Technical Report

UCAM-CL-TR-114
ISSN 1476-2986

Number 114

An architecture for integrated services on the local area network

Stephen Ades

September 1987

This technical report is based on a dissertation submitted January 1987 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Trinity College.

ISSN 1476-2986

Summary

This dissertation concerns the provision of Integrated Services in a local area context, e.g. on business premises. The term Integrated Services can be understood at several levels. At the lowest, one network may be used to carry traffic of several media—voice, data, images etc. Above that, the telephone exchange may be replaced by a more versatile switching system, incorporating facilities such as stored voice messages. Its facilities may be accessible to the user through the interface of a workstation rather than a telephone. At a higher level still, new services such as multi-media document manipulation may be added to the capabilities of a workstation.

Most of the work to date has been at the lowest of these three levels, under the auspices of the Integrated Services Digital Network (ISDN), which mainly concerns wide area communications systems. The thesis presented here is that all of the above levels are important in a local area context. In an office environment, sophisticated data processing facilities in a workstation can usefully be combined with highly available telecommunications facilities such as the telephone, to offer to the user new services which make the working day more pleasant and productive. That these facilities should be provided across one integrated network, rather than by several parallel single medium networks is an important organisational convenience to the system builder.

The work described in this dissertation is relevant principally in a local area context—in the wide area economics and traffic balance dictate that the emphasis will be on only the network level of integration for some time from now. The work can be split into three parts:

i) the use of a packet network to carry mixed media. This has entailed design of packet voice protocols which produce delays low enough for the network to interwork with national telephone networks. The system has also been designed for minimal cost per telephone—packet-switched telephone systems have traditionally been more expensive than circuit-switched types. The network used as a foundation for this work has been the Cambridge Fast Ring.

ii) use of techniques well established in distributed computing systems to build an 'Integrated Services PABX (Private Automatic Branch Exchange)'. Current PABX designs have a very short life expectancy and an alarmingly high proportion of their costs is due to software. The ideas presented here can help with both of these problems, produce an extensible system and provide a basis for new multi-media services.

iii) development of new user level Integrated Services. Work has been done in three areas. The first is multi-media documents. A voice editing interface is described along with the system structure required to support it. Secondly a workstation display has been built to support a variety of services based upon image manipulation and transmission. Finally techniques have been demonstrated by which a better interface to telephony functions can be provided to the user, using methods of control typical of workstation interfaces.

Preface

I should like to thank my supervisor, Professor Roger Needham, for his wise and enthusiastic counsel throughout this research. I should also like to thank Dr Ian Leslie, Dr Alan Jones, Dr Christopher Adams, David Tennenhouse, Roy Want and Roger Calnan for many stimulating discussions. Whilst a research student I visited Xerox Palo Alto Research Center. I am grateful to Dr Daniel Swinehart and to Professor Needham for making this possible and to Dr Swinehart, Dr Polle Zellweger and Dr Douglas Terry for a very stimulating period of work there.

Special thanks are due to all those who read and commented on drafts of this dissertation—as well as some of those above, to Professor David Wheeler, Dr Christopher Cooper and Olivia Nagioff. I owe much to Olivia Nagioff for support and encouragement throughout the production of the dissertation.

I am grateful to the following bodies who provided financial support for this work. The Science and Engineering Research Council provided money for subsistence and travel. British Telecom provided money for equipment, subsistence and travel. Trinity College provided money for travel.

Except where specifically stated otherwise, this dissertation is the result of my own work and is not the outcome of any work done in collaboration. Furthermore, this dissertation is not substantially the same as any that I have submitted for a degree, diploma or other qualification at any other University. No part of this dissertation has already been or is being concurrently submitted for any degree, diploma or other qualification.

Several parts of this dissertation have been published as papers. An appendix to the dissertation lists all such papers.

Chapters

Contents

List of Figures

1. Introduction

1.1 Background

There is currently much interest in Integrated Services systems—systems handling within one network or architecture a mixture of voice, data and (eventually) video. The term Integrated Services can be and is interpreted widely and at various levels. At the lowest level (transmission) a network may carry traffic of a number of different kinds. This traffic may be combined only in order to carry it across the network and may end up at a series of customers' terminals, each handling only one medium. Alternatively all the traffic may be delivered to a single terminal. The network may or may not offer guarantees of synchronisation between the different media carried. Integrated Services may also be offered to users at a much higher level. As an example, facilities might be offered for several remote users collectively to edit a document which is displayed in front of them all, whilst communicating over voice and/or slow scan image links between them. Such a collection of communications channels might be carried on the same network or on very different networks—service integration at one level does not necessarily imply integration at another one.

Most of the work on Integrated Services to date has been at the transmission level, under the title of the Integrated Services Digital Network, or ISDN. In essence, the ISDN is a series of protocols agreed by the PTTs (national [postal,] telephone and telegraph operators) of the world, by which terminal equipment carrying some communications medium (such as telephony, data file transfers or facsimile) can be connected to a public network and communicate with equipment handling the same medium at another (remote) location served by a public network. Most of the work on the ISDN has been geared towards the use of the PTTs' networks (which are largely the existing telephone networks, plus a little extra infrastructure to handle particular non-telephony loads) for a combination of different media. The ISDN is mainly seen as a wide area network, but some ISDN work deals with more local facilities, based around the private automatic branch exchange (PABX) sited on a customer's premises. There has been a gradual increase in the range of facilities offered by the typical PABX, such as personal short dialling codes, call forwarding etc. The ISDN work on PABXs has largely been on the use of conventional star-configured PABXs as a means of connecting for example terminals and computers within a customer's premises [Swart 86].

Whilst the PTTs have been using some of their telephone network capacity to carry data traffic, some experiments have also been done on the use of computer networks to carry a little voice traffic: the earliest and most well known work was that carried out as part of the ARPAnet experiments [Weinstein 83]. Like the ISDN, this work was an attempt to fit one medium onto a network designed for another: the protocols chosen indicate that the second medium had to be 'shoe-horned' somewhat to fit it into the network.

All of the above work is concerned with networks. Recent years have also seen the growth of the personal computer and the 'workstation'. In the office environment such an object plays an increasing role in most of the aspects of the user's working day. Depending on work requirements, the workstation might typically provide its user with facilities for text composition, document preparation and typesetting; a programming environment and computing facilities; electronic mail; spreadsheets and accounting facilities; customer lists and inventory control; CAD for various engineeering applications etc. At least at present, emphasis is placed upon the workstation as a processing engine rather than as a communication tool, although electronic mail clearly falls into the latter category. Workstations are generally linked together by computing local area networks (LANs). This was done originally to enable such expensive resources as disc file servers and printers to be shared among several workstation users. Increasingly it has been found that centralised filing systems and LANs allow sharing and exchanging documents and other data: this is an important communications function but is normally limited to local sharing of information.

1.2 Aims of this Research

Against the above background, this dissertation describes a new architecture for Integrated Services provision, which is innovatory at three levels:

i) the use of a single network to carry Integrated Services. Unlike the examples above, we will describe a network which is well suited to the simultaneous transmission of different media, rather than being designed for a single medium with other types of traffic then 'grafted on'. The network is a packet network: design of packet protocols which satisfy the normal criteria set by PTTs for voice transmission is an innovatory subject.

ii) the provision of functionality equivalent to an Integrated Services PABX, but produced using a style of organisation more typical of distributed computing systems. Research in distributed computing [e.g. Needham 82] has shown that a series of small machines linked on a LAN, each implementing some simple function (the 'distributed server concept'), can be used to provide the capabilities equivalent to a large operating system running on for example a large mainframe computer but with greater elegance and more cost-effectively. The same architecture is proposed as an alternative to the centralised star-configured PABX: software is reported to account for 80% of PABX costs [BT 86], so that an architecture which enables the components to be produced more simply seems promising, particularly as the PABX functions are expanded to include for example store-and-forward voice messages and similar capabilities.

iii) the provision of high (i.e. user) level Integrated Services. Many engineers concerned with the ISDN are beginning to realise its major limitation to be an almost exclusive concern with the network level: traffic emerges from the ISDN via a series of single medium terminals. A large potential benefit from integration for the business customer would come from the ability to manipulate all kinds of data together at one 'Integrated Services workstation': this would make possible facilities such as integrated document architectures containing voice (e.g. for easy annotation) and control of the telephone from a workstation rather than by using obscure sequences of numbers typed on a 12 key telephone pad. All of this can make a user's working environment more pleasant and productive. Many new facilities can be provided once it is possible to transport voice, images and data files between users in a coherent and integrated manner.

1.3 Scope of this Work

This thesis is concerned with realisation of the above aims mainly within the local area context. Limiting the scope to either wide or local area design is beneficial in that the design tradeoffs, economics and traffic balance will be very different for each case. Within the local area context, a study made on an integrated office containing both PABX facilities and distributed computing resources [Sincoskie 83] found there to be more data traffic than voice traffic. In the wide area domain most traffic, by some way, is still voice and a large proportion of the rest is electronic mail.[1] The traditional methods of wide area networking, employing circuit-switched connections, will remain the economic solution for a long time. On the other hand, in a local area context different types of switching are more favourable. A PABX employing packet switching is now on field trial by one major manufacturer [Burst 86]. The realisation that LAN bandwidth is very cheap and that high bandwidth networks are not necessarily much more expensive than lower bandwidth systems, together with the increased cost of software production, makes new architectures for provision of Integrated Services facilities attractive as the basis of telecommunications products for local area use in the near future.

Several research efforts are attempting to address only a part of the three levels of work set out as aims above. Since each area is a large topic in itself, a reduced frame of reference may seem sensible. However we feel that consideration of all three areas together has benefited our design at all levels. For example, as will be shown in chapters below, the way in which facilities are to be provided to the user affects the organisation and partitioning of the servers which form the infrastructure of our system. The functions of these servers, and in particular the mixes of traffic which they will be called upon to support has affected the design of network protocols and of network interface hardware to support those protocols.

1.4 Basic Design Considerations

1.4.1 The Network

PTTs in general use circuit-switched channels to carry mixed media traffic. Circuit switching is well suited to voice traffic, which requires a fixed amount of bandwidth to be available throughout a telephone call. The tight delay requirements of telephone connections imply that this bandwidth needs to be available almost immediately upon demand. The bandwidth need not actually be available at *all* times throughout a conversation: in a fully duplex telephone conversation, only 35-40% of the bandwidth in each direction will be used [Bullington 59, Brady 68].[2] Each participant will be silent for 60-65% of the time. However for a single connection, the timing of the 'talkspurts' in the conversation is unpredictable and full bandwidth must be available almost immediately a talkspurt begins. Therefore the unused bandwidth cannot be allocated for anything that is not immediately pre-emptible. On a trunk route carrying large numbers of speech connections, the designer can rely on statistical effects and this so-called TASI

[1] This is only considering voice and data transmission. If the bandwidth used for all electronic communications is considered the majority is television signals. This remark is pertinent in that in Japan plans have been announced to carry voice and data on cable television networks.

[2] These measurements were made on normal phone conversations. An interesting figure from another study is to be found in [Gan 86], which demonstrates that use of an efficient silence detector can reduce the bandwidth required for a (simplex) prepared reading, such as a news broadcast, to 65%.

advantage [Bullington 59] can be used to share a reduced amount of bandwidth between all the conversations.[3]

Circuit switching has traditionally been an efficient way to handle voice since between the times of starting and finishing a call a constant amount of bandwidth will be required. Where only a few simultaneous calls are to be handled on one link, committing one channel to handle each voice stream exclusively at just under half utilisation of the channel has traditionally been better than providing the intelligence to switch every single packet of voice along uncommitted channels.

The same is not true for data traffic. Typical examples of data traffic are a keystroke or a screen update sent between computer and remote terminal, or a file transfer between two filing systems. In each case the traffic is bursty, so that fixed channel allocation will be very wasteful. Unlike voice traffic, where some modest level of errors is of no consequence (since it will not be noticed by the listener), data transmission must be error-free, implying a requirement for protocol layers quite different from anything needed for voice. Whilst voice requires very small and strict transmission delays, the exact timing of a terminal screen refresh is of little importance—in particular the information is perfectly comprehensible whenever it arrives. The user waiting for a file transfer or screen update clearly desires as much bandwidth as possible during the burst of transmission activity, whereas a phone connection never requires more than a fixed level.

Thus packet switching is more attractive for data, since in a packet network it is feasible to have many connections set up simultaneously but idle for much of the time. The available bandwidth will be divided at any instant between those connections actually engaged in transmission bursts. The network may occasionally become overloaded and have to discard traffic: data transmission protocols are in general designed to cope with loss of packets. In general a data network will happily allow large numbers of connections simultaneously—it is understood that from time to time the network will become heavily loaded and each transmitter will obtain very little bandwidth. This approach would be quite unacceptable for voice: if "all lines to London are engaged" we accept the need to "please try again later", but once a call is set up it must be allowed to proceed unhindered until *the caller* terminates it, which means that full bandwidth must be available on demand. This mode of working is clearly natural to circuit-switched systems.

Given the very different requirements of voice and data, how can a single network attempt to carry them both satisfactorily? The PTTs have tended to force data into the circuit switched mode because it is the minority traffic by a significant margin. The ARPAnet packet voice work effectively did the same but in reverse. Recently, several 'hybrid' systems have been produced. The term 'hybrid' was first of all used to refer to networks which provided analogue multiplexing of voice and data. Examples include the IBM Hybrid Switch [Rudin 78] and the AT&T Datakit™ [Fry 86]. The more recent hybrid networks of interest here, however, are packet-based structures. At the hardware level some proportion of the bandwidth can be reserved for voice traffic with the rest allocated to more bursty loads. This clearly provides a good basis for transmission of

[3] TASI-based designs typically specify a small percentage blocking probability at the start of a talkspurt. Blocking can be regarded as a type of burst error, but note that samples discarded at the start of a word will generally be more noticeable than burst errors in the middle.

integrated media. Generally the bandwidth allocation is dynamic, i.e. the voice traffic is guaranteed some maximum proportion of the total bandwidth with some guarantee also about delay, but any bandwidth not actually used by the voice traffic at an instant will be offered to other traffic.[4]

Most of the hybrid networks of interest for this work are broadcast bus or ring systems. Traditional PABX systems, in contrast, have a star configuration. It should be noted that whereas star systems are based around central switching modules, bus or ring LANs are in themselves distributed switches, with no centralised handling of traffic. We will only be concerned with networks offering distributed switching of traffic.

1.4.2 Partitioning the System

Mention was made above of the 'distributed server concept', which has been shown to ease the task of producing a large system [Needham 82]. An operating system, for example, may be split into fairly simple modules and each of these implemented as a 'server', i.e. a small computing engine linked to all other servers by a local area network. In building a PABX, such modularity is very attractive. Individual facilities (such as conference facilities) can be implemented as single function servers. A PABX can be offered to the customer with a variety of options: each requested option or small group of options may entail installing a server. If the servers are partitioned well, each will have a very straight-forward and self-contained function and therefore will be fairly easy to produce. If the software is concentrated on achieving one function only, it can do this efficiently, whereas a large system running many tasks may not execute any of them in an optimum manner. This is especially important in the case of real time telephony functions.

If the customer is to be allowed options as to which features are provided in any single installation, this implies that there will need to be standardisation of interface between various servers, so that when not all services are installed the system can run in several different configurations with different interconnections between the servers. Standardised interfaces also have benefits in terms of upgrades and new facilities. Modular construction and uniform interfaces together imply ease of making enhancements, which can be effected either by altering the behaviour of existing servers (without the need to change all other servers in the system) or by introducing new servers which conform to a single standard interface. The inability to add new facilities is a major problem of conventional PABXs. The average installed life expectancy of a PABX has fallen (from 20 years in the case of simple Strowger PABXs) to less than 5 years [BT 86]. These PABXs become obsolete not because of insufficient capacity but because customers demand new facilities.

If the configuration of components in a system can be altered, there needs to be some form of control system which knows what facilities are available in any one installation and how to connect them together. This control system may itself simply be another server. If we consider as a basis for this work only LANs which provide implicit distributed switching, the traffic handling requirement upon the control server will be very modest, compared with that on the central

[4] Dynamic division of bandwidth is most useful in networks where the sizes of data and voice packets are very different [Hagirahim 86].

switch of a central star-configured PABX. When for example two telephones have been set up to talk to one another, they will not need any further explicit switching. For a simple call, no further attention will be required, other than possibly shaking a "dead man's handle", until the connection needs to be closed down. The control system will only be concerned with altering connections, not with maintaining them.

Overall this approach to partitioning can be economically very attractive. In a PABX system the phones need to be very cheap, there being so many of them. Upgrades to the phones should not be required in order to support new facilities. Both of these can be ensured by partitioning all functionality out of the phones and into simple servers. Partitioning also facilitates making a reliable system. Both phones and servers, running a simple set of functions, are much easier to make reliable in terms of software. The hardware of the phone can also be made very simple and hence reliable. For servers there will need to be some form of monitoring for failures. This needs to be done in close cooperation with the control system, with considerable data sharing between the two functions. Both failure checker and control server probably need to be replicated for reliability. Failure checking might well be done actually as a part of the control server's functions. If this is so then one can envisage some number of control systems, all running concurrently and checking both the rest of the system and each other.

1.4.3 Network Utilisation

This is an issue which shows up a difference in traditional philosophy between those involved with distributed computing and telecommunications systems. The bandwidth utilisation in a computer network will often be very low. In contrast, under-utilised bandwidth has traditionally been regarded as a heresy by communications engineers—bandwidth is expensive and a trunk route which can be made to carry more telephone calls will earn more revenue.

However at least for local computer networks bandwidth is not an expensive commodity. In particular increasing the bandwidth of a LAN may, within limits, not entail much expense. Research into distributed computing has therefore generally been aimed at an organisation of servers which makes complex functionality easy to produce. If it is easier to build a network with ten times as much bandwidth as is required than to build an intelligent network interface capable of utilising all of a lower network bandwidth then this may be a sensible engineering solution. If it is easier to build a voice handling system out of three servers than out of one, it may not matter that the same voice traffic makes three journeys across the network rather of one. By running a simple packet network at high speed we may be able to ensure adequate bandwidth and delay guarantees for voice traffic, whereas at a lower speed we might have to resort to a more complex and expensive hybrid switching network. These three examples are of course ones which will be developed later in this dissertation.

This type of approach has been used in local area computing systems for some time now. It is gradually becoming more appropriate in the telecommunications domain. For example we are approaching the point where the electronic modules (transmitters, repeaters, regenerators) on the ends of an optical fibre link are more expensive than the fibre itself. It may therefore be attractive to use cheaper modules which employ redundant coding, thus wasting bandwidth but producing a more economic system overall.

1.4.4 Support for User Level Facilities

The natures of the high level facilities that we want to present for example to the user of an 'Integrated Services workstation' may well affect the partitioning of functionality amongst our servers. For example this dissertation will discuss support for integrated document architectures. Whereas some researchers have built separate voice and data filing systems for this, we will propose a design of a single filing system together with 'front end' servers tailored to the different media supported. The motivation for this is to simplify the software that marshalls the components of an integrated document together. High level applications software is complex and expensive to produce: a server architecture which makes the task easier is therefore an important part of our design objectives.

We have mentioned the idea of controlling complex telephony functions from a workstation rather than from a simple telephone keypad: a much better user interface can be produced in this way. One might seek to provide such an interface by incorporating the phone physically into the workstation. The problem here is that expectations for availability of the phone and workstation are very different. (Users expect to be able to program their workstations and therefore to be able to crash them. The phone is not expected to become unavailable as a consequence.) Therefore we must consider techniques for controlling the phone from a (physically separate) workstation, by use of suitable control paths, such that failures of the workstation do not affect the availability of simple telephony functions.

1.4.5 The Integrated Workstation

In the Cambridge Distributed Computing System, use is made of a bank of processing machines. These machines are not assigned permanently to single users: users have only a low cost terminal on their desks. Under control of a bank manager, sessions may be set up in which a terminal is connected to a bank machine. During a session the machine serves as the user's personal computer. At the end of the session it returns to the free pool. The advantage of this scheme is highlighted by the observation that the average business personal computer costs around three thousand pounds and is used for less than 30 minutes a day [Shiu 84]. It is clear that placing only the low cost terminal on a user's desk makes sense.

Workstations with bit-mapped displays have not been built along the lines of the processor bank scheme in the past, although some researchers have wished it so. The bandwidth required to drive a bit-mapped display has made such an approach impractical. Use of a bit-mapped terminal is essential for the work described in this dissertation, in particular to allow integrated displays of images, text and graphics. An equivalent of the bank for such displays might be the combination of a low cost workstation/display and more high-powered bank machine. I/O-intensive tasks, such as editing, would be performed in the display and processing tasks, such as compilation, in the bank. However editors are becoming increasingly complex modules: if a multi-media editor is one of our design goals this will become even more true. This implies that the desktop workstation/display may cost more than we would like. Also the basis of successful programming environments is a tight coupling between display/editing facilities and interactive use of key system components such as compilers. If significant use is to be made of bank machines then there will need to be procedure-level interactions between distributed software modules. Of course much work has been done and is being done on this issue. However another scheme which one might pursue is to use the display only for i/o, running the editor on a bank machine. In order to keep the bandwidth requirements between display and bank machine down, communications between bank and display would be at the level of "mouse button *b* pressed at location *l* (or at glyph *g*) in window *w*", "display the character string *s* in font *f* at location *l* in window *w*" or "display the image from file *ERIC* at location *l* in window *w*". These transactions imply very modest bandwidth handling requirements between display and bank. Fonts and images can be transferred directly between fileserver and display. In the case of images at least, the bank machine will manipulate only the structure of a document and not its content.

The display needs to be able to obtain fonts and images at high speed from a fileserver. More generally, it should be able to receive, transmit, code and decode images at high speed, in order also to support services such as telewriting, single image transfers and video conferencing for which this is essential. The display should also be equipped with a camera or other scanning device for image input. From all of this discussion, the Integrated Services workstation display might well principally consist of a framestore with fast network to image plane transfer capability: it might usefully also be designed in such a way as to facilitate the addition of image compression hardware of modest complexity.

At this stage one must ask the question "what emphasis should be placed on video services?". Given a suitable hybrid network (for example offering two prioritised classes of bandwidth allocation, used for voice and video) it is possible to support video traffic using existing LAN technology. The MAGNET network [Lazar 85], whose characteristics are described below, is an example of this. Experience from MAGNET and similar projects suggests that support for high bandwidth video is a large topic in itself.

As will be discussed below, this work will be concerned with networks which are unable to support video traffic at the same level that MAGNET can. However video compression techniques continue to reduce the bandwidth needed for full frame rate transfers dramatically. Also there are very many applications for video services other than full frame rate transfers. A study which explored use of various communication media as an aid to instruction in accomplishing technical tasks [Chapanis 72] concluded that the most useful single communication medium was voice. The ability to exchange diagrams and sketches, as would be provided by a combination of telewriting and single image transfers, was almost as useful as voice communication.[5] The study found that full frame rate video links, even when combined with voice channels, would offer very little additional advantage over voice channels on their own, despite the heavy cost. If the problem of explaining the function of an electronic circuit to a remote enquirer is considered, these findings ring instinctively true.

