System for Multimedia Applications.* ACM Operating Systems Review, 24(2):19–34, April 1990. (pp 3, 11, 39, 40, 71)

[Hopper92] A. Hopper. *Improving Communications at the Desktop.* Technical Report 92-3, Olivetti Research Ltd., Cambridge, March 1992. (p 3)

[Jardetzky92] P. W. Jardetzky. *Network File Server Design for Continuous Media.* Ph.D. thesis, Computer Laboratory, University of Cambridge, August 1992. (pp 39, 40, 77)

[Jeffay91] K. Jeffay et al. *Kernel Support for Live Digital Audio and Video.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (pp 11, 45, 110)

[Khanna92] S. Khanna et al. *Realtime Scheduling in SunOS 5.0.* In Proceedings of the USENIX Winter '92 Conference, pages 375–390, 1992. (p 16)

[King92] T. King. *Pandora: An Experiment in Distributed Multimedia.* Technical Report, Olivetti Research Ltd., Cambridge, May 1992. Presented at EuroGraphics '92, Cambridge. (pp 3, 12)

[Kopetz87] H. Kopetz and W. Ochsenreiter. *Clock Synchronisation in Distributed Real-Time Systems.* IEEE Transactions on Computers, 36(8):933–940, August 1987. (pp 14, 52)

[Lamport78] L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, 21(7):558–565, July 1978. (p 52)

[Le Gall91] D. Le Gall. *MPEG: A Video Compression Standard for Multimedia Applications.* Communications of the ACM, April 1991. (p 11)

[Leslie84] I. M. Leslie et al. *The Architecture of the Universe Network.* In Proceedings of ACM SIGCOMM '84, Montreal, June 1984. (p 12)

[Leung90] W. F. Leung et al. *A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks.* IEEE Journal on Selected Areas in Communications, 8(3):380–390, April 1990. (p 35)

[Levi90] S. Levi and A. K. Agrawala. *Real-Time System Design.* McGraw-Hill, New York, 1990. (p 13)

[Leydekkers91] P. Leydekkers and B. Teunissen. *Synchronisation of Multimedia Data Streams in Open Distributed Environments.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (pp 8, 34)

[Li91] G. Li. *Towards a Real-Time ANSA*. Internal report, Computer Laboratory, University of Cambridge, December 1991. (p 41)

[Little90a] T. D. C. Little and A. Ghafoor. *Network Considerations for Distributed Multimedia Object Composition and Communication*. IEEE Network Magazine, pages 32–49, November 1990. (p 101)

[Little90b] T. D. C. Little and A. Ghafoor. *Synchronisation and Storage Models for Multimedia Objects*. IEEE Journal on Selected Areas in Communications, 8(3):413–427, April 1990. (p 101)

[Little91] T. D. C. Little and A. Ghafoor. *Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks*. IEEE Computer, pages 42–50, October 1991. (p 101)

[Luther89] A. C. Luther. *Digital Video in the PC Environment*. McGraw-Hill, New York, 1989. (pp 30, 38, 62)

[McAuley89] D. R. McAuley. *Protocol Design for High Speed Networks*. Ph.D. thesis, Computer Laboratory, University of Cambridge, September 1989. Published as Technical Report No. 186. (pp 15, 28, 71, 85)

[McJones87] P. R. McJones and G. F. Swart. *Evolving the UNIX System Interface to Support Multithreaded Programs*. Technical Report 21, DEC Systems Research Centre, Palo Alto, September 1987. (p 22)

[Meissner91] K. Meissner. *Architectural Aspects of Multimedia CD-I Integration in UNIX/X-Windows Workstations*. In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 68)

[Mills83] D. L. Mills. *Internet Delay Experiments*. Internet RFC-889, December 1983. (p 14)

[Mills89] D. L. Mills. *Internet Time Synchronisation: the Network Time Protocol*. Internet RFC-1129, October 1989. (pp 13, 52)

[Montgomery83] W. A. Montgomery. *Techniques for Packet Voice Synchronisation*. IEEE Journal on Selected Areas in Communications, 1(6):1022–1028, December 1983. (p 14)

[Nakajima91] J. Nakajima et al. *Multimedia/Realtime Extensions for the Mach Operating System.* In Proceedings of the USENIX Summer '91 Conference, Nashville, pages 183–197, June 1991. (p 110)

[Nicolaou90a] C. A. Nicolaou. *An Architecture for Real-Time Multimedia Communication Systems.* IEEE Journal on Selected Areas in Communications, 8(3), April 1990. (p 18)

[Nicolaou90b] C. A. Nicolaou. *A Distributed Architecture for Multimedia Communication Systems.* Ph.D. thesis, Computer Laboratory, University of Cambridge, December 1990. Published as Technical Report No. 220. (pp 11, 35, 67, 101)

[Partridge91] C. Partridge. *Isochronous Applications Do Not Require Jitter-Controlled Networks.* Internet RFC-1257, September 1991. (p 31)

[Postel80] J. B. Postel. *A Structured Format for Transmission of Multi-Media Documents.* Internet RFC-767, August 1980. (p 107)