This dissertation will not consider support for high bandwidth video services, although it will touch on simple approaches to compressing video to a level where it can be sent using a moderate amount of bandwidth. The emphasis will be on an architecture in which a wide variety of user level Integrated Services can be accommodated and controlled rather than on

[5] The study used an image transfer medium which was much slower than telewriting. If the delay element had been removed, it would have been nearly as effective on its own as voice communication. It is unlikely, however, that telewriting would be used without voice.

implementing particular very high performance services. The emphasis of this thesis is intended to be upon organisation and combination of services: we are interested in a single voice/data network as much for its effect on the organisation of servers and services that sit upon it as for the protocol problems at the network level. We are more interested in supporting both telewriting and telephony using one workstation design on this single network than in simple high bandwidth video transfers.

1.5 Other Relevant Research

There have been several designs of Integrated Services LAN. Some of these fall into the hybrid category. Notable examples are BID [Ulug 81], Expressnet [Tobagi 83], Fasnet [Limb 82, Limb 83], MAGNET [Lazar 85], Orwell [Adams 84], Prelude [Thomas 84], QPSX [Newman 86] and SCPS [Takeuchi 86]. These examples represent a spectrum of designs, from objects which look like a traditional PABX to those which look like a computing LAN.

At one end, MAGNET is a slotted ring which supports two classes of packets. An elaborate monitor station sets up a slot structure and regulates the occurrences of each slot type so that delay guarantees can be offered for one of the classes of slot. The structure of Fasnet is a pair of unidirectional broadcast buses. For use as an Integrated Services vehicle it uses fixed sized packets and a pair of end stations, whose behaviour is much like the monitor station in MAGNET. Fasnet is only geared to one voice stream per transmitting station and uses voice packets of 10ms duration—longer if system efficiency is an important criterion. Expressnet and BID are very similar in design to Fasnet—in all three cases the protocol for Integrated Services support is implemented in hardware of considerable complexity. In the Orwell ring there is no monitor or end station to set up different slot types. Each station has a register which defines how much bandwidth it may use in each 'epoch' of the ring. The ring is intended to support large, unevenly distributed telephony traffic loads and each station is of considerable complexity.

At the other end of the spectrum Prelude has a hardware architecture very much like a digital circuit switch, although it is based on packets and supports a very wide range of traffic data rates. SCPS uses a synchronous ring structure linking conventional concentrator switches to handle both circuit switched traffic and HDLC packets with guaranteed fixed delay. It too supports a very wide range of traffic data rates. In between the extremes, QPSX looks in outline like Fasnet, but at a lower level it is a circuit switching/packet switching hybrid, with slot reservation for synchronous traffic and Orwell-like sharing of unreserved slots for data.

It is rare now that any local area network is designed without some claims being made that it is suitable for voice/data integration. Limited work has been done to present the IBM Token Ring [Bux 83] as an integrated services bearer, but it cannot be regarded as a serious contender due to its restricted capabilities, notably that synchronous traffic may only consist of individual fully duplex connections. MSTDM [Maxemchuk 82] is an attempt to modify a CSMA network to offer the time guarantees required by voice, but again it is only useful for a limited number of configurations. Many other instances exist [e.g. Gopalakrishnan 84] of networks only supporting a very specific mix of traffic, such as stations which each allow one telephone channel and one serial data link to be connected.

Similarly there are systems offering user-level integrated facilities which only support specific applications, such as the Centerpoint workstation [Centerpoint 86] which is designed around the executive-and-secretary paradigm, and the MIT 'Phone Slave' project [Schmandt 84] which demonstrates only a very specific, although sophisticated, set of capabilities. There are various examples of editors and document systems which combine voice with text and other visual media. There are also systems which allow facilities such as simultaneous access to a document for editing from several terminals. Examples of such systems will be given in the chapters of this dissertation which deal with high level integrated facilities.

There is one notable project, the Etherphone project at Xerox PARC [Swinehart 83], which aims at a generalised architecture for flexible and extensible user-level integration. Some of the work described in this dissertation was done under the auspices of that project. The Etherphone project has deliberately concentrated on the application level: it aims to say nothing about the network and little about the servers which support those applications, specifically so that the applications architecture may be portable across a variety of for example PABXs from various manufacturers.

Work at Columbia University, based on the MAGNET hybrid network, is beginning to address high level applications. However most of the results from that project so far concern low level network performance.

There has been quite a lot of work on packet voice transmission. Most of the early work was done under the auspices of the ARPAnet [Forgie 75, Weinstein 83]. Since then many people have proposed simple (and usually remarkably similar) packet protocols [e.g. Montgomery 83], generally without regard for the delay characteristics demanded for practical telephony systems. There has been some work on human factors aspects of packet connections, which will be mentioned in later chapters. The protocol architecture of the UNIVERSE project was an interesting environment for mixed traffic transmission: some of the work described in this dissertation was done within this architecture.

1.6 Constitution of this Dissertation

1.6.1 Thesis

The subject area for this thesis was set out in section 1.2 and rough sketches of approaches to various aspects of the problems have been given in section 1.4. The aims given in section 1.2 represent a considerable area for work and are in fact the objectives of the ISLAND project in the University of Cambridge Computer Laboratory, a project involving several people.

The contributions to the ISLAND project which are claimed in this dissertation as original work constituting a thesis fall into three categories. First there is the outline design of the whole ISLAND architecture. This covers network support and protocols for integrated services, design of a PABX layer on top of this and above that facilities for user level Integrated Services, as set out in 1.2.

Whilst the way in which the components of the PABX system fit together is part of the thesis, detailed design of some has been the work of others. An example is the control server, whose purpose will be discussed in chapter 4. The need for and role of such components is part of this thesis, but their detailed construction is not a part of the arguments put forward here and forms independent research efforts.

The second component of the thesis is detailed work on implementation of parts of an Integrated Services system. These parts have been selected as those particularly required in order to justify the overall design and each represents a significant research effort.

In order to justify more of the overall design than could be implemented in detail, we have experimented in outline with a number of ideas relevant to the design. In several cases demonstration systems were designed by us and implemented by others as projects for the Cambridge Tripos and Diploma in Computer Science: where implementations were not done by us this is acknowledged in the text. Only work for which we can claim at least the design has been included. Miscellaneous demonstrations of relevant ideas form the third constituent of the thesis. At the risk of presenting them without sufficient justification of design decisions for each, they have been included to justify that the overall architecture is coherent and consistent at a higher level.

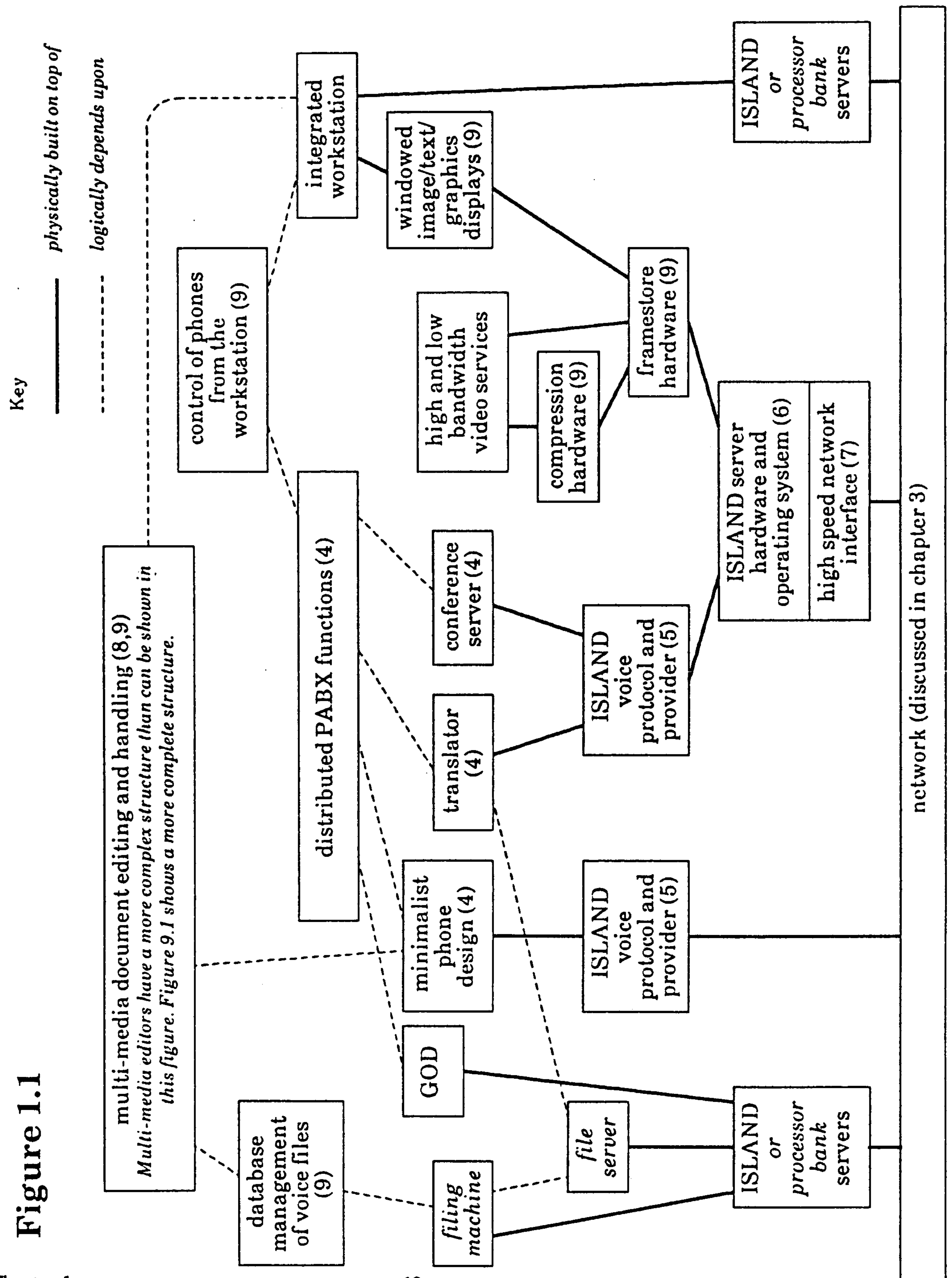
1.6.2 Topics Covered

Chapters 4 to 7 of the dissertation represent a top-down system design from the level of a PABX service down to network hardware. Chapter 4 puts forward a PABX design, using cheap simple telephones, more complex voice handling servers and a control system. It proposes techniques which can be used to reduce the cost and facilitate the construction of an advanced Integrated Services PABX. Much of the overall architectural design of ISLAND is described within this chapter.

Design of the partitioned PABX is closely tied in with voice protocols and choice of a network to support it. ISLAND protocols are the subject of chapter 5. In the design of these voice protocols, much use was made of experience gained from voice system design which we undertook as part of the UNIVERSE project. The design of protocols for UNIVERSE is a major piece of work that is presented in chapter 2. Chapter 3 considers good and bad features of the UNIVERSE approach. It also discusses various other issues relating to the transmission of voice in a network, which are of importance to the design of ISLAND. Finally it deals with the choice of a network for ISLAND and the properties of that network in relation to multi-media traffic.

The UNIVERSE work and related issues appear in this dissertation before the overall ISLAND design because an understanding of the characteristics and demands of voice traffic affects that design. The reasoning presented in chapter 5 relies on all three previous chapters. Chapter 5 describes in detail first an ISLAND voice protocol and then the way in which it can be implemented efficiently in both phones and voice servers, that implementation being specified as a service called the 'voice provider'. Chapter 5 represents significant work by the author on the design of protocol and provider, but implementation of both phones and voice provider is the work of others. Chapter 5 closes with discussion of just how simple a phone can be made as a

Figure 1.1


Key
physically built on top of
logically depends upon
multi-media document editing and handling (8,9)
Multi-media editors have a more complex structure than can be shown in this figure. Figure 9.1 shows a more complete structure.
database management of voice files (9)
control of phones from the workstation (9)
integrated workstation
distributed PABX functions (4)
windowed image/text/ graphics displays (9)
high and low bandwidth video services
compression hardware (9)
framestore hardware (9)
conference server (4)
translator (4)
minimalist phone design (4)
GOD
filing machine
file server
ISLAND voice protocol and provider (5)
ISLAND voice protocol and provider (5)
ISLAND server hardware and operating system (6)
high speed network interface (7)
ISLAND or processor bank servers
ISLAND or processor bank servers
network (discussed in chapter 3)

result of the design principles in chapters 4 and 5.

Chapters 6 and 7 continue the top-down progression of 4 and 5. Chapter 6 concerns design of software and hardware for servers suitable for voice services. The chapter represents a lot of work on implementation but rather less research interest. Topics of relevance to the ISLAND design set out in the chapter are choice of an operating system and design of network handling software to enable the voice provider service to run with satisfactory performance. The chapter also looks at how servers for ISLAND use should be loaded and controlled as part of external resource management. The end of the chapter outlines a demonstrator project showing the feasibility of building a conference server according to the ideas in chapters 4–6.

The lowest service level is dealt with in chapter 7, which concerns the design of a high speed network interface and explains why special work at that level is required in order to support integrated traffic. Largely due to the lack of a suitable working network for most of the period of this research, the interface described has not been fully built and tested. Chapter 7 has been included because much of the reasoning behind the design is the result of experience described earlier in the dissertation and because the design has reached a sufficiently advanced stage to be worthy of some description.

The architecture expounded in chapter 4 is intended to both be a useful service level in itself and support higher level Integrated Services, i.e. new integrated facilities at the user interface level. Chapters 8 and 9 attempt to show how it does so. Chapter 8 describes a single and self-contained piece of work on incorporation of voice into documents. It reveals some of the user interface issues concerned with handling voice in company with other media.

As well as representing a significant piece of work in itself, chapter 8 raises issues concerning storage and manipulation of the voice components of a document. These are discussed in the first part of chapter 9, which attempts to show how various parts of the ISLAND infrastructure fit together to support multi-media documents. The remainder of chapter 9 presents a miscellaneous series of possible integrated services together with notes on how they can be implemented in the ISLAND scheme. Several demonstrator projects are mentioned. Much of the work described relies on a framestore which was designed and built to support image-based user level facilities and whose features are outlined.

The components described in chapters 4–9 are intended to fit together as a single coherent architecture. The diagram given as figure 1.1 may help the reader to understand the interrelations between components. In the figure, objects labelled in italics are components which already existed and whose characteristics were known when the ISLAND was designed. Where work on an object is described in this dissertation, the chapter where description of the work is to be found is shown in the figure.

Chapter 10 presents a summary of the important results and an estimation of their validity, followed by notes on promising areas for further work.

1.6.3 Working Environment

With the exception of chapters 2 and 3, which describe early work that gave us experience on which to base what was to follow, all the work described in this dissertation represents part of our Integrated Services design. At the end of chapter 3 there is a discussion of the network that we wished to use for ISLAND and the properties that make it an appropriate choice. However that network was not available during nearly all the time when the work described was done. The work was therefore done on other networks. In all cases detailed design work has been carried out at a level at which the substitute environment offered the required features, with the exception of work described in chapter 7, which is so closely coupled to the network that no alternative environment could be used. It was noted above that the design in chapter 7 has not been fully proved. In chapter 3, the discussion of choice of a network is followed by a comparison with the other networks used, showing the limitations of these networks and the extent to which they affect the validity of conclusions to be drawn.

Apart from use of a different network, various demonstrator projects (such as the voice mail system mentioned in chapter 9) did not use the ISLAND voice protocol and voice provider environment. The object of these projects was to gain some idea of the usefulness and feasibility of some of the services that the ISLAND design would be able to support. That experience was gained in parallel with designing and building the overall infrastructure. Use of non-generalised architecture for these projects does not invalidate the lessons we learnt from them.

2. The UNIVERSE Voice Experiments

This chapter gives a detailed description of the UNIVERSE voice protocol and the rationale behind its design. The local area network upon which the protocol ran was the 10Mbs^{-1} Cambridge Ring. The low level structure of this network has some features which are useful for integrated traffic, but they do not affect the design as set out here and hence will not be discussed. The protocol structure of the UNIVERSE network will be discussed from the packet level upwards, since these levels are relevant to this work and also influenced our design for ISLAND.

This work was undertaken prior to the design of integrated protocols for ISLAND. Several proposals have been published on how voice might be carried in a packet network [e.g. Montgomery 83], but few have been followed up by practical implementations. The UNIVERSE work provided valuable practical design experience when the ISLAND design was undertaken; although the UNIVERSE was a substantially different network environment from ISLAND, it was a packet network and many of the same design decisions had to be made. Following on from this chapter, chapter 3 will present lessons learned from this design—both the problems encountered and the features that worked well.

An understanding of the UNIVERSE protocol architecture is important to the material presented in this chapter. The important characteristics of data protocols in more general terms are relevant here and to several parts of this dissertation, so we will begin by discussing them.

2.1 Data Transfer Protocols

Data protocols can be classified into two basic types. The first can be termed the single shot or 'question and answer' protocol. One station sends a request to another and expects a reply within some time. If no reply arrives the request may be retried. Single shot protocols are

(a) used for requests which are idempotent (such as 'give me bytes 1000 to 1999 of the file "ERIC"') or which produce a reply that is meaningful independent of other replies (such as 'what is the time?').

(b) the basis of many remote procedure call (RPC) implementations.

The other type is the reliable stream protocol, X.25 [CCITT 80] being a well known example. Stream protocols ensure that some quantity of data is correctly transferred exactly once. They typically do this using block numbering, acknowledgements and retry strategies. Windowing mechanisms are often used in order to increase throughput. Stream protocols are used for such applications as remote terminal connections and file and mail transfer protocols.

Stream connections are generally set up using an end-to-end negotiation similar to the single shot protocol—during this preliminary exchange the two ends may negotiate over parameters for the connection and the network between them may use the exchange to set up routing for the stream.

2.2 The UNIVERSE Network

This consisted of a series of 10Mbs^{-1} Cambridge Rings linked together by (a) satellite bridges and (b) local Ring-to-Ring bridges. The Cambridge Ring is a slotted ring and data is transmitted across it in trains of slots known as 'basic blocks'. Stations on the UNIVERSE network did not support simultaneous transmission or reception of more than one block. These blocks correspond to what is generally termed the 'packet level' and are the lowest level of the transport protocol hierarchy of interest in this chapter. Bridges which join rings together transmit and receive in units of one block's worth rather than one slot's worth of data. The satellite bridges were composed of a single broadcast satellite channel, which provided 1Mbs^{-1} of bandwidth to be shared between six earth stations transmitting to one another in any combination.

The UNIVERSE architecture supported both of the types of protocol outlined above: they were called Single Shot Protocol (SSP) and Byte Stream Protocol (BSP) respectively. The BSP was set up using an exchange not unlike an SSP request and reply, termed an OPEN and OPENack. The major difference between these pairs concerns network routing. Two stations could communicate using SSPs or BSPs regardless of how many bridges of whatever kinds lay between them. An SSP request not only told its receiver how to reply to the requester but also caused the network to set up a return path for the reply when it came back. The OPEN/OPENack exchange set up paths in both directions between the two parties, to allow stream data and acknowledgements to flow.

We referred above to a satellite bridge rather than to a satellite *gateway*. This was to emphasise that no protocol translation was performed in the network at all. Bridges appeared transparent to any pair of stations. All services were accessed by means of nameservers—objects provided on each LAN, which transformed a service name into an address. Nameservers interacted with bridges to set up routing for SSP requests and OPENs. A transmitter could not see transmission through a bridge as different from transmission to another station on the same ring.

The UNIVERSE network was used as a vehicle for integrated services research, supporting distributed computing experiments; image and document transfer and packet voice links. One major aim of the project was to discover what type of network and protocols would be suitable to support such an integrated services environment.

2.3 Lightweight Channels

The flexibility of the UNIVERSE protocol architecture for integrated services use, and for voice in particular, stemmed from the fact that connections could be set up with a wide variety of characteristics.

The OPEN/OPENack sequence was not logically part of the BSP and it could be used to set up connections other than BSP. At its simplest it would merely establish two paths along which the user could operate as he chose. In the UNIVERSE project many applications used such a lightweight protocol rather than the full BSP mechanism—such facilities as reliable file broadcast[1] [Waters 84] were very much better served by building upon lightweight connections. A skeleton protocol is ideal for voice, which is not usefully served by a 'reliable data stream' service but for which circuit switching is an unnecessary inefficiency.

To explain the last point, a 'reliable data stream' service, such as X.25 level 3 or the TCP/IP connection level, is undesirable for voice since it imposes error recovery by retransmission. In the case of voice such retransmission introduces a step function delay, exactly what is not wanted. For packetised voice, it is generally true[2] that it is better for packets not to arrive at all than to arrive late—if they arrive too late to be 'played back' they can only be discarded by the receiver. They are most likely to have been late due to congestion of the network. Possibly therefore it would have been better, given a suitably sophisticated network, for the network to throw them away in mid-flight once they became sufficiently delayed to be useless to the receiver. This would ease the network congestion and help the following voice packets to arrive in time.

The UNIVERSE network supplied connections with known error rates. These were very good, a connection through a series of LANs plus one satellite hop giving an error rate of around 10^{-9}. Packets received containing errors would be so marked: they could optionally still be delivered to the receiver under most circumstances. The network did not impose any flow control—it was up to all users to send information at a suitable rate. It was therefore possible to overload bridges, but these were designed to handle high traffic rates and contained substantial buffering to cope with instantaneous loading problems. In the satellite bridge, all queues of traffic for different connections were treated equally, regardless of the length of the queues, which was beneficial to the voice/data contention problem. If congestion did occur, a congestion control algorithm released enough buffering to allow the bridge to continue functioning. This was done by deleting the longest queue, an approach found to be successful in overcoming short term congestion

(a) by discriminating against 'rogue' processes which suddenly produce large amounts of frivolous or malicious data

(b) particularly helping voice, whose moderate length queues need to be expedited during conditions of heavy load.

Once a voice connection had been established by an OPEN/OPENack exchange, packets could be sent transparently through the network without further protocol overhead. No formal closing parameters were needed, connections being deleted after a certain period of inactivity. This scheme was feasible because the mappings required within the bridges to produce the transparent connections were not a scarce resource.

All of these characteristics were favourable to voice. The bit error rate (BER) offered was far better than that required for good voice quality. Therefore it was better to deliver and play back packets containing errors (provided that the addressing and protocol information was known to be

[1] Reliable file broadcasts were provided on top of the lightweight broadcast mechanism available in the UNIVERSE network. As a file transfer progressed, each receiver needed to acknowledge data that it had received from the broadcast. Acknowledgement messages were implemented on top of the single shot protocol. In this application it was not appropriate to use the network-provided reliable stream protocol, but rather to build a high level protocol above two other protocols provided by the network.

[2] It will be true for voice carried in a 'strict time' protocol, which will be defined below.

intact) than to produce gaps in the sound. To employ error recovery by retransmission could only widen the distribution of packet delays through the network, which is highly undesirable for voice traffic, particularly as the only benefit would have been to enhance an already adequate BER. This fact is emphasised by the satellite connections, where retries would have produced a minimum step function delay of 0.5 seconds.

The transparency of the lightweight UNIVERSE protocol meant that very simple microprocessor-controlled telephones could implement a voice system. Once a connection had been set up, the microprocessor merely had to packetise and send samples in one direction and play back those received in return. The network imposed very little protocol overhead to be handled by the phone.

2.4 The Voice Protocol

2.4.1 Packetisation Effects

Packet voice systems can potentially be more efficient than circuit switched types in their use of bandwidth [Weinstein 78, Weinstein 79], if buffering is used within the network at times of overload and if the network is allowed to introduce small additional delays (due to use of this buffering). For both types of switching, bandwidth utilisation is made more efficient by use of silence suppression. A circuit switched system will be required to perform with a specified maximum probability of 'blocking' (i.e. there being no bandwidth available for allocation to an active channel). A packet switch can trade the introduction of delay against blocking level to produce greater efficiency.

The drawback of using a packet network for voice, particularly if the network carries other types of traffic which are not immediately pre-emptible, is that the transmission delay through the network is subject to statistical variations. When a voice channel is 'active' packets will be produced at a constant rate by the originator, but when delivered by the network they will have acquired a significant statistical distribution, or jitter. This distribution is likely to vary as a function of the total load on the network. Clearly sufficient buffering at the receiver would smooth out these effects, but at the expense of possibly unacceptable delays, were the worst case of jitter to be provided for.

Protocols for packet systems can be of two types—*strict* timing and *delayed* timing [Weinstein 83]. In both cases, all notions of elapsed time in a receiver may be measured not relative to the transmitter but from when the first packet sent along a connection begins to be played back at the receiver. (This should be some short time after the first packet was actually received, the time being set to allow for the amount of jitter expected in the system.)

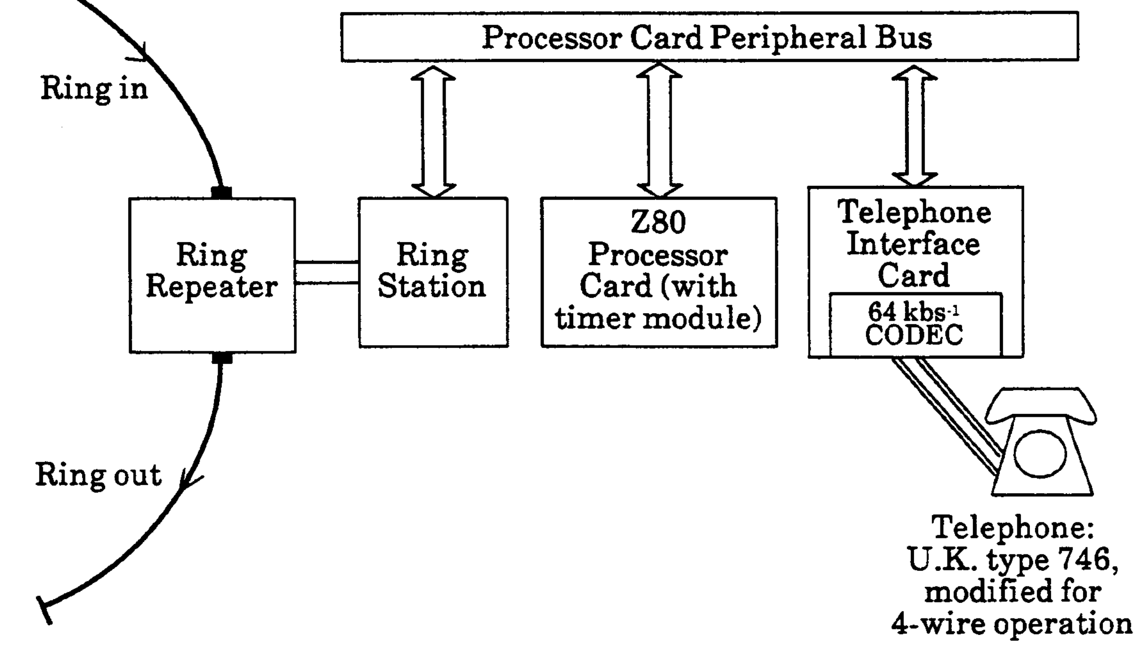
In the strict time case, where a packet arrives too late to be played back it is simply ignored—the receiver sticks rigidly to its measure of elapsed time throughout a call. In the delayed time approach, packets which arrive too late to be played back are taken as an indication that network loading is becoming heavier, so that the timing of playback should be retarded. This will prevent the sound from becoming fragmented due to the continual late arrival of packets. The timing should be held back for as long as the conditions of heavy loading continue to produce a widened distribution of the jitter.

The UNIVERSE protocol was of the second type—it attempted to adapt the delay in a call to the network conditions during the course of the call so as to produce acceptable sound quality. Any protocol with an automatic retardation facility for conditions of heavy load must of course have a matching 'advance' facility to reduce the delay as network loading subsides—both of these mechanisms are discussed below.

2.4.2 Implementation of Telephone Interfaces

The UNIVERSE network contained many small processor systems used for discrete system functions requiring no peripherals other than the ring. These were mainly Z80-based processor cards, connected to the station and repeater circuits which comprise a standard Cambridge Ring node. The phones were realised using a card[3] which was seen by the Z80 as a peripheral and which was linked also to a slightly modified version of a standard (type 746) U.K. telephone set. This arrangement is shown in figure 2.1.

Figure 2.1: Universe Telephone Hardware


Processor Card Peripheral Bus
Ring in
Ring Repeater
Ring Station
Z80 Processor Card (with timer module)
Telephone Interface Card
64 kbs-1 CODEC
Ring out
Telephone: U.K. type 746, modified for 4-wire operation

The peripheral card contained a 64kbs^{-1} A-law PCM CODEC, connected to the telephone set by a 4-wire connection. (Standard phones operate with a 2-wire interface, to save on cabling to the local exchange. For this application a 4 wire interface is simpler to realise. It also reduces problems of echos.) Slow and fast loop disconnect detection (for 'hook switch' status monitoring and detection of dialling pulses) was implemented in hardware, as was silence detection. Silence detection consisted of comparing the (rectified) signal from the handset microphone with a preset reference voltage. This arrangement could not cope with wide variations in background noise but

[3] built by Martin Slater of GEC Marconi Baddow Research Laboratories, which was a member of the UNIVERSE consortium

a single preset level for each room was found to give satisfactory results. Silence detection was presented to the Z80 as a status bit, allowing the Z80 to integrate the silence detection over any period.

A timer on the processor board was used to interrupt the Z80 every 125µs during a call, to transfer voice samples to and from the CODEC. The timer was also used for timeouts, tone generation and other phone protocol functions.

2.4.3 The Telephone Code

The code which realised the telephone protocol was divided into two layers—the call set-up system and the 'real time' monitoring mechanism which ran during a call. The software was written in Z80 assembly language (necessarily, on grounds of speed) and totalled about 8 kbytes of code, leaving up to 54 kbytes of space for packet buffering and status dumping. This code was automatically loaded into all Z80 telephones upon power-up by a local booting service [Needham 82]. The code used between 80% and 100% of the processing power of the Z80, depending on the packet size.

Various aids were provided in the code in order to monitor its performance internally. These consisted of simple status information which was dumped into a region of the Z80 memory whilst the protocol was running. This information could then be inspected using a remote debugging system.

(This method of debugging and monitoring has the advantages that the code required in the phone is

(a) inexpensive—simple minded dumping of information does not slow down the protocol appreciably whilst it is running, so that the process under observation is perturbed minimally by the act of observing it.

(b) simple—it requires a minimum of assembly code.

When the protocol had been running for some time, a complete dump of the information recorded could then be collected and analysed in a remote machine using programs

(a) not of critical time in nature

(b) written more effortlessly in a high level language.)

2.4.4 Call Set-up

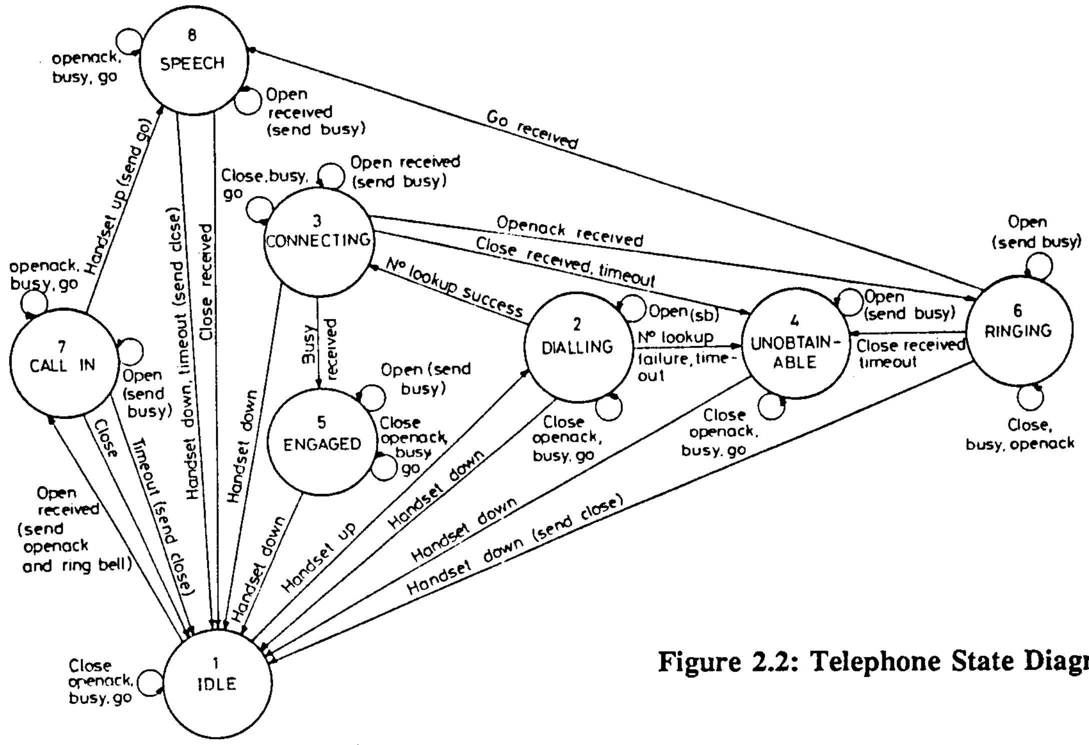


Figure 2.2: Telephone State Diagram

Figure 2.2 shows the state machine for this layer of the protocol. It provides correct transitions between dialling, ringing, speaking and various 'fault' conditions—both expected conditions such as engaged and unexpected inconsistent states. States are shown as numbered circles, with transitions given as labelled arrows. The labels show the stimulus causing the transition and in some cases an action taken during the transition (shown in brackets). Normal telephone conventions are used and standard U.K. tones were produced in software by the Z80. The state machine is fairly conventional, so that only a few particular features need to be mentioned.

The number which a user would dial to start a call was of the form

<service type> <site> <user number>

The site numbers corresponded to the different LANs or LAN clusters linked by the satellite and the user numbers connoted individual users at each site. (For convenience the latter corresponded to users' existing PABX extension numbers.) The service type specified

9:	set up call to specified user
1:	play back a message for specified user
0:	record a message for specified user

In order to contact the correct service or telephone, the calling phone used a local directory service, implemented in the local nameserver. In the case of service type 9, the phone would look up the name "RealTimeVoice-<site><user number>" in the nameserver. (In the case of service types 0 and 1 lookup was performed on the name "RecordedVoice-<service type><site>", since whilst there might be several machines running a store-and-forward message service, only a single one would be providing all of either recording or playback facilities for a single site.)

The nameserver would return an address, plus a function number which will be explained below. The phone would send an OPEN to the given address and wait for an OPENack in return. The OPENack might be a normal OPENack, indicating that the remote phone was now ringing, or a BUSY OPENack, indicating that the remote phone was unavailable at the time. (There were of course suitable timeouts for failure during the set-up process.) The BUSY OPENack had the form, in terms of Cambridge Ring protocol definition [JNT 82], of a 'fail' OPENack, i.e. it would clear down the connection mappings through any bridges on the route. After a normal OPENack, the calling phone would expect a GO packet, indicating that the called phone had been lifted off hook.

2.4.5 Initial Jitter Estimates

The voice playback mechanism in each phone would start up when it received the first voice packet along the connection. In order to allow for some level of jitter along the link, this first packet would be delayed for some period of time before playback was started. The jitter expected in the case of any particular call could be anticipated to be a function of the connection path. Therefore it appeared sensible to tailor the amount of initial delay of the first packet to that connection path. This was the purpose served by the function code given by the nameserver. When any telephone service name was looked up, the nameserver replied with not only an address, but also a characteristic of the route. The function code (which simply prescribed the initial delay as a number of 125µs periods) was not only retained by the calling phone but also passed through in the OPEN block, so that both ends of the connection would know the expected characteristic of the path.

The name lookup had to be performed locally rather than in an arbitrary nameserver because the function code needed to reflect not only where the remote machine was but also what path would be set up to it. This was not the manner in which use of nameservers was prescribed for all of the other UNIVERSE applications. In all other situations, the two partners in a data exchange would not know the locations of each other. Apart from services that were duplicated at several sites (such as for example bootservers), most names would map onto single machine addresses. Bridge/nameserver interactions would be used where appropriate to set up paths and then transformed addresses would be sent to clients by the nameservers. This all reflects the claims made above, that the network topology was transparent as far as any two machines requiring to communicate were concerned. For most transfers this claim was indeed true, since they would only be concerned with the logical steps taken in order to communicate. For voice the situation is different: since so much emphasis must be placed on delays, the routing of a call is always of importance.

The function code was used to suit the nature of the remote machine as well as the route to it. So far, the UNIVERSE store-and-forward message service has been alluded to only briefly. Recording and playback facilities, designed to integrate with the 'real time' voice system, were implemented on GEC 4090 minicomputers. Unlike the phones, these machines were not running their voice service as a single task in time-critical manner—they ran a conventional multi-user operating system and were not dedicated to the voice service. Hence when playing back messages they were unlikely to be able to launch packets with the regularity expected of a phone, so that jitter estimates would need to be set higher in this case. There is of course no 'real time' constraint on playback of a stored message, so that the long delay between the transmission onto the network and eventual playback in the phone of a voice packet would be of little consequence. Given sufficient buffering in the phone, the flexibility of the function code mechanism allowed for a very large jitter estimate to be given for playback of messages from the minicomputer—hence the speech quality was 'smooth' even though the minicomputer was launching packets irregularly onto the network.

2.4.6 The Real Time Adaptive Mechanism

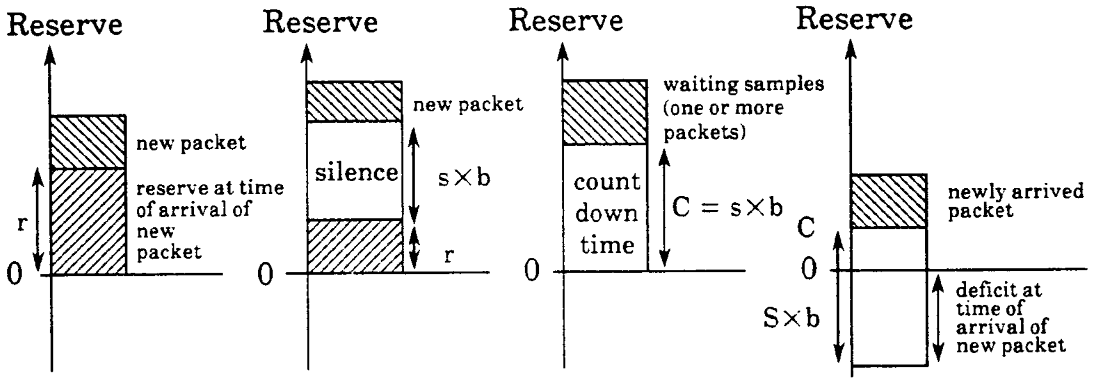
The connection of two phones in a 'real time' manner will be assumed in the following description of the adaptive mechanism. (The procedure applies equally to the store-and-forward system, except that the constraints are much less severe than for 'real time' connections.)

Transmission was straightforward. Once a voice path had been established, the transmitter would begin to assemble voice samples from the CODEC into packets, checking the status of the silence flag. When a packet was complete, a sequence number would then be added, this number being incremented every time a packet was assembled. The transmit mechanism then checked to see if the complete packet consisted entirely of silence. If so it would simply mark it as 'successfully transmitted', otherwise it then sent it on the outward voice connection.

Because the sequence number was incremented every time a packet was generated, whether or not it was actually sent, the sequence numbers received at the remote phone would be a linear function of time of generation and hence would enable the receiver to stay synchronised, detecting missing packets from gaps in the sequence number. The first packet of a stream would always be sent, even if it contained only silence, since the first packets in each direction were used to start the reception timers. During long periods of silence, packets were also sent periodically to prevent the connections from timing out.

The playback mechanism was more complex. Once the first packet had been received and the prescribed period waited, normal playback mode was then entered. Throughout the playback mechanism, the concept of a *reserve* was used, being the number of voice samples the receiver had in hand ready to be played back. The reserve held at the time of arrival of a packet was of particular importance, giving an indication of how close to a gap in the voice stream the mechanism was running at that time. A negative reserve (or *deficit*) is an expression of the number of 125μs periods since the playback mechanism last had any samples to play back.

Figure 2.3: Real Time Buffering System



Reserve
new packet
reserve at time of arrival of new packet
r
0
Reserve
new packet
silence
s×b
r
0
Reserve
waiting samples (one or more packets)
count down time
C = s×b
0
Reserve
newly arrived packet
C
0
S×b
deficit at time of arrival of new packet

2.3a: Playback without deficits or missing blocks

2.3b: Missing blocks detected, reserve positive

2.3c: Playback of silence

2.3d: Missing blocks detected, reserve negative

Consider first the case where the reserve is always positive and all packets arrive with consecutive sequence numbers. As each packet arrives the situation will be that shown in figure 2.3a. The number of samples in reserve just before the packet arrived, **r**, is recorded for the use of the dynamic adjustment mechanism (described below) and then the new packet is added to the queue of samples awaiting playback.

Now consider the case of a packet which arrives whilst the reserve is still positive, but bears a sequence number such as to indicate that **b** preceding packets are missing. Such a packet is placed on the queue of samples but the reserve which is recorded must now be **r+S×b**, where **S** is the number of voice samples in a packet. This is the number of 'virtual samples' that were held in the buffer when the new packet arrived. Logically the situation at this stage is that represented in figure 2.3b.

When all the samples from previous packets have been played back, a delay must then be inserted before this new packet is played. The playback mechanism enters a 'countdown in silence' state—logically the reserve held in the phone is that shown in figure 2.3c. For an interval of **S×b** 125µs periods nothing is played back. At the end of this time normal playback resumes.

When a packet arrives with a non-consecutive sequence number, it is most likely that the receiver will hold a deficit of samples. Considering now the general case of occurrence of a deficit, once the samples held in the phone have all been played back, then if no new packet arrives the receiver merely plays back nothing, decrementing the reserve count once every 125µs.

When a new packet arrives, the sequence count may indicate **b** missing packets, as above. In this case, the product **S×b** is added to the reserve count. This may have the effect of drawing the reserve count positive (as would normally be expected when the transmitter sends the first packet to follow a period of silence). The reserve status will then look like that shown in figure 2.3d. The system will now behave just as in figure 2.3c except that the countdown period **c** will not be S×**b** but the amount by which S×**b** exceeds the accumulated deficit. A reserve of **c** will be recorded (this again being the number of 'virtual voice samples' in hand).

When a packet arrives during the time of a deficit, the addition of the product S×**b** to the reserve may not in fact draw the reserve positive (in particular if **b** is zero). As mentioned above, this voice protocol is of delayed rather than strict time type. Therefore when a 'genuine' deficit occurs (i.e. a deficit after compensation for any missing packets), instead of discarding as many samples as the value of the deficit, the protocol starts playback immediately from the first sample in the newly arrived packet. The value of the deficit is recorded for the use of the dynamic adjustment mechanism.

2.4.7 The Dynamic Adjustment Mechanism

The procedure outlined above copes automatically with the case of jitter becoming greater than expected during the course of a call, by holding back the playback mechanism as just described. There needs to be a corresponding mechanism to cope with the build-up of a greater reserve than actually needed, once conditions improve again. The network may also be subject to gradual drift in mean packet delivery time and the protocol should be able to accommodate this.

Both of these needs were met by the dynamic adjustment mechanism. This mechanism was 'fired up' in the phone at regular intervals, values of 5 to 20 seconds being used at various times in the UNIVERSE experiments. When fired up, the mechanism would examine the queues of deficits and reserves. If there were deficits, both queues would be cleared and no further action would be taken, other than to record the value of the last deficit which had occurred. If the deficit queue was empty, the reserve queue would be examined. If its length was deemed not statistically significant, it would be left to grow further. (The criterion used for statistical significance was that its length was at least the number of packets which would be played back, in the absence of deficits and missing packets, during the interval between consecutive adjustment sequence runs.) If the queue was sufficiently long, the algorithm would find the smallest recorded reserve. The timing of playback would then be advanced by half this value. The advance would be effected by shortening the next period of silence to occur. When next either of the situations shown in figures 2.3c or 2.3d occurred, the countdown value **c** would be reduced by the calculated advancement value (unless **c** were less than the advancement value, in which case **c** would be set to 0). In this way timing advancement could be effected without the listener being aware of its occurrence.

In order to prevent a rapid oscillation between timing advancements and similarly sized deficits, various schemes can be concocted whereby for example any timing advancements that are smaller than the last recorded deficit are cancelled. However in practice such oscillation was neither rapid nor noticeable to the user, because the dynamic adjustment routine firing frequency was set suitably low.

2.5 Typical Behaviour of the Protocol

The protocol was tested out on single rings, across ring-to-ring bridges and across satellite connections. In the first two cases the results were very good. The graphs discussed below show the protocol coping with particular conditions of network loading. Connections across the satellite, although producing satisfactory speech quality for most of the time, did suffer from two particular defects, also to be discussed below.