[Puckette86] M. Puckette. *Interprocess Communication and Timing in Real-Time Computer Music Performance.* In Proceedings of the International Computer Music Conference, pages 43–46, 1986. (p 41)

[Rangan91] P. V. Rangan et al. *A Testbed for Managing Digital Video and Audio Storage.* Technical Report CS91-193, Department of Computer Science and Engineering, University of California, San Diego, April 1991. (pp 36, 61)

[Ripley89] G. D. Ripley. *DVI - A Digital Multimedia Technology.* Communications of the ACM, 32(7):811–822, July 1989. (p 63)

[Sarin85] S. Sarin and I. Grief. *Computer-Based Real-Time Conferencing Systems.* IEEE Computer, pages 33–45, October 1985. (p 37)

[Scheifler86] R. W. Scheifler and Gettys J. *The X Window System.* ACM Transactions on Graphics, 5(2):79–109, April 1986. (p 3)

[Schmandt88] C. Schmandt and M. A. McKenna. *An Audio and Telephone Server for Multi-Media Workstations.* In Proceedings of the 2nd IEEE Conference on Computer Workstations, Santa Clara, California, 1988. (p 40)

[Schmidt85] B. L. Schmidt and J. M. Roth. *The Synchronisation of Audio Production in Computer Music.* In Proceedings of the International Computer Music Conference, pages 341–345, 1985. (p 55)

[Shepherd89] D. Shepherd and M. Salmony. *Extending OSI to Support Synchronisation Required by Multimedia Applications.* Technical Report 43.8904, IBM European Network Centre, Heidelberg, Germany, April 1989. (p 106)

[Sreenan90] C. J. Sreenan. *Synchronised Retrieval of Multi-Media Data.* In Proceedings of the 1st International Workshop on Network and Operating System Support for Digital Audio and Video, Berkeley, November 1990. Published as Technical Report TR-90-062, International Computer Science Institute (ICSI), Berkeley. (p 39)

[Staehli91] R. Staehli. *The Utility of Prefetching Data in a Distributed Multimedia Storage Server.* Internal report, Oregon Graduate Institute, May 1991. A related position paper appeared in ACM Operating Systems Review, April 1992. (p 68)

[Steinmetz90] R. Steinmetz. *Synchronisation Properties in Multimedia Systems.* IEEE Journal on Selected Areas in Communications, 8(3):401–412, April 1990. (p 100)

[Sventek87] J. S. Sventek. *An Architecture Supporting Multi-Media Integration.* Technical Report AO.33.02, Advanced Networked Systems Architecture (ANSA), January 1987. (pp 11, 34)

[Swinehart83] D. C. Swinehart et al. *Adding Voice to an Office Computer Network.* In Proceedings of IEEE GlobeCom '83, November 1983. Also available as Xerox PARC technical report CSL-83-8, February 1983. (p 7)

[Szabo91] B. I. Szabo and G. K. Wallace. *Design Considerations for JPEG Video and Synchronized Audio in a UNIX Workstation Environment.* In Proceedings of the USENIX Summer '91 Conference, Nashville, pages 353–367, June 1991. (pp 30, 38, 53)

[Tennenhouse89] D. L. Tennenhouse. *Layered Multiplexing Considered Harmful.* In Proceedings of the IFIP Workshop on Protocol Design for High-Speed Networks, IBM Zurich Research Laboratories, pages 1–6, May 1989. (p 15)

[Tokuda90] H. Tokuda et al. *Real-Time Mach: Towards a Predictable Real-Time System.* In Proceedings of USENIX Mach Workshop, Burlington, Vermont, pages 73–82, October 1990. (p 110)

[Verbiest86] W. Verbiest. *Video Coding in an ATD Environment.* In Proceedings of the 3rd International Conference on New System Services for Telecommunications, Liege, Belgium, pages 249–253, November 1986. (p 18)

[Verbiest88] W. Verbiest et al. *The Impact of the ATM Concept on Video Coding.* IEEE Journal on Selected Areas in Communications, 6(9):1623–1632, December 1988. (p 26)

[Verma90] D. C. Verma et al. *Guaranteeing Delay Jitter Bounds in Packet-Switching Networks.* In Proceedings of the 1st International Workshop on Network and Operating System Support for Digital Audio and Video, Berkeley, November 1990. Published as Technical Report TR-90-062, International Computer Science Institute (ICSI), Berkeley. (p 45)

[Wayner91] P. Wayner. *Inside QuickTime.* BYTE, pages 189–196, December 1991. (p 4)

[Yu89] C. Yu et al. *Efficient Placement of Audio Data on Optical Disks for Real-Time Applications.* Communications of the ACM, 32(7), July 1989. (p 38)

[Zellweger89] P. T. Zellweger. *Scripted Documents: A Hypermedia Path Mechanism.* In Proceedings of the ACM Hypertext'89 Conference, pages 1–14, November 1989. Also available as Xerox PARC technical report EDL-89-3, December 1989. (p 4)