2.5.1 Terrestrial Connections

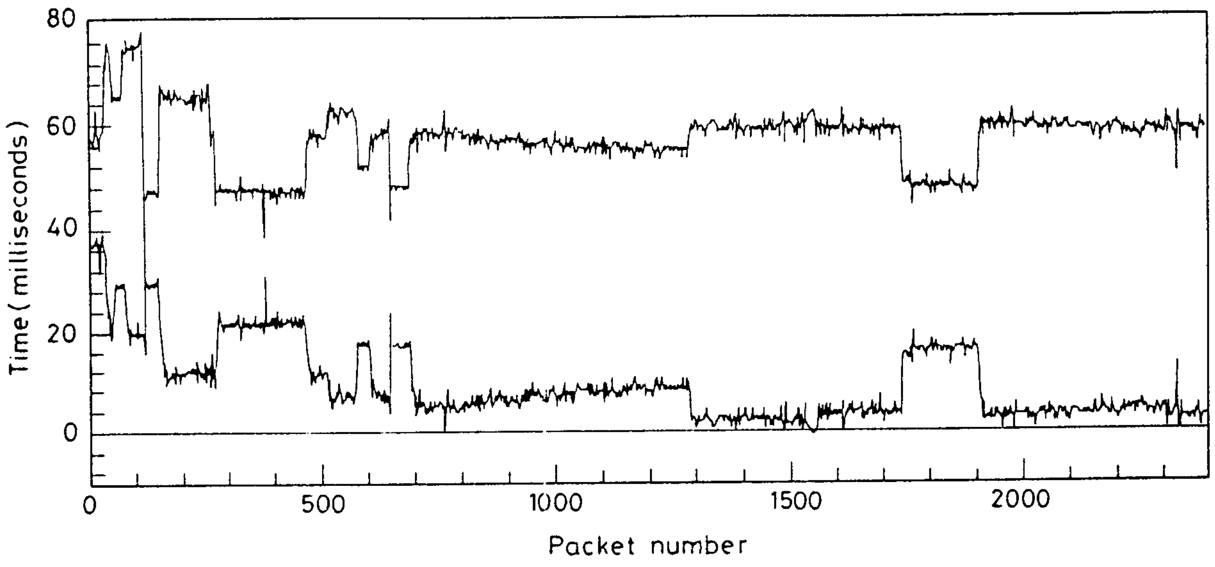


Figure 2.4: Adjustment of Excess Reserve, Followed by Light Loading

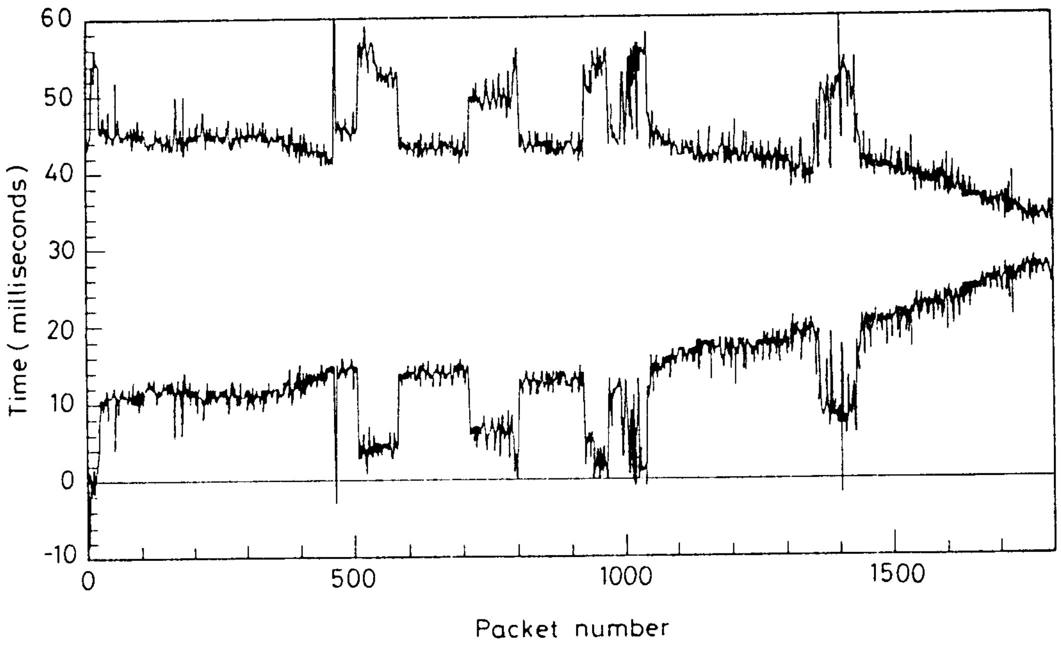


Figure 2.5: Performance Under Heavy Bridge Loading

The graphs in figures 2.4 and 2.5 show typical behaviour of the protocol under various network conditions—they were both produced by processing information which had been dumped in the phones during conversations connected through ring-to-ring bridges. In both cases the figures show two traces. The lower is the reserve held in the phone as each packet arrived and the upper the transit time of the packet through the network. The reserve held is an absolute quantity (shown in milliseconds rather than as a number of voice samples) whereas the delay through the network is only a relative value (i.e. the absolute value on the time axis signifies nothing for the upper traces). Points where the adjustment mechanism caused a timing advancement can be identified by comparing the two traces. The manner in which the occurrence of a deficit served to retard the timing to cope with an increased level of network delay can also clearly be seen from the figures.

Figure 2.4 illustrates two different things: the second half of the trace (from a little after packet number 1200) shows a typical period of running under conditions of light bridge loading. Statistical fluctuations in the delivery time of packets can be seen. As previously noted, the protocol did not attempt to produce a complete absence of deficits under such conditions; only to keep their occurrence infrequent. The bridge was under light rather than zero loading—this can be seen from the dips in network delay (probably signifying no other use of the bridge at those instants).

Figure 2.5 shows the result of placing a sudden high load on the bridge. The level of reserve held prior to this load increase (at the beginning of the trace) can be seen to have been insufficient for the delays suddenly encountered. A single large deficit therefore occurred. This deficit retarded the timing sufficiently to cope with the new load. It can be seen that the load was bursty. (It was in fact a series of moderately sized file transfers.) Because the adjustment mechanism in the phone had a suitably long time constant to cope with such traffic (that traffic being fairly typical of the 'reliable stream' data transfers to be found in an ISDN environment) it did not compensate for the reduced delivery time in between bursts so rapidly as to cause another heavy deficit at the start of the next burst.

The first half of figure 2.4 shows the behaviour of the mechanism after a heavy load had subsided. At the start the reserve held was larger than necessary for smooth sound quality. (This reserve had of course been built up whilst the loading conditions required it.) Since the adjustment mechanism acted at regular intervals, producing a timing advancement of half the recorded minimum reserve each time it acted, in the absence of jitter and silence periods an excess reserve would be reduced exponentially. The reduction of the reserve can be seen by checking the values of reserve against some constant value of network delay (e.g. 57ms on the *time* axis) along the trace.

During the call from which the data in figure 2.5 were collected, the sound quality was not reported by the listeners to be noticeably poor. A single large deficit, particularly if it occurs within a word, might produce a noticeable click, but because the deficit retards the timing and hence prevents further large deficits, this occurs only infrequently and is not irritating.

2.5.2 Satellite Connections

The typical behaviour of the protocol as discussed above drew on examples from calls connected through ring-to-ring bridges. Connections made around a single ring behaved in a nearer ideal way, as would be expected. Calls made across satellite links, although producing good voice quality for most of the time, suffered from two problems not found on the other types of connection:

(i) the overall delay was considerably greater than it need have been (from consideration of the signal propagation to and from the satellite alone)

(ii) under certain network loading conditions, successive voice packets were delivered each one with a slightly longer delay than the previous one. This would occur at times for quite a long sequence of packets and because of the way in which the playback mechanism operated (as explained in section 2.4.6) such a series of small deficits led to a short click between each packet (sounding not unlike an audio signal which has passed along a cable containing a badly soldered joint).

Both of these are unacceptable and they identify two weaknesses in the system:

(a) the satellite bridge was not specifically designed to accommodate voice and hence did not make allowances for the specific requirements associated with voice streams. (For example, loss of window during periods of silence was directly responsible for problem (i), as will be discussed further in section 3.2.1.)

(b) the bridge contained no simple monitoring aids of the kind that the phone did. Where performance was poor (particularly in the case of problem (ii)) it was difficult to investigate the causes.

Point (ii) is believed to have corresponded to a persistent shortfall in bandwidth availability compared to the demands of the phone: this emphasises the fact that variable delay protocols can cope with a sudden increase in network loading only if at the higher loading level there is still sufficient bandwidth available for the voice. Otherwise the timing will be *incrementally* retarded, producing many (and worse still, probably regular) gaps in the speech. These problems have affected our views on Integrated Services gateway design, which is discussed in later chapters.

(Another aspect of the system architecture which adversely affected the satellite connections was the block level protocol of the 10Mbs^{-1} Cambridge Ring, which does not allow for multiplexing of block receptions by a node. This becomes noticeable when a fast node (such as a satellite bridge) and a slow node (such as a UNIVERSE phone) are exchanging blocks. This will be discussed further in chapter 7, which addresses Ring interface design. The issue does not however have a great effect on the arguments to be presented in the next chapter.)

3. Lessons from the UNIVERSE

Following on from the description of the UNIVERSE protocol, this chapter discusses the lessons learnt from that work and how they influenced the design of ISLAND. The issues to be discussed divide into four main topics. The first is silence detection and suppression. Many researchers have stated as an 'obvious' assumption that silence suppression needs to be used in order to gain the benefits from use of a packet network for integrated traffic. In this chapter a conflicting view will be put forward: we believe that within the local area context silence suppression involves provision of extra hardware at places where it is important to minimise costs and serves only to conserve a resource which is not precious.

The next topic addressed is whether integrated protocols should be designed as end-to-end or step-by-step connections and how their performance should be monitored. One essential prerequisite for successful design of integrated protocols is an appreciation of the effects of delay upon voice connections. Some of the important facts concerning delay in voice connections are discussed here—these consist of both observations from the UNIVERSE work and guidelines laid down by the PTTs and standards bodies.

The chapter concludes with discussion of the choice of a network for ISLAND work, in the light of the demands imposed on it by a mixed load of voice and data traffic. Since not all the work described in this dissertation was done using that network, a discussion of the validity of results obtained using the other networks is also included.

3.1 Silence Detection

3.1.1 Energy Thresholds and Clipping

The UNIVERSE silence detection hardware produced an indication of whether the voice input level had risen above a set threshold during the previous 60ms. To a first approximation, since the input signal had a known and roughly constant quiescent value, this represented an indication of whether the peak energy of the signal had risen above some given threshold during the 60ms. No single suitable threshold value can be found which suits the ambient noise levels of all rooms. Therefore the voltage threshold was made adjustable using a preset resistance: if it was set to a suitable value according to the room in which each phone was used this provided acceptable results.

Sending only the voice packets which were recorded whilst the silence detection hardware indicated 'sound' gave unsatisfactory results, however, particularly if the phones were set to use short packets. The sounds produced sounded *staccato*—there was pronounced clipping of the ends of words. The reason for this is that whilst the start of a word produces a transient, normally large enough to trigger the silence detection hardware, the energy level of a word often dies away gradually. Considering a word such as "three", the energy in the vowel "ee" is sufficiently low that the silence detection flag may change state during this vowel, clipping the word in a manner unacceptable to the listener. It is not possible to lower the silence detection threshold to a level which will make all vowel sounds qualify as 'non-silence' without causing the ambient noise level in a normal room also to qualify as 'non-silence'.

There are two obvious ways to combat this problem. The first is to use two voltage thresholds V_1 and V_2, where V_1 is greater than V_2, and to define 'sound' to be that occurring from the time when a level of V_1 or greater is detected until such time as no level of V_2 or above has occurred for some short period (e.g. 60ms). The other possibility is to use only one threshold, V, and to define a burst of 'sound' to occur from the time when the signal exceeds V to the time when the level V has not been exceeded for a rather longer period, such as 0.5 or 1.0 seconds. This period can be termed the 'hangover' time.

Both of these methods produce acceptable results. The first is the more reliable and less empirical: it has been used in several systems, not only for silence detection on phone links [Gan 86] but also for detecting boundaries of words in voice recognition systems [e.g. Lamel 81]. (Both of these examples employ small additional refinements in their algorithms.) The second method, used for example by Gruber [Gruber 83a], works most of the time if the right 'hangover' period is found. It is rather rough and ready, but has an advantage when used with delayed time protocols. For values up to about 200ms, the maximum amount of jitter that will go unnoticed by the listener is equal to the hangover time. (This is not surprising since the hangover time will dictate how many small silence periods there will be).

The second method was used in the UNIVERSE system: the choice was largely out of our control since when we designed the voice protocol hardware for the telephone interface had already been designed. Adopting the first scheme would have required either (slightly) more complex hardware or computation by software. As will be remarked below, the processing power for such computation was not available within the phone. Using the hardware as designed required the silence flag to be read only once every few tens of milliseconds; the 60ms holding time was mainly a 'de-bouncing' device to ensure that the flag could be read reliably by software.

3.1.2 Loss of Background Noise

Although reliable silence detection may be possible in most cases, this does not necessarily imply that silence suppression, at least of a simple kind, is desirable. Various experiences suggest that it is not always so. For example in the UNIVERSE experiments, telephone calls were often made from a noisy computer room (which had a high white noise level) to a quiet office. The users of such links found the complete disappearance of background noise whenever the speaker paused disturbing, and disliked even more the sudden appearance of a loud background sound just as the speaker began a phrase, this noise making the beginning of each utterance less easy to understand.[1] Other users of silence-suppressed links have expressed displeasure when confronted with situations where there is a conversation just audible in the background: that conversation is cut up according to when the main conversation is active, which is distracting whether or not the caller is trying to listen to the background sound. Another problem occurs with telephones in normal service and is easy to miss in an experimental environment. When there is no specific sound to listen to for a prolonged period during a call, if the listener cannot hear any general background noise, (s)he will very soon become concerned that the connection may have failed

[1] It was remarked in chapter 1 that loss of speech samples at the beginning of a word is more serious than in the middle: distorting the initial transient at the start of a word significantly reduces its intelligibility.

and will often hang up.

Of course people have used silence-suppressed transatlantic telephone links for nearly two decades without expressing any of these complaints. Generally the high noise level on these lines has masked such effects: listeners' attention has focused upon that noise. The situation is very different for local phone links in quiet environments, as experiences from the Etherphone project have shown.

A very simple way to avoid the first and last problems was tried in the UNIVERSE system. The hangover time was extended to about two seconds. The receiver then assumed that the last second of samples which it had received before transmission ceased consisted of background noise. When there was a deficit of samples this second would be played back repeatedly until more samples arrived. Most of the time this worked very well. The background noise did not change appreciably during silent periods and hence the user did not notice when the repeated replay cut in and out. Of course there were pathological cases where the second's worth of samples contained a distinctive feature (a speaker in the background or a door slamming for example) which could then be heard repeating at 1Hz. This sounded bizarre.

More complex methods along these lines could avoid such problems. The telephones might hold a white noise sample of say a second in length. When sound samples were not being received, the ambient noise level could be calculated from the last second's worth received (assuming a suitable hangover period). The white noise sequence would then be played back repeatedly until samples were again received, adjusted by some multiplication factor to match the ambient sound level.

It should not be forgotten when discussing these techniques that the longer the hangover the less bandwidth is saved by use of silence suppression.

A different technique was used by DeTreville and Sincoskie [DeTreville 83], which relied on the fact that the ear is less sensitive to small, gradual changes in amplitude than to large, abrupt ones. They attenuated the sound by n dB when n milliseconds into the hangover period. No mention is made in their report of a similar approach to easing the problem at the start of a talkspurt. Like the previous technique, this one calls for multiplication of logarithmically encoded speech samples—use of lookup tables is suggested in the report.

An alternative which would solve the problem of coherent background noise cutting in and out according to the level of the foreground sound would be variable rate compression of the digitised voice stream, with for example no compression during sound periods but fairly heavy compression during silence. This would bring channel utilisation down to near TASI predictions, whilst retaining something of the background all of the time.

These methods have one thing in common: they all require considerable processing in either software or hardware, which implies additional cost for every object that employs such a bandwidth usage reduction technique. The question must therefore be asked why we are trying to reduce the utilisation: the answer depends on the routing of each call. If a call occurs between two phones on the same ring or even between phones connected by Ring-to-Ring bridges, then the bandwidth of the networks, as we have already remarked, is not expensive and we are not concerned to conserve it. Furthermore, silence suppression in a simple Ring telephone implies an increase in the complexity and cost of each phone. The situation is very different for a satellite link: bandwidth there is very expensive, so that the extra cost of bandwidth reduction equipment may be justified. This implies that the place to put voice compression equipment is not in each phone but in the access equipment at each end of an expensive link.

3.1.3 Heads of Phrases

It was said above that the beginning of a sound burst can easily be detected because it represents a large enough transient to activate a simple 'silence detection' circuit. In practice the assumption that the beginnings of words will sound satisfactory does not always hold: whilst there will be no problem for most of the time, particular words will sound wrong. For example, in the word "Stephen" the S and half of the t will be missing, such that the word cannot be understood at all. To combat this problem, some systems employing silence detection [e.g. Gruber 83a, Joyce 83] send the samples that were produced just before the start of a talkspurt was detected. Both of the systems mentioned send 10ms of such samples. If this is necessary for satisfactory intelligibility of speech then it provides another reason why silence suppression is undesirable for use in a LAN, since the procedure clearly adds another 10ms of delay to the link.[2]

Several studies of TASI effects [e.g. Jankowski 76] note, however, that 15ms of clipping at the front of a talkspurt is imperceptible anyhow. Sending samples before the time of the energy threshold seems to be a way partially to avoid another effect, that weak fricatives do not have enough energy to trigger a simple silence detector. [Gan 86] shows that these weak fricatives can be detected reliably by triggering on zero crossing frequency as well as energy level; this is a more reliable approach. It increases the complexity of the silence detector, however: a silence detector functioning in this manner might best not be located in every phone.

3.2 Gateways for Integrated Traffic

The UNIVERSE voice experiments tested the idea that a single protocol could be built which would work well in a variety of different connection environments. The results, particularly the behaviour of the satellite connections suggest that such a protocol is neither easy to build nor desirable. To a small extent problems with the satellite connections were due to the protocol itself: we will consider this later. We will discuss first the more significant problems which lay within the network.

3.2.1 Classes of Service

The UNIVERSE satellite bridges did not handle speech well, simply because they were designed for data rather than for Integrated Services traffic. The total delay normally encountered on a voice link was much in excess of that due to signal propagation to and from the satellite, because the bandwidth allocation mechanism was not suited to voice. The satellite used was a broadcast type employing dynamic allocation of bandwidth 'windows' on request. When a phone

[2] In the ISLAND voice protocol that will be outlined in chapter 5 the delay time compensating for network jitter could however be subtracted from this figure.

resumed sending packets after a period of silence, the satellite generally did not have bandwidth immediately available (since window allocations would be deleted after a certain period of disuse) so that a request for such a window would need to be sent (across the satellite link to one of the broadcasting stations, which would be acting as 'master') before transmission could begin.

In fairness to the designers of the UNIVERSE satellite bridge it should be remarked that these shortcomings for Integrated Services traffic were anticipated at the time of the design. There was a lack of hardware suitable for a more advanced specification. As a result of experience gained from the UNIVERSE experiments, a satellite bridge of much more advanced specification and greater suitability for Integrated Services has since been designed and built [Benito 84].

The important point learnt was that gateways designed for data are unlikely to provide good performance for Integrated Services unless some prioritisation mechanism is provided as support for voice traffic. A gateway handling both packet voice and data traffic is likely to produce satisfactory service for both only if it is designed to

(a) distinguish the two types of traffic

(b) handle each type in a manner suited to its requirements.

In other words a gateway needs to provide the same guarantees for voice and data traffic mixtures as a hybrid LAN does. The very different natures of voice and data traffic have been discussed. A load on a mixed traffic gateway can be expected to consist of a moderate and bounded number of usually active voice channels plus a potentially large number of usually inactive data channels. An Integrated Services gateway (or its associated route section) needs to support a specification, very much like that of a hybrid LAN, along the lines that *for up to N voice channels, P% of packets will be delivered within D milliseconds delay* and also that *at any instant, the part of the available bandwidth B not required to support the instantaneous voice load is available to deliver data packets using some queueing strategy S.*

For a satellite link, there is a further reason for separating voice and data traffic. Typically the raw error rate of a satellite channel will be much better than that required for voice. However data traffic requires error-free delivery. The number of packet retries needed to send a datagram may substantially be reduced (thus speeding the datagram delivery and possibly consuming less actual satellite bandwidth) if data transmission uses redundant coding and forward error correction to produce better error rates than those of the raw channel. This suggests that it may be useful to provide a number of low capacity channels with low latency and poor error rate plus rather more channels with high peak instantaneous capacity and low error rate, the latter resorting to some queueing mechanism in the case of high offered loads. By approaching gateway design in this manner, it becomes possible to offer voice channels with delay characteristics expected of circuit switched systems, but with a channel utilisation efficiency expected from packet switched systems.

3.2.2 Link Specification and Monitoring

A key problem that must be addressed in the design of an Integrated Services network is that of specifying and then monitoring the performance of each link in the network. In particular, for voice traffic connections there will be a specification of some total delay budget and of a percentage of time for which the connection must be within budget. Both specifications and monitoring may be of either of two types—*end to end* or *hop by hop.*

The approach throughout the UNIVERSE experiments was end to end. The bridges played no active part in the protocol and monitoring facilities were provided only in the telephones, as described in chapter 2. The lack of intermediate monitoring was a major problem. End-only monitoring was adequate to verify that certain links performed satisfactorily, but when end to end performance became degraded unacceptably the absence of status information from each component meant that there was insufficient information to determine the cause of the problem. Problems with the satellite bridge were very difficult to investigate for this reason. Status information collected at each node need not have been very complex and need not have had an adverse effect on the performance of the nodes being measured. The simple status dumps produced in the phone for use by a remote debugging system were adequate and hardly affected the phone's performance. However some very primitive behavioural information is essential for realisation of a mixed media link.

In a hop by hop approach to specifying and monitoring an integrated network, suppose a connection were to consist of two LAN links joined by a WAN. The delay specification would consist of a figure for each LAN and a figure for the WAN. Each component of a complex network would have been designed and tested to a specification. Before a call is set up along any route in the network the performance of the complete connection can therefore be calculated. A connection not meeting the end to end performance requirements would not be set up. In longer term network planning, if certain routes do not meet end to end requirements but are deemed useful, a decision may be made to select a single component and upgrade its performance. The new figure for the improved section can then be used in complete link performance calculations.

In contrast, if monitoring is performed end to end, each possible combination of route segments that can be used will need to be tested to ensure that it meets the end to end requirements. When a link is commissioned or improved, such that new connection paths become possible, all such possible connections will have to be measured. These measurements will be okay if all we wish to know is which combinations satisfy our criteria and which fail to meet them, but we are more likely to want to know which link segments account for any particular failure, i.e. how a delay budget is being absorbed. The answers to such questions will be found only by using hop by hop methods.

The above arguments advocate hop by hop monitoring and hop by hop budget calculations, but do not imply that the voice protocol itself need operate in a hop by hop manner. For a protocol, the distinction to be drawn between end to end and hop by hop methodology is between the use of a gateway merely to relay packets flowing in accordance with a protocol operating between the initial transmitter and eventual receiver of the voice (although the gateway may handle the packets according to a 'class of service' specification) and a gateway which understands the details of the protocol that it handles and which may perform protocol translation if it forms the boundary between two different environments.[3]

[3] The hop by hop approach is equivalent to Cheriton's notion [Cheriton 86] of 'Transport Level Gateways' for data traffic, in that each link takes responsibility for its own operating environment.

How do these two approaches differ? Consider two voice streams flowing along the same route through a network, subject to the same overall delay, but one stream using each of the protocol strategies we are discussing. Suppose that a single component of the hop by hop connection is temporarily not meeting its delay specification. If that component is producing gaps in the voice stream then the whole system will be violating the specification. In the end to end case, there is a probability that another link is sufficiently within its specification at that instant that the whole connection will produce gap-free speech. This is another way of stating Naylor and Kleinrock's observation (from mathematical modelling) [Naylor 82] that the buffering needed at the end of **n** identical links, in an end to end strategy, is less than **n** times the buffering needed at each link in a hop by hop system, if both must give the same performance. If component **n** of the link performs such that voice traffic will be delayed less than D_n except for P_n% of the time, then to a first approximation the hop by hop system will be outside its specification for the sum of all P_n%s of the time, whereas the other protocol style will do better.

3.2.3 Minimal Protocols for each Link

From the above discussion, it seems that better performance will be obtained from a series of links by using an end to end protocol. However there are reasons why we may wish to operate a hop by hop system. We may wish to operate different protocols for different transmission environments. It was suggested above that silence suppression was not useful for local links but might be justified across a satellite link. This implies a protocol translation and may be desirable on economic grounds. (Provision of channels with different error rates for voice and data on a satellite link requires that the bridge have the ability to distinguish different traffic types, but this is a 'class of service' mechanism rather than a protocol translation issue.)

On grounds of economics it is worth considering the use of a gateway employing protocol translation in order to minimise the protocol complexity on each link, to the simplest that its environment will stand. For example on a LAN connection between two phones, the transmission environment (at least on a slotted ring, as we will justify below) is not at all severe; the simplest of protocols will be adequate. This simplicity can be exploited to make the telephones cheap, simple objects. The phone-to-phone protocol must also have very small delay characteristics. On a wider area network connection, cost of implementing a protocol will be much less important (given all the 'class of service' mechanisms which will also have to be supported). The available delay budget will be greater and use can be made of a more heavyweight protocol to make the link function satisfactorily in more adverse conditions.

The use of different protocols on wide and local area connections may actually make some configurations more viable, despite Naylor and Kleinrock's observations. Consider three people involved in an audio conference, two on the same LAN as a conference bridge and the third across a WAN link from the other two. Suppose that all three phones' outputs are sent to the bridge, which then sends back to each participant the sum of the other two signals. If the connection is of hop by hop style then the greatest delay on the link will be twice the local link delay plus one wide area delay. Because the local link protocol can be made very lightweight on account of the easy environment in which it has to work, the local link delay may be very small. In the end to end case the protocol will have to be heavyweight enough to work over all links: this will almost certainly imply that it introduces more delay when used over a LAN connection than the hop by hop LAN protocol would. The delays over two end to end connections (remote phone to bridge and local phone to bridge) when added together may well be greater than the two local plus one wide area link delays in the hop by hop system.

This is not simply a contrived example to show a case where hop by hop protocols could be better. It has already been said that in the design of the ISLAND it may be sensible to send voice streams three times around one LAN in order to benefit from the cost savings of partitioning voice functions into a series of single function servers. For this to be viable we will need an environment where a local LAN hop introduces very little delay.

3.3 Delay

The UNIVERSE work took very little account of the delay requirements that a telephone engineer might place on a network. However the packet size was varied to test how people reacted to various delays. (The packet size clearly contributes directly to the delay on a link: if the samples in a packet last a time t then packetisation introduces a delay of t into the link.) The phones were of a four wire type and the acoustic coupling between ends of a handset was very small, so that only delays rather than echo problems were observable. (There were no loudspeaking telephones in the system.)

The delay which can be tolerated on a link is not a simple issue. In the absence of echo, there is a length of 'round trip' delay below which the parties in a conversation will not be bothered. We found this to be about 125ms.[4] Above that level they will begin to find themselves repeatedly backing off because the other one has begun to speak, then both starting again and so on.

The interaction between echos and delays is more complex [Hills 79]. For delays of more than 50ms, the attenuation required in an echo path for the echo to be tolerable to the speaker rises steadily with increasing delay [Gruber 83b]. For delays of less than 50ms the speaker will not recognise the echo for what it is. It may be confused with sidetone,[5] although there have also been cases where speakers have complained of a crossed line when what they could actually hear was an echo of their own voice. Echos delayed by around 30-40ms can unnerve the speaker and induce speech impediments.[6]

For this reason the maximum round trip delay allowed in 2/4-wire networks before echo cancellers are installed is 24ms (at least in the U.K.). The delay that any local phone system can be allowed to introduce if it is to interwork satisfactorily with a national telephone network will depend on the deployment of echo cancellers within the national network. Until recently, in the U.K. the allowed delay for phone links within a customer's premises was 2.5ms. This figure was set according to the capabilities of conventional PABX technologies and made it very difficult to interwork from packet systems to the national network. Recently however the figures have been altered (a) in recognition that new techniques such as packet switching are under investigation

[4] Others have found similar results: a typical working value used by PTTs is 100ms. PTTs' figures are generally expressed in terms of 'percentage customer objections'.

[5] Sidetone is the term used in analogue phone systems for very local echos (e.g. in the phone instrument itself).

[6] U.K. Industrial Tribunals use the following test for deafness: a person is given headphones in which can be heard his/her voice, amplified and delayed by around 40ms. He or she then reads some passage of text. Only a deaf person can avoid stuttering under these conditions.

and near to introduction and (b) as a result of deregulation of some aspects of U.K. telecommunications. The latter has meant that there now need to be specifications for a 'private network' rather than for a PABX. The delay allowed looking into and back out of a private network is 10ms [Oftel 86].

3.3.1 Strict and Delayed Time Protocols

For a simple self-contained voice system like the UNIVERSE network, large and varying delays may be acceptable, but if a packet network is to be interfaced to PTTs' networks then the delays that it produces must be not only short (for a local area system in particular) but also well bounded. Since the ISLAND should be able to interwork with public networks, this suggests that strict time protocols may be more attractive for ISLAND use. If the delayed time approach were used it would need to be strictly bounded. Given that those bounds are fairly tight then it would be better to use a fixed timing system which employs the maximum permitted buffering in the receiver at all times, so as to minimise the proportion of samples which arrive too late to be of any use. The protocol will also be simpler, no longer having to decrease and increase the level of buffering in response to network loading.

Delayed timing is really only useful in the case of networks which are not designed for integrated traffic but are primarily intended for data traffic. Even then, as remarked above, delayed timing will not help in the situation where instead of jitter distributions varying with time the true problem is periods during which there is inadequate bandwidth. (There are schemes designed to cope with inadequate bandwidth, where for example at times of high load the least significant bit(s) of PCM samples are not carried by the network and are reconstructed as fixed values by the receiver. However such schemes require either the transmitter to react to network loading information or the network both to identify voice packets and to understand their internal formats.)

There is one further problem with delayed timing protocols, that of reaction time of the delay mechanism. Consider the arrival of a single abnormally delayed packet. Assuming that this packet is short and subsequent packets are not abnormally delayed, we would rather it did not pull back the timing. The timing should be pulled back if the condition persists however. On the other hand if the condition persists, the whole point of pulling back the timing starting with the first packet of a series to be received late is that there will only be one interruption to the speech. If we wait until the condition persists we will have produced a greater interruption. There is clearly no right answer to this problem: for UNIVERSE it was assumed that individual packets arriving late nearly always presaged a general retardation.

The correct frequency at which to run the dynamic adjustment routine is also not obvious. We found that experience of operating on each type of connection enabled suitable values to be selected. An engineer building this type of network for service use would probably need to gather and analyse a considerable amount of information about the traffic patterns to be expected.

3.3.2 Packet Sizes and Processor Use

Although the packet size used in the UNIVERSE network was adjustable, it could only be adjusted between a maximum packet duration of 240ms and a minimum of 16ms. The minimum was set by the processing power of the Z80 microprocessor which ran the phone protocol. Protocol execution could be divided for accounting purposes into a per-sample component and a per-packet component. Every 125μs, samples had to be pulled in and out of the CODEC. Since the Cambridge Ring carries two data bytes per slot, the tight loops within the mechanisms to transmit and receive basic blocks also needed to handle a mid-basic-block slot every 250μs. The per-packet overhead was due first to the protocol handling (sequence count incrementing, deficit/reserve accounting, silence flag examination etc.) and also due to the Ring basic block handling mechanisms, which have a checking overhead for every block transmitted and received.

The simple status dumping information added a little to the per-packet overhead, but not a significant amount. The most expensive component of the protocol was the simple interrupt service routine to handle the CODEC: in the middle of packet assembly this consumed 62μs out of every 125μs. This was the performance of a 6MHz Z80 processor: coding the same functions on an 8MHz 68000 or a 10MHz 32016 processor would reduce the time to almost exactly half.

The use of a network with a larger slot size than the two bytes carried in the Cambridge Ring would have reduced the minimum packet size that the Z80 could have handled. Nevertheless for it to be possible to produce a phone, based on a simple microprocessor, which can handle packets of somewhat less than 10ms in length in order to satisfy normal PTT requirements, a very simple protocol definition and finely tuned software structure are called for.

A mass-produced packet phone would likely be based upon special VLSI rather than on software running on a general-purpose microprocessor as in the experimental implementation of telephones for ISLAND. We would still want to use the simplest possible protocol, to reduce the complexity and design cost of such special purpose hardware. There is another place where the efficient handling of voice streams by a microprocessor will be of concern, regardless of whether the phone is microprocessor-based—in the servers which were mentioned in chapter 1 as implementing various voice functions. Suppose that these servers run some high level operating system. There are fairly large per-packet overheads for the handling of network i/o in most operating systems. Special design work will be needed for a general-purpose operating system to be able to handle voice streams as well as high level data structures. These various points concerning performance will be developed in the next four chapters of this dissertation.

3.4 Choice of a Network

It is now appropriate to discuss the specific local area network used in the design of ISLAND. Most of the special properties of voice traffic have now been mentioned; these clearly need to be appreciated when chosing a network.

The network used as the basis for the ISLAND design is the Cambridge Fast Ring. It will not surprise the reader to be told that it was not the desire to build an Integrated Services system that led to selection of the Cambridge Fast Ring as vehicle. Rather the design of the Fast Ring led to consideration of what applications it would be well suited to. This and the belief that ideas from the Cambridge Distributed Computing System would be useful for telecommunications applications provided the impetus for the ISLAND project. Notwithstanding this, discussion of the suitability of the Fast Ring for Integrated Services is important to this thesis. This section will consider first slotted rings in general and then the Fast Ring in particular.

3.4.1 Slotted Rings

The principle of the slotted ring is fairly simple. A fixed-format train of slots circulates indefinitely and transmission is achieved by waiting for an empty slot; inserting into it destination address and data and awaiting its return with a marker indicating whether the data has been accepted by the destination. On return the slot is marked empty and passed on.

The properties of slotted rings have been discussed extensively in the literature [Temple 84]. Only those which affect simultaneous transmission of voice and data will be mentioned here. A slotted ring provides automatic division of available bandwidth between contenders. This is simply because a station must pass a slot on after each use; therefore if **n** stations all wish to transmit they will each obtain a slot within **n**+1 slot revolutions (in the simplest case of a single slot ring). This is good for voice, at the cost of reducing the maximum available point-to-point data rate, which is less good for data applications. If the ring is run faster this will increase the maximum point-to-point rate available after sharing. Data traffic handling will therefore be improved, without this making the performance for voice or other synchronous traffic less good. For a simple slotted ring, these higher point-to-point rates require longer slots [Falconer 85], since the higher speed just 'makes the bits shorter' rather than affecting their transit time. In other words, since a station must wait for a complete revolution of 'its' slot before trying to transmit again, to obtain improved point-to-point rates as the total bandwidth is increased the amount of data that can be transmitted by a single transmitter per revolution (i.e. the slot length) must also be increased.

Higher point-to-point bandwidth, as well as an improvement in overall bandwidth utilisation, can alternatively be obtained from a slotted ring by using for example a 'destination delete' strategy [Adams 84] (in which the slot is marked empty after passing the destination rather than after returning to the sender). However the simple slotted ring offers equal division of bandwidth between contenders without recourse to complex additional hardware. This is not true of the destination delete system.

3.4.2 The Cambridge Fast Ring

The Cambridge Fast Ring [Hopper 86] is a VLSI slotted ring which at the time of writing runs reliably at 50Mbs^{-1}. Each slot accommodates 32 bytes of data and each node on a Ring has a 16 bit address. Most of the complexity of the Ring is contained within a single CMOS IC, which can function as a station, a bridge (placing two ICs back to back on a pair of Rings) or the 'monitor station' which sets up and maintains the slot structure of a Ring. Which of these three functions a single IC performs is set by a voltage level on a single pin. The 'bridge' mode of the IC is designed to support a mesh of linked Rings, with a flat 16 bit address space across the whole mesh.

The characteristics of slotted rings of interest to integrated traffic were discussed above. We will state them again more specifically for the Fast Ring in terms of the three following properties:

i) where **n** stations all request maximum available bandwidth, they will each obtain an **n**th share of the total available. This is not an unusual property for LANs, but there are some LANs where the network utilisation falls as the offered load increases. This is not the case here.

ii) this bandwidth sharing occurs at a fine granularity. The unit of sharing is the slot, carrying 32 data bytes. On a Cambridge Ring, it is perfectly normal for some stations to be transmitting very small blocks and others very large ones. As will be discussed further in following chapters, this is very well suited to a voice/data mixture. On many LANs, for example CSMA and token passing types, the use of very small data packets will drastically reduce the total available network bandwidth, simply because the 'guard bands' between transmissions will dominate the actual transmission times. Worse still, the actual available bandwidth will remain roughly constant even if the data rate of the network is raised, so long as the packet size remains small. This is simply because the guard band time is a function of the physical size of the network and not of the data rate. In order to keep the usable proportion of bandwidth of a CSMA network constant as the raw data rate is increased, the packet size must be increased proportionally [Tobagi 83]. It was noted above that packet size must be increased on a slotted ring also in order to produce an increase in the point-to-point rate as network data rate is increased, but

a) this does not apply to *total* network bandwidth availability

b) the packet sizes in a slotted ring are very much shorter than those for example for an Ethernet of comparable data rate.

iii) where **p** users request small amounts of bandwidth and **q** users request the maximum available, the **p** users' requests will be satisfied within a very short time, whilst the **q** users will share the bandwidth actually remaining after the **p** users' needs have been satisfied. This clearly suits the voice/data mix: the **p** users are telephones which require bandwidth with a very short access delay, but whose bandwidth requirements are small compared with the network bandwidth. On a 50Mbs^{-1} Fast Ring of length 1km with 10 nodes, the access delay cannot exceed 100μs. (For a 20 or 50 node ring of the same length the access times will typically be 100μs and 300μs respectively.) The q users are bursty data traffic. At any time a few transmitters will request as much bandwidth as possible. They will obtain a share of all bandwidth which is surplus to the phones' requirements at that instant: unlike the circuit switched case, the available bandwidth will be that not *actually* in use by the phones. If we chose to suppress silence transmission, then during silent periods the bandwidth not used by a voice connection can be made available for data transmission, since the Ring will allow immediate pre-emption by the voice traffic as soon as a voice stream needs bandwidth.

All of these properties can be achieved using hybrid networks, but the complexity of a hybrid system is rather greater than that of a Fast Ring. The Fast Ring does have limitations compared with some of the hybrids; it will not work in all of the domains in which for example MAGNET would excel.[7] The simple restriction is on the total bandwidth that can be guaranteed to a single station. If there are n stations all transmitting at one time, then no single station can be guaranteed more than an (n+1)th of the available bandwidth. For a $50Mbs^{-1}$ ring, allowing for the overhead of address space etc., this limits 50 transmitters to about $820kbs^{-1}$ each, or to 13 standard telephony connections per station. A study of the use of rings for telephony applications [Falconer 85] reported the Cambridge Ring as an unsuitable candidate. However the usage envisaged in that study was for switching of highly concentrated traffic entering the ring at individual nodes. (It also considered the older $10Mbs^{-1}$ Cambridge Ring.) The above example shows that adequate performance cannot be guaranteed for more than 13 multiplexed conversations per transmitting node. However ISLAND is mostly concerned with both low network utilisation conditions and individual telephones which only require $64kbs^{-1}$ of transmission bandwidth. For these the ring is clearly satisfactory. The limit on bandwidth per node becomes important in the cases of certain components of our system (e.g. the one which provides audio conference facilities) which need to launch several voice streams simultaneously. Certain video applications may also be a concern.

3.4.3 Use of Other Networks

As was remarked in section 1.6.3, networks other than the Fast Ring were used for much of the work described in this dissertation. It is appropriate to mention here how far they differ from the behaviour outlined above.

The work set out in chapter 8 was done in the Etherphone environment, which is built on the Ethernet. As noted above, contention networks offer poor performance when used to transport very small packets, but small packets are important for small delays on voice connections. Where several voice streams are being carried on one contention network simultaneously, the performance degradation will be worsened further by another factor. Voice streams produce packets at regular intervals and several voice streams can interact to produce regular collision/backoff patterns, thus frustrating the collision control algorithms [DeTreville 83]. None of this invalidates either the Etherphone project or the work described in chapter 8, both of which address issues at levels above the network, and for which the Ethernet is a perfectly satisfactory network base.

Implementation of the voice provider in phones and servers, as described in chapters 5 and 6, was performed using the $10Mbs^{-1}$ Cambridge Ring. All the comments above about bandwidth sharing and delay guarantees apply equally to the Fast Ring and to the older $10Mbs^{-1}$ ring. The differences between the two Ring types of relevance for ISLAND use are

a) the lower bandwidth which can be guaranteed to a single station on a $10Mbs^{-1}$ Ring. This alters the number of stations which can be allowed on a single Ring, for any given level of bandwidth guarantee. However if it is only required that each station can be guaranteed $64kbs^{-1}$ transmission bandwidth, for simple phone services, the older Ring will still support more than 60 stations, which is adequate for experimental purposes (and more than ever known on a single

[7] As outlined in section 1.5, MAGNET allows for a variable amount of bandwidth with a guaranteed response time to be shared amongst some subset of all transmitters. This is accomplished using two classes of service. The division of bandwidth between them is configurable, so that for example two high speed video transmitters could be assured of sufficient bandwidth if they were the only members of the prioritised class.

Ring in Cambridge!).

b) the smaller packet size of the older Ring (2 rather than 32 bytes). This has two implications. First almost no protocol can be implemented in two bytes, which is why the 'basic block protocol' [JNT 82] was designed, in which a train of slots is formed into a long data block containing header, sub-address, data and checksum. For reasons which will be discussed in chapter 7, the older Ring does not allow multiplexing of block receptions. This means that a translator, as described in chapter 4, could only be made to handle voice streams in one direction at a time. It is also half of the reason why only the compounding routines and not the network interface of the conference server are described in chapter 6, as the older Ring would not have supported the latter. The other half of the reason is efficiency: the larger packet size of the Fast Ring means that better performance can be obtained from the Ring using interface hardware and software of the same complexity, which means that the capacity of voice servers will be increased. All the techniques which have been used and which are described in later chapters for construction of servers and the software that they run are equally valid for both types of Ring. The designs of server architecture and voice protocol were in fact optimised for the Fast Ring, but give adequate performance in the implementations currently existing on the older Ring. What clearly could not be demonstrated was scaling of the ISLAND design to exploit the greater capacities of servers running on the Fast Ring.

4. A Design for PABX Functions

This chapter discusses a method for obtaining and expanding the functionality of a PABX, using a LAN-based system of voice servers and simple phones. The first issue addressed is the design choice as to where the capabilities should be supported. The important design decision which was taken on this matter was to minimise the complexity of the phones themselves. Their capabilities and how they should be controlled are both discussed. The next part of the chapter looks at servers to support the features of the system, in particular voice message storage and conference facilities. The discussion is generalised to how the system should facilitate the introduction of new services, with connection to wide area systems used as an example. Finally the design of a control system to handle the various facilities available is outlined.

4.1 The Telephones

4.1.1 A Minimalist Approach

The first decision to be made in partitioning the implementation of PABX facilities in the ISLAND system is "where is the work to be done?"—i.e. where will the complexity of the system lie? The primary rule which will be proposed here is that as far as possible functionality should not be provided in the phones themselves.

A possible alternative approach to phone design would be a microprocessor-based telephone instrument able to

look up the name or number of a desired station to find its network address

form speech samples into short blocks if they are to be sent to another phone or long blocks if they are to be sent to a fileserver

perform the compounding functions associated with conference calls.

To many designers, for example from distributed computing backgrounds, this seems exactly the right level of functionality. (It is roughly the level of complexity of the telephones used in the Etherphone project for example.) It is therefore important to discuss in some detail the reasons for rejecting this approach.

Such a scheme aims for resilience by putting as much of the intelligence as possible into the phone, relying as little as possible on external or centralised agents. The problem is that it is not easy to predict just what functions a long-lived system will develop. It is therefore undesirable to constrain them by software fixed rigidly in the phone. There is a tendency to design PABXs with some of their features implemented within the actual phone instruments.[1] Not only does this mean that full interworking with standard telephone sets is impossible, but also the next generation of PABX features will require upgrades to all the current 'new' telephone instruments. The complexity of telephones normally supplied by PTTs to small business customers to provide particular management functions is considerable: these phones are entirely inextensible and therefore either bind a customer to a particular set of facilities or have a very short installed life.

[1] for example see [Swart 86]: whilst the declared aim of that work is to use the ISDN interface as a means of standardising and simplifying the phone, the functionality required within AT&T's 'multi-button feature phone' is still considerable.

Of course much of this problem can be solved by making the phone software down-loadable. However such an approach implies that there are enough computing resources—processing, memory and i/o hardware—in each phone to run any program that may be loaded for the foreseeable future and also to support a reliable loading protocol. If this be true, then these resources will make the phones more expensive—maybe not by much, but this increment, even if small, will be multiplied by a large number of phones. (On a more frivolous level, in a down-loaded system, one can imagine a TV announcement suggesting that all citizens reboot their phones because a bug-laden version has been down-loaded. However, put more seriously, a down-loadable system for use by the general public is a non-trivial design exercise.) The main reasons for preferring a modest, almost minimal, degree of functionality in the phone are then that

i) if the phone is carefully designed it should, with the aid of external services, be adequate for providing all likely user functions, upgrades being accomplished by enhancing the external parts of the system rather than the much more numerous phones.

ii) if functionality is kept out of the phone it can be made both cheap and reliable—vital for such a ubiquitous component.

This view not only differs significantly from the expected position of many distributed computing advocates, but also minimises the functionality of the phone below that now regarded as normal by the PTTs. In some respects, the functionality which will be outlined below for the phone is not unlike that once provided by the PTTs, but there is a tendency for new phones (even for domestic subscribers' use) to for example be able to retain ten favourite numbers accessible by short codes. Such functionality does not, in our view, belong in the phone.

The approach championed here has some precedent in the distributed computing world. In the Cambridge Distributed Computing System [Needham 82], nameservers are provided to transform names of services into suitable addresses. The only address that is ever bound into a machine or its software is that of the local nameserver. As in the UNIVERSE network, the nameservers handle routing across multiple networks for their clients by setting up paths through to the eventual destinations and then handing to the clients the address of and sub-address in the first gateway or bridge on the route.

This has the effect of minimising the complexity (and hence cost) of the smallest object which can be placed on the network as a first-class citizen. For example the cost of connecting a single terminal to the network can be reduced. There is clear parallelism between this approach and the attitude to phone functionality presented here.

4.1.2 The Capabilities of a Phone

The design aim of keeping unnecessary functionality out of the phones leads to the inclusion of only very restricted capabilities. The phone is to be capable of observing that its handset has been raised, of broadcasting attention calls to higher authority and of acting upon received instructions to do one of a very few simple actions. When, for example, digits are dialled on the phone's keypad, it simply relays these to the higher authority. Like a traditional phone, it knows nothing of the significance of the numbers dialled. It merely passes the digits on until it receives an instruction of the form "talk to x and listen to y". x and y are simple addresses; they may be equal to one another or null. It is not even necessary for phones to interact with the general apparatus of the distributed system, such as nameservers and other low-level utilities: this is important, being in accordance with the general view that as little knowledge as possible about the external system should be bound into the phone implementation. Once the phone has been instructed to direct voice at and receive voice from some other components of the network, it will assemble voice samples into small blocks and transmit them as soon as assembled. It has already been stated that a very lightweight phone protocol should be used for LAN connections, and that the non-hostile environment of the Cambridge Fast Ring for integrated services makes a simple protocol possible. Details of the chosen LAN voice protocol will be described in the next chapter: it will suffice here to remark that being lightweight it requires very little processing power in the phone.

4.1.3 Control of Telephones

It has been assumed in the above section that some external agent is in control of the functionality of the system—the phone sends a series of keystrokes out to a higher authority and then obtains commands of the form 'talk to *x*, listen to *y*' as if from God. Roughly for this reason, the ISLAND telephone controller has become known as the General Operations Director (or GOD for short). The detailed design and implementation of GOD fall outside the scope of this thesis, but the way in which GOD must interface to phones and servers is of relevance. It is clearly in GOD's domain (at least as seen by the phones) to provide a directory service, i.e a mapping of dialled numbers (or their equivalents) into commands to connect a phone to another phone or to some server. GOD, as remarked before, is not part of the switching process for voice samples: that function is implicit in the Ring. GOD is only required to act when the values of *x* and *y*—the addresses with which the phone is currently exchanging voice streams—are changed.

4.2 Voice Recording and Playback

4.2.1 The Storage Medium

In order to provide recording and playback facilities for voice there are several design issues which need to be considered. The most obvious is the storage medium, or rather storage method. There is a choice between digital and analogue means. For a medium such as magnetic tape, more voice can be stored per inch by analogue means than digitally, using comparable technology. In this respect, analogue storage seems more attractive. It can come as a shock to realise just how little voice can be stored on an average Winchester disc in digital form. 30Mbytes of storage represents only roughly an hour's worth of standard PCM-encoded voice and this is with an audio bandwidth very much worse than that available from a domestic audio cassette.

A major advantage of digital storage lies in uniformity of document representation. Later chapters will discuss the idea of a document architecture including voice and images. This discussion will highlight the advantages of being able to access a mixed-media document as one entity and transport it simply as a 'bag of bits'. Various techniques will also be discussed which can be used to minimise the space occupied by a series of voice files, particularly if some were created as the result of editing others. Such schemes only work well if there is almost immediate random access to any of the stored voice.

The availability of random access most clearly makes digital storage the right choice for voice. Suppose that our system allows the placing of voice in a document (for example with markers visible in parts of the document and a button marked 'press to playback'). The user will expect the sound to be available immediately. It is not satisfactory to have to wait while a long tape rewinds, particularly if (as will be suggested later) voice is to be used for spontaneous and off-the-cuff annotations of text. The technology for recording analogue signals in an equivalent manner to which digital information is held on a disc does not exist. In principle it could be developed, but this is unlikely in a world where new types of domestic audio system are all digital. Quite apart from preventing exchange of electronic multi-media documents across a digital network, it would also be an astonishing choice for a system which avoids the hybrid use of digital and analogue techniques in so many other respects.

4.2.2 File Access and Locking

Having decided upon digital storage for voice, one might therefore envisage creating a 'voice fileserver'. In the same way that in distributed systems personal computers interact with a single server which stores a large number of data files, the phones might interface to a server which stores large numbers of voice messages. There are ways in which the properties of voice and data fileservers need to be very different. Data filing systems usually contain protection schemes. On the one hand there will be access locks for example to prevent client **a** writing to a file or deleting it whilst client **b** has rights to read it; on the other hand protocols such as 'two phase commit' [Gray 78] to ensure that filing transactions are atomic, so that if a machine fails during an operation data files cannot be left in inconsistent states or be lost altogether. This is very important as a filing system which occasionally lost the odd four kilobytes of text of a program or of a dissertation would not find many contented users. Such an amount of text might represent a considerable investment in programming time.

In contrast the loss of four kilobytes of PCM voice during recording is rather less serious, representing only a half second of sound. If the half second is part of a simple voice message, it is very easy to re-record any sound lost. Only a minimum of protection against inconsistencies in the filing system need be provided, like that described below for the Etherphone filing system. On the other hand, as will be discussed in later chapters, it matters very much how soon after recording it a voice file can be played back. It also matters that voice can be recovered from a fileserver smoothly—at a constant bandwidth and without large gaps in delivery. There is always a performance penalty associated with the locking mechanisms in data filing systems, which will make these requirements hard to satisfy. Since there is no need to build such a heavyweight scheme for voice, it can be expected that acceptable results may be obtained from a voice fileserver by setting the performance/integrity tradeoff differently from the usual point for data fileservers.

4.2.3 Voice Packet Sizes

It has already been said that small voice packets are appropriate for phone-to-phone links on a LAN: the small packets introduce only small delay and Cambridge Rings are well suited to transmitting small packets. However handling streams of very small packets will be a problem for a fileserver. The interface between the fileserver and the Ring will probably consist of software containing a fairly high 'per packet' overhead for i/o handling. Therefore if the fileserver has to transmit and receive small packets for several phones at once (which will be required both on grounds of cost and because a user wishing to retrieve a voice message from an overloaded fileserver cannot simply go and get it from another!), its performance is likely to fall off quite dramatically.

4.2.4 The Etherphone Voice Fileserver

Given these problems of design of a voice fileserver, it is worth examining the approach to its design taken on another project. The Etherphone system is built around microprocessor-based telephones, a series of personal computers each with their own local disc for data file storage, and a set of fileservers on which all valuable files are normally kept. Use of the local disc for voice storage was rejected because it involves unnecessary copying of voice from workstation to workstation whenever voice is to be replayed. (In the case of a voice message the person who records it rarely listens to it: the recipient however does [Gould 83, Nicholson 85]. Users' Etherphone telephone units are physically separate from their workstations, so that the voice might just as well be sent to a common fileserver, immediately it is generated, as to the workstation.) However the global fileservers are totally unsuited to voice streams: they can neither produce the turn-around nor support the real time performance required for voice. The solution adopted was therefore to build a special filing system on a dedicated server. This consists of a very large file (as understood by the normal filing system) which is operated upon by a voice fileserver program. The voice fileserver uses the large file as a complete filing system, allocating various logical voice files out of it. The underlying data filing system sees only that the voice fileserver program always holds an exclusive lock on the large file. The voice fileserver allows its clients very free and lightweight access to the voice files: for example one writer and any number of readers are allowed to operate on the same file simultaneously. Simple protection is provided to make the reading of a non-existent portion of a voice file produce nothing worse than silence. The filing system is not fool-proof or atomic: server crashes may leave a file with some amount of data not written through to disc. Crashes can also leave disc blocks neither allocated nor free: a scavenger program is provided to straighten out the server in such cases.

The problems of small blocks are avoided in two ways in this fileserver. First of all the Etherphones send longer voice packets to one another than would please a PTT; secondly the phone-to-phone protocol is a little different to the phone-to-fileserver protocol, the latter being more suited to this particular interaction. The voice fileserver runs on a 'Dorado' [Lampson 81]. The Dorado is more than twice as powerful as a Vax 11-780 and has network handling software written mainly in microcode. The performance problems for small frequent blocks are therefore not nearly so serious as for a small microprocessor-based fileserver.

4.2.5 A 'Front-End' Approach

The Etherphone system uses a separate type of filing system to cope with both real-time access to files and locking problems. The simplicity of this low level filing system eases the problem of transferring fairly small blocks. The voice filing system is only accessible across a LAN using one simple protocol (as opposed to supporting a variety of styles of access for general resident applications) which enables this access to be made specially efficient. However we believe that the same fileserver should be used for both data and voice storage: we will justify this proposition in later chapters when discussing the way in which integrated documents, containing for example voice and data, are manipulated.

The problems of small block sizes could be avoided by running a different protocol in the phone for talking to filing systems from that used for talking to other phones, as the Etherphone system does. Whereas the packet size needs to be minimised for phone-to-phone connections, recording and playback of stored messages is not a real time process and hence longer packet sizes are a possibility. However the requirement to handle two different protocols increases the complexity of our phone, something which we wish to avoid. There is also much to be gained from rigid refusal to treat different voice connections in different ways, as will shortly be discussed. The ISLAND solution to this problem is called the *translator*.[2] This is a server which can be told to receive streams of short packets from phones, block them up into longer packets and deliver them to the fileserver. The reverse function is also supported. This of itself simply relieves the load on the fileserver's LAN interface. It does not remove the time-critical requirements upon the fileserver for storing voice blocks or retrieving them at regular intervals. However the translator can provide sufficient buffering for each voice channel in use that these intervals can be increased and the time-critical requirements upon the fileserver can be relaxed to an acceptable level.

4.2.6 Fileserver Primitives

The design of the translator as a front end to a conventional fileserver removes the small packet problem and relieves the real time access requirement. It does not cope with the issue of lightweight file access: this issue needs to be solved within the fileserver. In the course of the ISLAND design it was realised that a fileserver whose properties solved the problem was at hand. It is appropriate here to describe this 'Cambridge Fileserver' [Dion 80] and explain how it might be used to support a combination of voice and data storage. The fileserver was designed to be operating system independent. It provides only primitives of the type "write these **n** bytes as bytes **b** to **b+n–1** of the file whose (bit pattern) ID is I" and "read **n** bytes starting from byte **b** of the file whose ID is I", together with the primitives needed to support hierarchical file indexing and to provide locks on files. Use of these locks is not obligatory and if locks are used it is up to the client to define the ways in which they are manipulated. More than one operating system has used this file server, each with a different style of filing system built on top of the primitives. All clients communicate with the fileserver over the Cambridge Ring.

As discussed in [Needham 82], a principal client of the fileserver is a series of processor bank machines all running the personal operating system TRIPOS. In the early development of the bank system, each TRIPOS machine used a common portion of the fileserver as its filing system and interacted with it to read, write, lock and unlock files. However as the system grew the performance of this scheme deteriorated badly. Therefore a server called the Filing Machine was created: it became the only TRIPOS client of the fileserver and hence it could handle locking of exclusively TRIPOS files more efficiently than the individual machines could. In particular it could cache directory structures and manage interlocks on them itself. The interface between the filing machine and fileserver was now much more lightweight, nearer to the simple primitives

[2] Detailed design of the translator has been the joint work of Roger Calnan and the author.

described above.

If the filing machine is a 'front end' for data users, the translator can be seen as a front end for voice users. The required lightweight locking mechanisms for voice can be provided by the translator without danger, making very loose use of the fileserver primitives, provided that the translator and other clients of the fileserver do not have simultaneous access to common files. This will be discussed further in chapter 9.

4.3 A Conference Facility

Audio conferencing is a facility expected from the modem PABX. At one time it would have been implemented by analogue mixing, using special hardware. It will be assumed that the ISLAND facility is to be digital and run on general-purpose hardware, in which case there are three obvious approaches—the single talker model, the broadcast method and the conference bridge.

4.3.1 The Single Talker Model

In this scheme, a single control server monitors each participant in the conference. The telephones provide the controller with silence detection information. Each time nobody is talking, the controller waits for somebody to start. The first one to do so becomes the 'active talker' and sends a voice stream to all other participants. If two people start to talk simultaneously, the controller choses between them, either using a fixed priority scheme (useful if one speaker is to be regarded as chairman for example) or on a round-robin basis. A hybrid of the two is also possible. A participant starting to talk but not being assigned 'the floor' will hear that somebody else has got in and stop. When the active talker falls silent the process repeats.

This was the method used in the ARPA voice experiments [Forgie 80]. Its choice was forced as a side effect of the compression technique used to bring voice down to the 1.2kbs^{-1} available from the ARPA network. This technique did not allow mixing of voice streams. (The compression technique was based on a vocal tract parameter model, which is not only unable to make a good job of compressing the sound of a symphony orchestra but also fails very badly when offered two voices mixed together.)[3] The single talker scheme is reported to have been received well by users, since in a conference when two people speak simultaneously one of them must back down anyhow. The speaker selection mechanism was fairly successful, although a break-in mechanism for a chairman was sometimes desired. The selection scheme replaced an earlier, unsuccessful manual system in which participants pushed a button to request the floor. The problem with that idea was that frequently several people would request the floor simultaneously. If the first made the same contribution as the others had intended, the others would invariably omit to cancel their requests, although they no longer wished to speak.

[3] In a system where compression of only a single speaking voice was all that was required, given that vocal tract models remove much of an individual speaker's vocal characteristics, one might usefully be more ambitious and try speech recognition as a means to compress the information further. [Knuth 84a] suggests that further compression still would then be possible by exploiting the redundancy of natural languages.

For ISLAND there are two problems with the single talker approach. First, unless the network supports a suitable broadcast transmission mode, the phone which is active needs to send packets in parallel to all other participants. As a result, the phone will not only need more power but also will need to treat conference calls differently from other connections. We wish to resist this on grounds of complexity, as we did in the case of voice recording and playback. The second problem is that the community which used the ARPA facility were scientists, who understood the reasons for and tolerated the single talker restriction. It is not clear that the public, accustomed to being able to plug two phones into one domestic line and thus achieve a simple "hear everybody" conference, would tolerate the restriction.

4.3.2 The Broadcast Method

To get around the second problem above, each active talker in the conference could broadcast packets to all other participants and each participating phone could combine received streams and feed the result to its earpiece. This is the approach used for conferencing in the Etherphone project. Again the problems with this approach concern complexity of the phone, but here to a greater extent. The phones would respond to conferences differently than to simple calls: they would need both to broadcast whilst active and handle more than one incoming stream of voice at once. The latter is objectionable on two grounds. Firstly it implies setting limits within the phone itself on the features that can be offered by the system. (The phone will only have the power to support some maximum number of participants simultaneously: increasing this means altering all the phones, whereas in the next approach to conferencing to be discussed only the bridge would need be altered.) Secondly the processing power required to support the merging of digital voice streams is considerable. At least if the voice streams consist of conventional PCM samples, which are logarithmically encoded, combining streams entails not a simple series of additions but some more lengthy arithmetic. (As an alternative a lookup table could be provided in ROM, but this would increase the cost of the phone.)

Against the first objection above, it could be suggested that once the number of active speakers reaches a certain value there is no point in combining further voice streams faithfully since all that will be heard is chaos and the incremental effect of adding more streams will be negligible. This idea is mentioned only because it is an example of the type of short-cut which might not always be acceptable. If bound into the phone hardware it would be hard to change. (The impact upon the phone's network interface of a great number of voice streams all arriving in parallel is also not discountable.) To give a frivolous example where this 'ample representation of chaos' idea falls down, it would make a poor job of handling a distributed 8 part madrigal group rehearsal.

4.3.3 A Conference Bridge

The third approach to conferencing follows the lines of that generally used by PTTs in their networks. All voice streams are sent to a common point (the bridge) where they are combined and an appropriate combination of the inputs is then sent back to each participant. This is a suitable approach for our system insofar as it enables the phones to be given simple addresses x and y with whom to exchange voice streams, x and y both being the conference bridge. Phones thus behave towards the conference bridge just as if it were another phone.

In a conference bridge server it would be feasible to use table lookup techniques. It was remarked above that non-trivial arithmetic is needed to merge PCM voice streams because these employ logarithmic encoding. An alternative to such computation is table lookup. Suppose that we require only the operation of "addition"—finding the PCM value which is the logarithm of the sum of the antilogarithms of another two PCM values. Since PCM values are 8 bit quantities, this operation can be provided by lookup into a 64kbyte table. This constitutes a large amount of memory to provide in each phone, which is why the Etherphone telephone instruments compute PCM "addition" in software. However in a single server it is much more attractive to provide lookup tables: the server will consume less processing power per conference and hence can support a greater number of clients simultaneously. The 128kbytes required for two lookup tables (the second being for "subtraction", as will be explained in chapter 6) is modest in a single server, whereas it would be unacceptable in every phone.

4.4 Uniform Interconnection

In this chapter, it has been suggested several times that the phones should view other phones and all other voice-handling entities in an identical manner. The complexity of the phones can be reduced if they use only one protocol to communicate with all external entities. It was suggested in section 1.4.2 that uniform connections between servers would facilitate running the system in several different configurations. The idea of uniform interconnection is vital to make enhancements of the system easy and cost effective to achieve, those enhancements being produced both by upgrading existing facilities and by introducing new ones. Considering the components so far outlined, introduction of voice storage facilities via the translator or of conferencing via the conference server requires no change to the phones provided the new servers talk exactly the same voice protocol as is used for phone-to-phone interactions. (The translator was orginally conceived in order to support a phone-like access to voice storage facilities.)

As another example, suppose that we wish to reduce the disc storage required for voice, by employing some compression technique. (This is an option that only certain customers, with very heavy storage requirements, might require.) There are several well-known techniques for compression. We will take here as an example Linear Predictive Coding (LPC) [Markel 76], which like most other compression methods introduces considerable delay into a voice stream, since it performs correlating computations on a large number of voice samples together. The delay is not a problem for voice storage, although it would be for a real time connection. The fact that the method acts on long blocks at a time suggests that an 'LPC server' (if installed as a separate module) may well best be placed between the translator and the fileserver. The translator uses the interface "these are/get me bytes **m** to **n** of file ERIC'sID" to communicate with the fileserver. Since the LPC server knows what compression factor it introduces, it can transparently be put between the two, using the same interface. Neither fileserver nor translator will notice the presence of the LPC server. The LPC server will pass on file identifiers unchanged, dividing or multiplying values of **m** and **n** as it compresses or reconstitutes samples. The LPC server itself need not know the natures of the servers to which it is connected, only the definition of their interface.

The only server that needs to know how services are interconnected is GOD. GOD will configure a series of connections if for example voice is to be recorded from a phone through a translator and an LPC server to the fileserver. None of the participants in the chain will have any knowledge, or need for the knowledge, as to whom they are connected to. Therefore installation of a new service becomes very simple. Server hardware is installed to run the new service and GOD is updated to know that the service is available. No other components of the system need be altered.

This approach to interconnection within the ISLAND ties in well with the ideas put forward previously on wide area interfacing. Chapter 3 proposed a hop-by-hop approach to WAN/LAN connection, meaning that a different protocol should be used in each different section of a connection. Each protocol would being tailored to its own section. The gateways between different styles of network will perform suitable protocol translation. This is compatible with our desire for uniformity. The protocol used from a phone to a WAN/LAN gateway can be the same as that used between phones and hence the phone will not see the WAN gateway as being different from any of the other objects to which it also connects.

4.5 Commonality of Server Hardware

It has been argued above that use of single-function servers is advantageous for production of software. The simplicity of the function reduces the production/maintenance costs and the standardisation of interface minimises the cost of upgrades. If the task of a server is reduced to one simple function, it can run under an operating system suited to that function, whereas the operating system of a server handling several functions will almost certainly compromise the different demands upon it. The single function approach also has benefits for the hardware structure. In practice, splitting the system function into simple parts implies that these can run on servers

(a) of relatively modest power—not state of the art and hence relatively cheap to build and not too complex to maintain

(b) which are identical from service to service, with benefits for cost of manufacture, support and maintenance.

(This is clearly not true of certain servers, such as the fileserver, but could reasonably be true of the filing machine, translator, LPC server and conference server.) This produces benefits also in terms of the costs of providing redundant hardware to increase resilience.

If similar or even identical hardware is to be used for different servers this implies a degree of over-generality in their design which is at variance with our attitude in the case of phones: each is clearly going to contain more resources than the bare minimum needed to accomplish its individual task. However the number of servers is much smaller than the number of phones. We want to define for the phone flexible and simple capabilities which will not need to be altered in order to upgrade the overall facilities of the system. The phone code can be supplied in PROM, or custom silicon can be used to realise the phone. This, combined with a minimalist approach to the phone's functions, enables the cost of this ubiquitous component to be minimised. In contrast, the cost of the high level functionality of a PABX is largely governed by the cost of upgrading/enhancing/increasing the system capabilities. There will be few servers in comparison with phones. Economies of scale suggest that it will be cheaper to provide the same hardware for as many servers as possible, even if the standard server contains as a result more features than any one service will require. The specification of these servers should be devised such as to be likely to serve the needs of more complex services than are currently being designed: the cost of upgrading the system will be reduced if a new service does not require new server hardware.

4.6 The Greater Plan of GOD

The discussion of PABX facilities will be completed with an outline of the functions apportioned to GOD. Detailed design and implementation of GOD is the continuing research work of Roy Want [Want 84]. It will not be discussed here. Of relevance to this chapter is an outline of how GOD should monitor and maintain the system. Many of the ideas on GOD's role and functions within the ISLAND system, but none of the design and implementation of GOD, are claimed as original work forming a part of this thesis.

It has been suggested so far that GOD's purposes will be the handling and interpretation of dialled codes from the phone, a knowledge of how to configure combinations of servers to provide various facilities and the sending of commands to phones and servers to set up and clear down connections. GOD is also to have dominion in the area of reliability. The reliability of the system can be considered in terms of three components—the phones, the servers and GOD. The simplicity of both hardware and software of the phone is the major contribution to its reliability. However, system-wide, this impacts more upon maintainability than availability since in common with the PTTs we must consider availability issues principally in terms of the components on which several people will be relying at a time.

There needs to be something in the system which will detect failures of servers by a variety of conventional means, such as failure notifications from the servers themselves, failures of servers to respond to instructions from GOD and status checking by a 'failure server'. It has been noted that since there needs to be much interaction between GOD and such a failure server, and since there needs to be as much special work on making each of them resilient, it is attractive to place all failure monitoring within GOD.

Use of identical hardware for most services, as described above, has the benefit that it is feasible to maintain a few spare servers as a backup for all such services. The first action upon observing that a service has failed is to load the relevant software into a free server. Many servers can be designed so as not to hold any long term state relating to the service they are rendering. It will therefore be possible to reissue "talk to *x*, listen to *y*" instructions without long interruption after having started a new server. Clearly the effects of an interruption will vary from server to server; for example the effects of replacing a failed conference server will not be serious provided that the failure can be detected and corrected before the human 'abandon this call as dead' timeout. There is not yet sufficient operating experience to tell whether this can be done fast enough in general to be useful, but there is no obvious source of difficulty. The logical final stage in this rebooting procedure is to load the failed server with a service which simply provides some automatic diagnosis.

It has been assumed in this discussion that it is possible dynamically to alter the locations of services: the Cambridge Ring naming architecture does not provide for this, although it has often been proposed that it should [e.g. Yudkin 83]. In this context however there is no need for the nameservers to provide dynamic naming capabilities, since only GOD will need to know where services are available. In this respect GOD would follow the Resource Manager scheme set out by Craft [Craft 85]. For the Fast Ring no nameserver scheme has yet been decided upon: it may well eventually allow dynamic naming capabilities.

It is evident that the GOD service itself must be very reliable. It must be replicated with state shared in such a manner that the tasks formerly handled by a failed GOD can be taken over by the others. Experiments to achieve this do not impact on the work described in this dissertation and are not the author's work. For completeness of this section it will be remarked that experiments are in hand with instances of GOD configured in a virtual ring, which simply means that there is a defined order in which communications pass between the GODs. The communications which pass around the 'ring' are updates of the states of all calls being handled by each GOD. Failure of one GOD can be detected by the others from interruption of communication around the ring and the information that was passed around before the failure occurred allows the surviving GODs to take over the calls which until then were being handled by the failed one. System designs for a redundant and resilient GOD could clearly include

(i) a single active instance of GOD with 'hot spares' on standby to handle failure of the active instance

(ii) a single active instance with hot spares but also with automatic regular rotation of which GOD is currently the active instance

(iii) a fully distributed system with all instances simultaneously active.

The last is the choice being pursued, as it is the most symmetrical and the most challenging. Also the others seem not to provide an easy recovery from the case of all 'spare' GODs failing whilst on standby and this fact not being detected until an attempt is made to use them in an emergency.

5. The ISLAND Voice Protocol

This chapter describes the voice protocol designed for ISLAND use within a mesh of Cambridge Fast Rings. It starts by reviewing issues of error rate and delay and then proposes a simple protocol. The protocol is intended only for LAN use and is to be used in all connections to or from a phone.

In an implementation of the voice protocol two things are of importance. The implementation should be kept as simple as possible in order to minimise the complexity of the phones. An implementation running in a server must permit parallel handling of several streams in an operating system environment. The latter also has implications for design of the operating system, which is discussed in chapter 6. The way in which the voice protocol has been implemented for phones and servers is nearly identical and is specified as a service named the 'ISLAND voice provider'. The description of this service presented here starts by outlining the common procedure for all applications and then lists the service options required by various clients. This chapter also describes a method for synchronising the sampling clocks in individual phones without the need to distribute a single common time reference.

The functional specification of the phone given in chapter 4 and the description of its voice protocol in this chapter completely describe the phone. The chapter ends by considering just how simple a mass-produced phone would be.

5.1 Performance Required from Voice Connections

In order to justify the design of any voice protocol, the two issues which need to be considered are acceptable network delay and acceptable error rate. In a packet network, delay will come from three sources—packetisation time, minimum transport time and statistical queueing jitter. The minimum transport time can be neglected within a local area context, being an inverse function of the speed of light. The statistical queueing jitter occurs because the bandwidth in a packet switched system is not pre-reserved and the time between demand and allocation will vary according to other offered traffic at any instant. Chapter 1 mentioned various packet networks which for this reason provide slots specifically reserved for voice. For simple phones launching packets at PCM bit rates onto a Cambridge Fast Ring the queueing jitter will typically not exceed 100–200μs, depending on the size of the ring in question. There is therefore no need to supplement this inherent bandwidth guarantee by increasing the hardware complexity of the network.

The most serious source of delay is packetisation, for a packet containing time t's worth of samples clearly implies a delay of t due to packetisation.[1] For this reason it is necessary to send very short packets of voice: it has been established that Cambridge Rings are well suited to this. The ISLAND protocol specifies a packet of length 2ms, i.e. carrying 128 bits of standard PCM-encoded voice. This length was chosen for three reasons. First it fits into a single Fast Ring slot,

[1] This delay is incurred in the transmitting phone between the time when the first sample in a packet is generated and the time when the whole packet, including that sample, is ready to be transmitted.

whilst leaving adequate space for simple protocol information. Transmission and reception in a phone will therefore be very simple, since there need be no procedure for blocking groups of slots into packets. Secondly, the delay induced is of the right order for speech on a local link. Thirdly, this packet length is the unit of 'not actually caring whether a packet is delivered', as will be explained below.

The question of what error rates are acceptable for voice has been investigated many times: an oft-quoted figure is 0.1% [Bullington 59], which applies to burst errors due to TASI 'freeze out' at the start of a talkspurt. Two more recent studies [CCITT 74, CCIR 76] have both produced the rate of 10^{-5} random errors under low sound level conditions. Which figure is appropriate depends on the types of error on the link. It is clear that simple random errors are irrelevant in the context of a Fast Ring: the error rate of the 10Mbs^{-1} Cambridge Ring is typically better than 10^{-10} [Dallas 80] and the Fast Ring gives similar performance. At this level, it is better to play back samples that contain errors than to reject them because of a checksum discrepancy. The concern on a Ring is for packets completely lost.

The distribution of jitter in a packet network normally shows a small tail-off of packets delayed an abnormally long time. There may also be a few not delivered at all (due to for example hardware level error rejection of a packet). In the context of the Fast Ring, such packets may also be accounted for by such occurrences as Ring reframes (when the slot structure of the ring is lost and the monitor has to remake it). For a real time voice connection, voice packets arriving at the destination too late to be played back are exactly as useful as packets which do not arrive at all.

The matter of concern on a Ring is therefore the ear's perception of burst errors rather than poisson-distributed single bit errors; burst errors being another way to view lost packets. Studies have shown [Gruber 83a] that the ear is not affected by bursts of errors occurring up to 1% of the time provided each burst lasts less than 4ms. Thus a packet of 2ms duration really is a unit of 'not actually caring whether a packet is delivered'. It is important to realise that in the context of any packet based stream, it is very difficult to build a service with complete guarantee of fast, first time delivery. The alternative approach adopted here is to design a service which does not need to make any such demands.

The target set in the ISLAND voice protocol work was to achieve phone-to-phone delays of a few milliseconds. The protocol described in this chapter will work in the range 2–3ms. Even considering the new Oftel delay budget allowances for private networks, this is a high figure if, for example, we wish to configure multiple ring paths into one connection, as in the case of a conference between two local phones and one remote phone. This work is aimed at showing that packet-based systems can work in the right 'ball-park', i.e. a few milliseconds rather than a few tens or hundreds of milliseconds as in the case of the ARPA, Etherphone, UNIVERSE and various other packet protocols. This protocol could be made to produce a smaller delay by reducing below 128 the number of bits sent in a slot, although the utilisation of the ring would eventually begin to suffer. We have professed not to be worried about utilisation efficiency of the Ring in this system. The only worrying effect that a reduction in the number of PCM bits sent per slot will have is that it will reduce proportionally the number of simultaneous voice streams that a server such as the conference server will be able to guarantee to transmit.

on the ring are sufficiently infrequent, there is no need to worry about checksumming. The protocol can in fact be very lightweight in all respects, sending voice as a stream without flow control or acknowledgements. There is no role for flow control, since both the network and any consumer of voice must guarantee to handle 64kbs^{-1} per stream. Acknowledgements can also be of no use: in data applications if a packet is delivered containing errors or is not delivered at all, the transmitter will time out the acknowledgement and retransmit. In a voice application there is practically no benefit from retransmission because the retransmitted voice will arrive too late to be replayed,[2] which, as already noted, is no more useful than it not arriving at all.

In any packet voice system, there needs to be some buffering in a receiver to give consistent voice quality despite the jitter in packet transmission times across the network. Much work on packet voice has assumed that this buffering should be adaptive during a call: this was the approach in the experiments described in chapter 2. However in chapter 3 we suggested that adaptive buffering is not useful where a protocol need produce a fixed maximum delay—it is better to work with a fixed amount of buffering set according to the maximum delay allowed. Adaptive buffering attempts to cope with networks whose delay characteristics cannot easily be guaranteed. The Fast Ring offers very good guarantees. It is accepted that occasionally packets may either fail to arrive or arrive with excessive jitter: the short packet size chosen means this is not a problem.

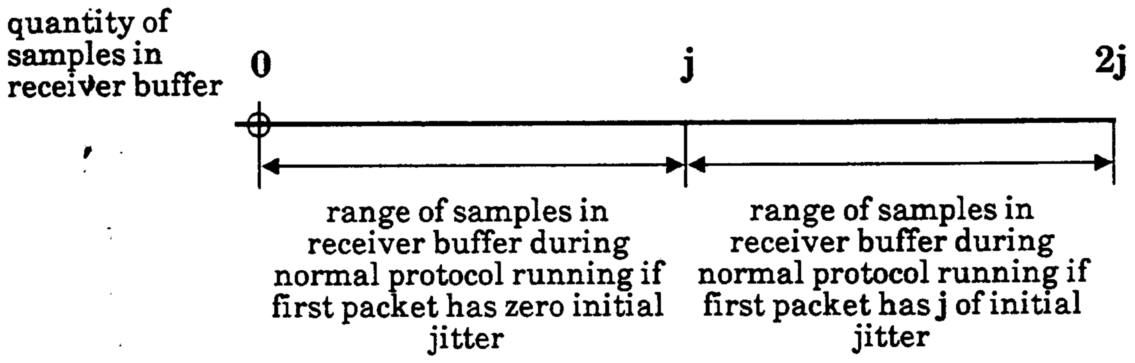
Before defining the protocol we will recapitulate the principal characteristics required of it. The protocol should be as lightweight as the prevailing conditions will allow: on a Fast Ring it can be very lightweight because the Ring is not a hostile environment. The protocol will only be required to work in the local area domain. A protocol which also had to cope with more hostile wide area conditions would be more heavyweight than needed in a local area climate, thus adding unnecessary complexity to the implementation in phones and producing delays unacceptable for the local environment.

The ISLAND voice protocol can be termed both 'strict time' and 'synchronous'. The latter implies that both transmitter and receiver have a notion of time linearly increasing without discontinuities. However the two ends neither hold the same idea of absolute time nor know the time difference between one another. This can be explained by describing how an individual voice stream is started up between two stations x and y. The system controller tells y to listen to x and when this command is acknowledged tells x to send to y. The 'listen' command tells y that the maximum expected jitter on the link is j. Upon receiving the first packet from x, y delays it for time j and then starts a regular playback of samples. This allows for the case where the the first packet was delivered with zero jitter, so that subsequent packets with up to j of jitter will arrive in time to be played back. The buffer space in the phone must be at least 2j long, to allow for the case where the first packet was delivered with j's worth of jitter and to ensure that there

2 For retransmission to be of use at all, i.e. for retransmitted packets to arrive in time to be replayed, the timeout in the receiver for requesting retransmission of packets not received would have to be set to less than the expected normal jitter time, which is not a sensible proposition.

is sufficient buffering in the phone to accommodate subsequent packets delivered with zero jitter.[3] The following figure shows how the quantity of samples buffered in the receiver will vary as the protocol runs, for the extreme values described above of jitter on the initial packet.

Figure 5.1


quantity of samples in receiver buffer
0
j
2j
range of samples in receiver buffer during normal protocol running if first packet has zero initial jitter
range of samples in receiver buffer during normal protocol running if first packet has j of initial jitter

All will be well if packets from x arrive with jitters in the range 0 to j: there will never be insufficient room in the buffer to store them, nor a lack of samples to be played back upon demand. The maximum total delay on the voice link is d+t+2j, where d is the packetisation delay, being the length (i.e. duration) of voice sent in a single slot; t is the actual transmission time across the network in the absence of any queueing delays and j is the maximum queueing delay expected normally. The component 2j is the maximum possible amount of receiver buffering at any time.[3]

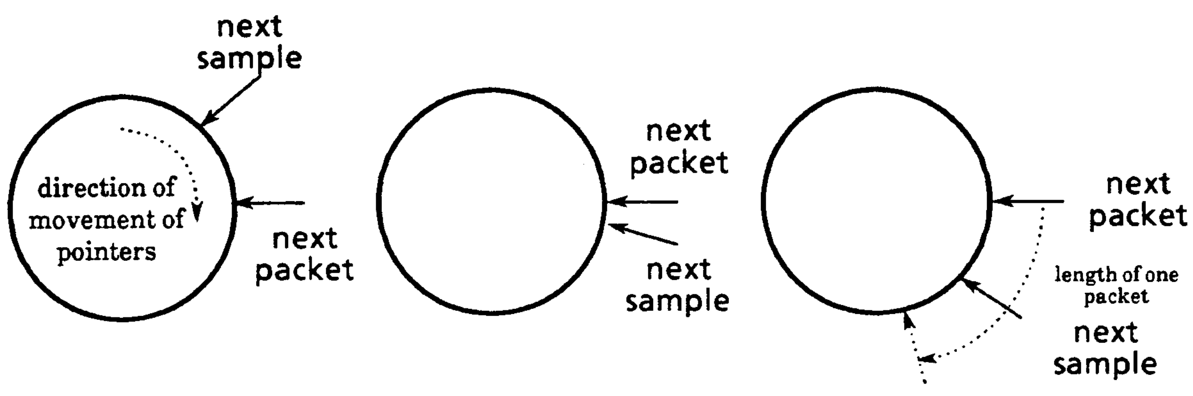
Consider now a packet arriving with a greater jitter than j, or not arriving at all. It has been established that such occurrences, unless very frequent,[4] will not upset the ear. It must now be ensured that they do not upset the protocol. The way that this is done is illustrated in figure 5.2 below. Suppose that a packet is heavily delayed. The transmitter generates packets at regular intervals. The receiver maintains two pointers into its sample buffer (which is a circular buffer). The first is incremented every 125μs and indicates where the CODEC will next pick up a sample to replay. The second indicates where in the buffer the next packet will be placed when it arrives. If a packet is heavily delayed the first pointer will overtake the second. The CODEC's sample fetching routine sets a flag playing back silence and continues to increment its next sample pointer (to retain synchronism) but now plays back a D.C. level instead of samples picked up from the buffer. When the delayed packet arrives, the reception handler observes that playing back silence is set to *true*. The packet is placed in the buffer and the next packet pointer moved up as normal. If the next sample pointer is now behind the next packet pointer then playing back silence can be set to *false*, so that the later part of the packet, which is not too late to be of use, can be played back. Otherwise the packet was entirely too late to be useful, but the pointers have been manipulated correctly for the arrival of the next packet.

[3] If the playback duration d of the packet is greater than j then there must be at least (d+j) of buffering.

[4] or regular—the ear is very sensitive to noise which forms a coherent pattern but not to a fairly high level of burst errors if these are truly random. The former was observed in the course of the ISLAND experiments when the unmatched sampling clock frequencies of two telephones caused single packets to be discarded at regular intervals. This was noticeable even at a level of a burst of 16 samples or less lost once every two seconds.

Figure 5.2

Behaviour of the protocol under normal and delayed conditions



Normal running	Abnormal Delay	Packet Arrival
The place where the next packet to arrive should be put lies ahead of the place from where the next sample should be taken. In other words, there are samples in hand.	Due to late arrival of a packet, next sample passes next packet, indicating that there are no more samples available. A D.C. level is played back.	When a packet arrives, next packet is adjusted accordingly. If it passes next sample, as above, normal playback resumes. Otherwise the D.C. level continues to be played back.

There needs to be a way to maintain synchronism if a packet fails to arrive at all. Therefore each time a packet is transmitted the sender increments a sequence number and sends its current value. When a packet goes astray, the receiver will detect this from the sequence number of the next packet to arrive. Suppose that the sequence number indicates one missing packet. The next packet pointer will be moved up the length of one packet before inserting the new arrival. Since in the ISLAND system j should be less than d, when the packet arrives playing back silence will almost certainly be *true*. In this case, after next packet has been moved to allow for the missing packet, the space between next sample and next packet will be filled with D.C. level samples. After the new packet has been added to the buffer, playing back silence can then be set to *false*.

Chapter 3 discussed the need to do something during times when there were no samples available for a UNIVERSE phone to play, rather than letting the phone just 'go dead'. This is not a concern here. In chapter 3 the concern was for long gaps between talkspurts. Here we are dealing with very short gaps: we assert that the network environment is sufficiently good that these gaps will be rare. They will therefore not be detected by the ear (according to [Gruber 83a]). It will be acceptable just to play back a D.C. level, i.e. just to hold the output of the CODEC at a constant level.

Having stressed several times already the desire to use the lightest phone-to-phone protocol that the network in question will permit, we should remark that the procedure just described is not the simplest protocol that could be got away with. Schemes where single samples rather than 2ms bursts are sent around a Cambridge Ring and schemes without sequence numbers could feasibly be implemented.[5] The doctrine of simplicity has deliberately not been taken to this limit. Engineering judgement suggests that the small increase in complexity due to packaging 2ms bursts together and adding a sequence number is justified since it makes various other parts of the integrated network vastly easier to realise. Without the packaging of bursts of samples the problems of transmission and reception of several voice streams simultaneously at a server would be practically insoluble. Both sequence numbers and the blocking of 2ms bursts of voice samples are essential if a network interface is to be realised for those servers which require to transmit and receive data and voice together (as will be discussed further in chapter 7). These two features do not add greatly to the complexity of the phone: the way in which a phone can be implemented using very modest hardware/firmware will be discussed at the end of this chapter.

The description of the voice protocol has so far only considered its behaviour under 'normal conditions'. Various error conditions need to be handled, but discussion of these will be postponed until after discussion of an implementation of the protocol, for the two are inseparable. The next issue to be considered will be the distribution of a common sampling clock and then after that a practical implementation of the protocol for both phones and the various servers will be outlined.

5.3 Distribution of an 8kHz Clock

Whilst two phones need not know the time difference between their two clocks, the above procedure works only if they have the same PCM sampling clock frequency. In conventional digital phone systems, there is a single reference distributed by the network to all speech handlers. This is rather heavy handed for a phone-to-phone connection. It is desirable to build a system where failure of the clock distribution service would not stop phone-to-phone interactions. The ISLAND phones[6] contain a crystal, trimmed as a VCO. This ensures that the sampling frequency is always close to 8kHz (as required by the filters in the CODECs for comprehensible speech) but can be trimmed to compensate for component drift etc.

Whenever a phone is receiving a voice stream from elsewhere, it compares the rates of receiving packets and of playing them back and adjusts the VCO to bring the two equal. (Adjustments are made with a suitably long time constant and in small increments). Thus two phones set at two different frequencies and then connected together will eventually adjust to the mean of the two. Suppose that a phone is receiving voice from either a wide area network gateway or from a translator. The wide area interface will contain a stable clock and it can be

[5] See [Leslie 81] for an example of a very much simpler protocol.

[6] designed and built by Roy Want, of the ISLAND project

arranged that the translator receives a similarly accurate clock. Neither of these components will adjust to the phone, so that the phone will adjust exactly to them. However a problem arises if a phone running at the wrong frequency is sending packets to a translator, this being a simplex connection, unlike the interaction of two phones. The scheme does not provide a way to signal back to a transmitter. The quality of voice from a conference server, if its input streams are at different frequencies, will also be poor, although the phones will gradually adjust to the server's clock through their receiver algorithms. It would be beneficial if a method could be found for the speaking clock. This stream is a voice stream according to the protocol definition above, emitted by a server with an accurate time reference. The phone's clock adjusts to this reference. The only unusual thing about the voice stream is that nearly all packets are missing—the speaking clock only sends a packet to each idle phone roughly once a minute. It was established above that the phone's reception algorithm can cope with missing packets. Both the network bandwidth overhead and the complexity of the speaking clock service are trivial, and the use of sequence numbers in the voice stream enables the phone's clock frequency accurately to be maintained.

Thus we have a service which normally keeps a common clock running throughout the system, but the phones will run satisfactorily in absence of the reference. The sound quality in two phones adjusting to one another may be noticeably poor for the first few seconds but it does not impair understanding of speech. Since the failure of the speaking clock will be a rare event, this occasional reduction in quality can be tolerated.

There is currently less need for the speaking clock in the ISLAND demonstration system than there was when it was first implemented. The telephones then contained relatively unstable oscillators, but improved circuitry and higher quality crystals now produce a discrepancy between clocks of 10^{-3} in the worst case and 10^{-5} more typically. At these rates the quality of a voice stream will not normally be affected. However two things should be noted. First in the interest of economy we may wish to put low grade crystals into a production phone. Secondly results of studies on perception of error rate such as [CCITT 74] or [Gruber 83a] do not apply to this situation. They all deal with random burst or single bit errors, whereas errors due to clock mismatch will be regular. One early lesson from implementation of the ISLAND voice protocol was that very low rates of regular errors are perceptible (such as the loss of every thousandth packet or less, as discussed already in footnote 4 of this chapter).

5.4 Provision of a Voice Service

The next issue to consider is how a voice delivery and transmission service should be provided to a client in (a) a phone (b) a server handling several streams simultaneously. The answers have turned out remarkably similar in both cases and are contained in a service named above as the 'voice provider'.[7] The service will be described first according to the way that it is

[7] Details of the voice provider's specification were formulated by the author and Roger Calnan

provided in phones. Differences for the cases of various voice-handling servers will then be noted.

The philosophy of the voice provider, like all of this approach to a voice protocol, derives from the observation that voice streams behave very well on the Ring, thus requiring little attention whilst running. For the phone this is very convenient since it reduces complexity. In a more complex real time application such as the translator we wish to minimise explicit interaction between the voice provider and its client, preferably confining it to channel set-up and clear-down. This can easily be done if the voice provider and its client normally communicate 'implicitly', using the pointers described in section 5.2. When a client calls the voice provider to set up a voice reception channel, it specifies the locations of the pointers and flags needed for the voice reception protocol. As it receives packets, the voice provider then updates the next packet pointer and reads the next sample pointer, setting playing back silence to *false* at appropriate moments. The client, as consumer, will update next sample and read next packet, setting playing back silence to *true* at appropriate moments.[8]

The advantage of this approach is that no explicit calls are made between the voice provider and the client, so that the system is very efficient—in such an application as the conference server this means that the number of streams which can be handled simultaneously is increased. To avoid explicit calls and to maximise efficiency, the voice provider is defined such that both provider and client can tell the state of the protocol by examining only pointers and flags. There is no need for example to keep a history of the orders in which various pointers have passed one another.

It is clear that a similar scheme can be implemented for the case of transmission, such that the generator of voice packets updates a pointer indicating when packets are ready to be transmitted and the voice transmission provider looks every 2ms for such packets and transmits them. Since transmission of voice streams across the Ring will normally be very regular, these simple approaches will work well given

(a) a circular buffer of sufficient size and
(b) procedures in the provider to handle a few error conditions that can occur.

The size of buffer required by the provider is 4j[9] rather than the 2j needed to hold all the samples that might be buffered at any one time. This is to allow testing whether one pointer is behind or in front of another. In normal running next packet should be no more than half the buffer ahead of next sample. If it exceeds this or falls behind then this can be detected by looking at the positions of the pointers. If a buffer of size only 2j were used then 'too far ahead', 'behind' and 'okay' could only be told apart by keeping a history of how pointers had crossed one another. The larger buffering scheme enables the state of the system to be learned from the current states of variables without any history and this allows both provider and client to take decisions without interacting with the other party. Exclusive use of a set number of state variables lends itself better to hardware implementation than a variable sized stack of history information.

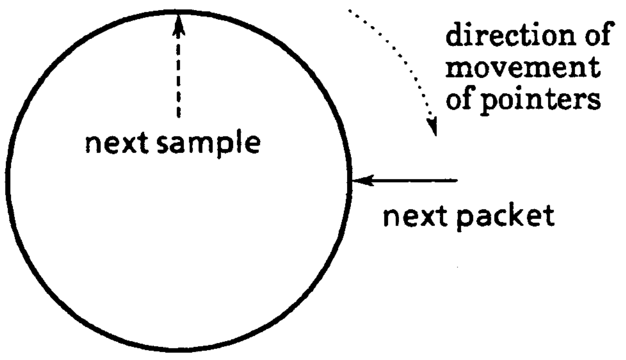
[8] If the reader is becoming baffled by detail at this point, (s)he should note that this is not very different from the standard procedure for manipulating a circular buffer.

[9] or 4d, whichever is the larger: it is more efficient to make the buffer an integral number of packets in length so that tests for wrap-around and no samples available need only be performed per packet rather than per sample.

Figure 5.3:

The voice stream receiver buffer under normal and abnormal conditions

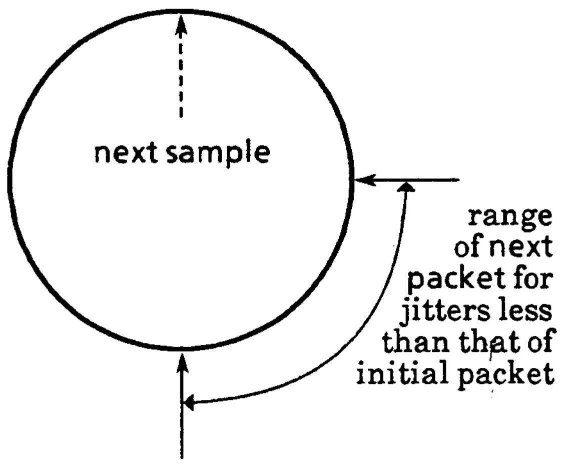
The circular buffer at the start of the protocol



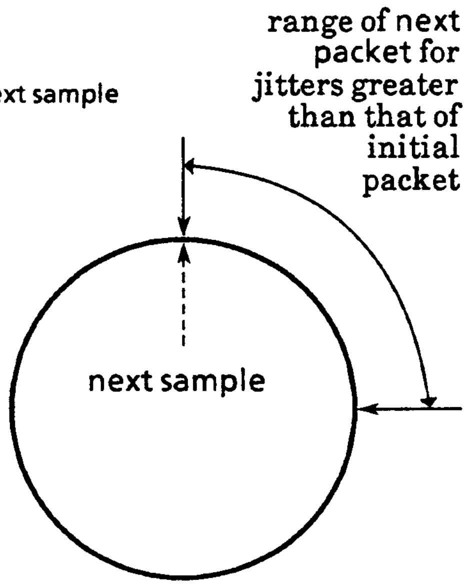
Position of the pointers with first packet received, after waiting time j. For convenience, these diagrams are drawn for $d=j$. In these figures, next sample is used as reference and is always shown in the same position.

Conditions of normal jitter

note that next packet *is in the half of the buffer ahead of* next sample



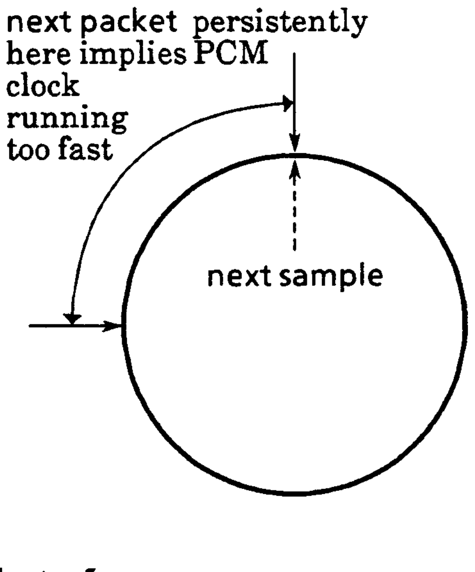
next sample
range of next packet for jitters less than that of initial packet



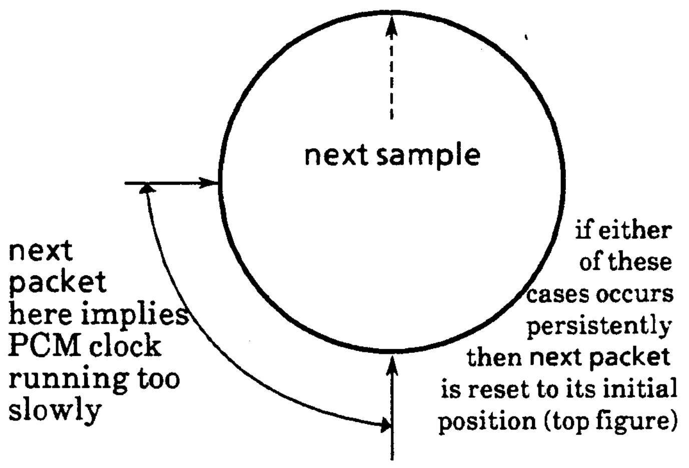
range of next packet for jitters greater than that of initial packet
next sample

Conditions of abnormal jitter or of unmatched PCM sampling clocks

note that next packet *is in the half of the buffer behind* next sample



next packet persistently here implies PCM clock running too fast
next sample



next sample
next packet here implies PCM clock running too slowly
if either of these cases occurs persistently then next packet is reset to its initial position (top figure)

The approach is believed to be worth the extra 2j of buffer required.

The error conditions which may arise concern cases where the reception buffer is overfull or habitually empty. In normal running, there may occasionally be no samples available. This must not occur too frequently and there should never be insufficient room in the buffer for packets when they arrive. There are two ways that these events can occur. One is that the first packet, against which synchronism is measured, is unusually delayed so that the buffer then is constantly overfull. The second case has been observed much more frequently: where the two phones' clocks are initially not at the same frequency, this will cause the buffer gradually to fill or empty.[10] When the frequencies do equalise, the buffer may then be left persistently too full or empty.

The diagrams in figure 5.3 show how the buffer will appear in normal and abnormal conditions. (They are drawn for convenience for the case of **d=j**.) The first diagram shows the state of the buffer just before playback begins, assuming that the jitters on the first two packets received are the same. The next two diagrams represent normal running conditions: they depict the same situation as shown in figure 5.1. Finally the last two diagrams show the buffer too full or empty.

The provider checks for continual occurrences of buffer overfull or empty (not single occurrences as the protocol is designed in expectation of these) and in such a case simply re-initialises the reception pointers—when an error condition is detected the provider causes the next received packet to be delayed for time j before playback, just as at start-up. The provider also takes suitable actions upon detecting abnormal leaps in the sequence count. It discards a single packet with an abnormal sequence number, as this may simply be a bit error, and will not reset its current sequence to the incoming sequence until this sequence appears persistent.

In the experimental ISLAND system the value of j is given to a phone (and to a server) by GOD as part of the 'listen to *y*' command. Various values of j could be used just as the variable jitter estimates described in chapter 2 were—for example they could be used to make the task of the translator easier. The translator has to launch several voice streams onto the network simultaneously and may have more difficulty than a phone in launching a rock steady stream of packets.

However this technique is not universally applicable, unless increased values of j are acceptable as part of the delay introduced by the transmitting component, which is questionnable in the case of a LAN/WAN gateway or the conference server, both of which have as much of a problem in transmission as does a translator. For any value of j used, the phones must have 4j of buffering available. Large buffers will increase the cost of the phone. There has not been enough experience of operating the ISLAND experimental system to know whether a single value of j could always be used and therefore bound into the phone implementation. If so then this would make the phone simpler.

[10] In the system discussed in [DeTreville 83], buffer over/underflow also occurred much more frequently due to clock drift than because of packet loss in the network. That system used clocks of wide tolerance and had no frequency adjustment mechanism. Since it performed silence suppression, adjustment of silence gap lengths could be used to correct the timing.

5.4.1 Silence Detection

It has been suggested that silence suppression is inappropriate for voice transmission on a LAN, but is more useful on wide area links where bandwidth is expensive and also for disc storage. However the ISLAND experimental phones include silence detection hardware. This hardware is like that used in the UNIVERSE phones. The reasoning behind its inclusion was that at the expense of only a modest amount of analogue hardware packets could be sent out containing an indication of the nature of their contents. All packets would still be transmitted, regardless of the status of this silence flag. As well as being useful to a conference server, as will be seen in chapter 6, the information could be used by a wide area gateway to determine when to suppress transmission. For the gateway to compute the silence flags itself from the digital samples would represent a non-negligible load on a general purpose processor.

[Lamel 81] and [Gan 86] suggest that reliable silence detection is non-trivial. It might therefore best be performed in the gateway rather than in the phone. The gateway would need to know about the ambient noise level where each phone was located. The ISLAND phones simply have a preset resistance to set up the noise level for each phone, just like the case of the UNIVERSE phones. As an alternative to this some server external to the phones could be charged with knowing the threshold level appropriate to each phone and supplying it to the gateways etc.[11] This information could be collected dynamically from the phones: one can envisage a service working in a manner opposite to that of the speaking clock. When phones were idle GOD would periodically tell each one to send an outgoing stream to the 'ambient noise server'. A standard voice connection would be set up from each phone for long enough for the server to calculate an ambient noise level for it, making suitable assumptions about what was sound and what was silence. This service, like the speaking clock, would be useful when running but not severely affect the system were it to fail.

The phrase 'suitable assumptions' does however gloss over a certain complexity. We experimented with determination of ambient levels in the Etherphone system. For speech and for miscellaneous background noise, reliable results could be obtained by considering the relative frequencies of occurrence of different energy levels and using that profile to set a threshold level. [Gan 86] reports the same thing. However it was found that such a calculation would set an impossibly high threshold level for a phone that was located in the same room as a symphony orchestra for example. Another (silly) example unlikely to give good results would be a phone left in a modem coupler whilst 'on hook'.

It is assumed that where a fileserver or WAN gateway used the silence detection information to discard samples or compress a voice stream[12] during silence periods that when the stream was relaunched onto a LAN link some effort would be made to recreate the silent periods and to recreate also the silence flags that would have been present in the original signal: this latter detail poses no particular problems.

---

[11] Even if the phones do detect silence themselves, it might be better for them to pick up this information from such a server.

[12] It may well be appropriate for a WAN gateway to use silence suppression and for a fileserver to employ variable rate LPC or a similar technique, since the latter gives the more efficient compression ratio but introduces delays unacceptable even for wide area connections.

It seems clear that suppression and reconstitution of silence is not a function appropriate to the local area environment. It involves considerable complexity, as discussed in chapter 3, and is therefore inappropriate for inclusion in the telephones. Silence detection involves less complexity and there is more of a case for placing it in the phone. Whether this is the correct approach or whether it is better performed only in components (e.g. the gateways, voice filing mechanism, conference server) where the information is needed is not a question that can be resolved except in the context of a mass production design of the phones. In the existing implementation of ISLAND detection is performed by the phones.

5.4.2 Provider Options

The description of the voice provider given so far has reflected the way in which it is used in a phone. It is also used by various servers, for which slightly different operation is required. The differences are small. Several references have been made already to the way in which servers use the provider. These have been cases where their behaviour is the same as that of the phones.

To identify the important similarities first, all clients of the voice provider use circular buffering, with explicit contact occurring only to start up or close down a stream. However for a translator space should not be left in the circular buffer to denote missing packets. The occurrence of missing packets should be flagged in the buffer. For the phone and the conference server spaces should be left—the voice provider should lay the stream into the buffer in exactly the correct sequence. A phone wishes to be notified of the increase in the sequence number each time a packet arrives, so that it can adjust its VCO. The options provided for voice reception are

Add Silence Flag:

If this option is set, a (16 bit) flag is placed in the buffer together with each voice packet to indicate whether or not the samples are to be considered as useful sound. This silence flag can optionally be placed before or after the packet to which it refers. The 'after' option is the most efficient for the provider (mirroring the way in which packets are actually sent around the Ring). The 'before' option is intended for the case of dual processor servers (to be described in the next chapter), where the provider runs in the interface and effectively reformats the incoming packets to make the task of an application, running in the main processor, possible within the processing power available.

Insert Silent Packets:

This option applies to the case where a packet is received with a sequence number greater than that expected, indicating that packets have been lost. If the option is set, the provider will add 'packets' of voice to the buffer, these packets consisting of PCM samples all having the value which represents the mean output level of an A-law CODEC. Having inserted as many of such packets as are indicated by the sequence number gap, it will then place the newly received packet in the buffer. If this option is not selected, the buffer must contain space for a further word per packet of voice, in which the provider will write the number of packets missing between each pair of packets actually received.

Set Samples Available:

If this flag is set then the provider will set playing back silence to *false* every time it places in the buffer samples which are of interest to the client, i.e. which are in front of the next sample pointer.

Test For Buffer Full:

This flag specifies that the provider should discard any packets which, if written into the buffer at the time they arrived, would cause packets as yet unprocessed by the user to be overwritten. Any packets thus discarded will count as missing for the purpose of setting the value of the missing packet count thereafter.

Not all combinations of the above options are useful or sensible. A 'complete implementation' of the voice provider would not implement any combinations where

Test For Buffer Full = *true* = Insert Silent Packets.

An explanation of how the provider behaves in each useful case can be facilitated by the definition of another pointer, used by the consumer of voice samples. End of Current Client Packet points to the end of a 2ms packet, into the middle of which next sample currently points.[13]

In order to define the exact behaviour for each option, the combinations can be grouped into six cases:

Case	Test For Buffer Full	Insert Silent Packets	Set Samples Available
0	NO	NO	NO
1	NO	NO	YES
2	NO	YES	NO
3	NO	YES	YES
4	YES	NO	NO
5	YES	NO	YES

Cases 0–3 are designed to support servers in which real time processing of voice occurs. Where these cases are used the client should allocate a buffer sufficiently large that it will not become more than half full of unprocessed samples in normal running. Hence the client will never need to check for the case of the buffer becoming overfull. (The provider will regard an overfull buffer as an error condition and reset the pointers.) An obvious example is the phone, in which the provider continually places samples in the buffer and the user advances through the buffer at a constant rate, using the samples it picks out from the buffer whilst playing back silence is *false*.

Cases 0–2 need no further comment. In case 3 playing back silence is only set *false* if (in the simple case of no missing packets) Next Packet is ahead of or at the same position as End

[13] It has been said already that the voice provider can be made more efficient if the client has to check some conditions (such as buffer wrap-around) only once for every 2ms packet. *End of Current Client Packet* indicates where next these checks should be made.

of Current Client Packet when the packet is delivered.[14] Where a packet arrives and would be delivered such that Next Packet is behind the End of Current Client Packet the provider need not place it in the buffer. Where packets are detected as missing, they need only be set to the DC sample value if a similar criterion applies.

Cases 4 and 5 are intended for use by clients which are not doing real time processing and whose buffers could fill up due to overload of the server or of another component of the system such as the network itself. In the case of the translator, an obvious example of a client requiring these options, the congestion might also be in the file server. In these cases, contrary to what we have already said, there is no notion of the client being 'ahead' of or 'behind' the provider, but rather of the buffer being somewhere between full and empty. The buffer is full if only the space for one packet is not currently occupied by samples delivered by the provider and not yet consumed by the client. (This is the normal behaviour for a circular buffer.)

In contrast to the several options for reception, transmission by the provider only has one normal form. Items are inserted by a client into the circular buffer in the following order:

number of missing packets before the following samples,
a packet's worth of samples,
silence detection flag for these samples.

The reasoning is that the generator of a voice stream is always able to supply all of this information, which is ordered reflecting the actual format of a voice packet on the Ring for efficiency. This contrasts with the task of the receiver, which is to provide a stream of samples reconstituted from the the packets received around the ring. This function can usefully be separated from the client that requires the stream, both in the case of a dual processor server where the client may be a different process in a different machine and in the case of the phone, where the client wishes simply to wake up every sampling clock period and pick a sample out of a buffer.

The only variation on the above transmission format is that which is desired for the conference server, which will be discussed in section 6.5. In that case the client declares that there are no missing packets and omits the first field.[15]

5.5 Minimal Phone Hardware

The target of the design of the ISLAND telephone has several times been stated to be a telephone that could be produced very cheaply and with a minimum of hardware. As might be expected, the experimental phones built for ISLAND research were not at all modest in their hardware. These experimental phones consist of what will be defined in the next chapter as a single processor server, plus telephone hardware (designed and built by Roy Want) connected to the server as a VME bus peripheral. Put another way, the phones contain a Motorola MC68000 processor, a large amount of RAM, very many gratuitous peripherals and the capability for the phones to be down-loaded from a bootserver. They occupy half of a 19 inch rack each. The

[14] Pointer q is ahead of pointer p if and only if $q - p > 0$ AND $q - p <$ buffer length / 2 OR $q - p < 0$ AND $q +$ buffer length / 2 $< p$.

[15] For efficiency in this case the provider may be defined as allowed to overwrite the dibyte before each packet of samples immediately before transmitting it, but must set it to a well defined value (the 'no packet arrived' value in section 6.5) immediately after transmission.

telephone hardware includes several features surplus to the requirements of the protocol outlined above.

This is not unreasonable. The phones were based on the same hardware as servers to economise on development effort. The generous supply of peripherals and the down-loading capability makes a realistic environment for development and experimentation. Nevertheless all the protocol work outlined above is aimed at producing a phone which could be realised with minimal hardware. The complexity of the phone might be roughly

the Ring station IC and Ring repeater circuit

a mask-programmed single chip microcomputer to handle the interactions with GOD, monitor a keypad and the Ring, set up in the Ring IC the destination address for the voice stream when it changes and notify the protocol handler chip (outlined below) of incoming voice packets

a CODEC chip to convert between digital and analogue speech, interfacing to

a large gate array implementing the voice protocol. This would pull voice packets from the ring IC and load them into it, as well as driving the CODEC. It would implement all the voice provider/voice handling procedures, possibly including (digitally processed) silence detection. It would also drive the control input of

a VCO-based 8kHz sampling clock.

The addition of some analogue circuitry, a buzzer, a plastic case etc. would complete the minimalist phone realisation.

6. Servers for Voice Facilities

This chapter discusses the design of servers suitable to run the services which make up the ISLAND voice system. It begins by considering the processes which will constitute these services and the levels at which the processes will run. Next the important types of operating system are discussed; their suitability for voice servers is considered and the properties of TRIPOS, the system which has been used, are outlined. Various server designs running TRIPOS already existed, but for ISLAND use a new generation has been built. The important characteristics of an existing set of servers are outlined, together with the reasons for building the new generation.

The ISLAND voice provider needs to be closely coupled to low level Ring primitives in order to produce satisfactory performance. The ISLAND servers facilitate this by use of a software structure in which Ring handling is brought into the main programming environment and by careful definition of an interface to the lowest level of Ring handling. On top of this interface sit both higher level i/o primitives for data traffic and the voice provider.

ISLAND servers have been built in single and dual processor configurations. A operating system structure to provide the environment desired in a dual processor system has not yet been completed, but its desired characteristics are outlined. Mechanisms appropriate to ISLAND use have been implemented to boot and load servers: their uses are discussed. The chapter ends with description of an implementation of stream compounding routines, which form the core of the conference server.

6.1 The Voice Provider under an Operating System

The functions that the voice provider fulfils for a client have been outlined. It is easy to visualise how the provider can be realised in a phone, simply because the phone is a straight-forward piece of software. In contrast, for higher-level systems its integration is not so obvious.

Consider first a conference server. The server will be using the voice provider to handle both transmission and reception of a large number of voice streams. The majority of its processing power will be spent on this and on compounding groups of streams. However it will also be interacting with GOD, to set up and clear down streams and to send status reports. There will be a number of such tasks to perform, which do not represent a significant processing load. The voice provider and compounding routines are likely to be implemented in assembly language, for efficiency. This does not represent an unreasonable amount of effort, since

the voice provider is defined so as to be applicable to a large number of systems

the compounding routines will not be very large or complex—they will not be made especially difficult to understand for being coded in a low-level manner.

However the miscellaneous control functions should be written in a high-level language. They will be relatively complex compared with for example the compounding routines. Since they will only run for a very small proportion of the time their efficiency is not crucial. Unlike the phone streams, whose i/o is handled by the voice provider, the control functions will probably use the standard LAN data protocols (single shot or reliable data stream), which are also likely to be implemented in a high-level language.

Consider next the translator. All that was said above about the requirements of the conference server is true for the translator, except that the translator will not use low-level software to process voice samples. Also no one voice stream will both come in and go out again via the provider. Instead samples will arrive in a voice provider circular buffer and then will be written out to a fileserver (or something else) in large blocks. The reverse path will also be supported. Communication with the fileserver will use standard data protocols and should be written in a high-level language. Although that communication will not be a negligible proportion of the server's processing load, this transmission/reception will be in large blocks. The inefficiency of high- rather than low-level software manifests itself largely as a per-block overhead. (The actual data block transmission and reception will most likely be implemented in assembly code anyhow.) Thus in contrast to the voice streams (of small packets) fileserver interaction can be handled perfectly adequately at a high level. The client of the voice provider will also probably be high-level code in the case of the translator. One design requirement upon the translator was that it should provide a large buffer for each voice stream being handled in order to make the real time requirements placed on the fileserver as loose as possible. These large buffers imply that the frequency with which the translator needs to check the status of each provider buffer is low and so this checking can be handled in a high-level language.

Excepting the requirement to interact with the fileserver, all that was said about the translator would apply to an ambient noise level calculation server: a voice stream would appear in a large circular buffer and at infrequent intervals a high-level routine would process a large number of samples in one go in order to determine the ambient level of the stream.

What requirements does the above discussion place upon the structure of an operating system for use in voice servers? First it must facilitate development and integration of high- and low-level code. Consider a control stream between a server and GOD. Suppose it is carried by a reliable data stream protocol. There are likely to be processes (i) handling transmission of packets across the network (ii) implementing a data stream protocol on top of this (iii) running the control functions. This clearly calls for a multiprocess system and that system must support both high and low level processes.

6.2 Choice of an Operating System

The first useful division which can be made to distinguish operating systems is between 'firewalled' and other systems. In a firewalled operating system various high-level processes are each given their own virtual memory space and are prevented from breaking out of that space. On the other side of the firewall is the kernel, which has access to the capabilities of the physical machine and which performs a large number of functions. The most popular example of this type of system is UNIX™, which particularly in the U.S.A. is currently achieving a level of popularity and infamy only rivalled by the FORTRAN programming language. This section will consider UNIX rather than other firewalled systems, since whereas others tend to be available only on mainframes or large minicomputers, UNIX is available for systems of the size and style of the ISLAND servers. The UNIX kernel handles all communication between firewalled processes; creation, scheduling, interruption and deletion of processes; plus all i/o functions. I/O functions include file access and all network handling. In the case of a reliable data stream the kernel would contain components to handle both packet transmission and reception across the network and also very likely the stream protocol on top of this.

The prime advantage of firewalled systems is that they can vastly ease the task of large software package development, if that software can be run on the safe side of the firewall. In an unprotected environment the time spent in detecting errors which cause processes to trample upon one another can be large. The kernel of a firewalled system can prevent such interaction and allow processes violating their access rights to be frozen for debugging. However inspection of the functions constituting the voice services described above suggests that many will have to be run on the 'wrong' side of the firewall.

The voice provider would certainly be in the kernel. An attempt to run compounding algorithms for the conference server in a firewalled task would be frustrated by performance problems. Continuing with UNIX as an example, its process switching time is of the order of tens of milliseconds. The entry time for a driver from a hardware interrupt is rather less than this, but nevertheless for real time servers such as the conference server process switching may completely rule out the performance required. (The widespread use of UNIX for computer networking has led to eminent Computer Scientists giving lectures with titles such as "Why Network Protocols Don't Go Fast" [Clark 86].)

If we are 'forced into the kernel' by performance problems associated with processes, there are then problems in writing those parts of our servers that would reside in the kernel. The UNIX kernel is a very large piece of systems software and worse still, a single threaded piece of software. Every modification requires a new release of the system. This is not the only reason why the kernel is not attractive as a development environment. It is estimated (from a major piece of work in building a distributed UNIX system [Satya 86]) that writing software in the UNIX kernel takes a competent programmer ten times longer than writing code of equivalent functionality as a protected process. If the more complex parts of the voice servers would have to be written in the kernel this suggests we are considering the wrong type of operating system.

The alternative is a single address space system. A great many examples of this type exist, from very low-level multitasking systems for process control to sophisticated high-level environments such as Cedar [Swinehart 86]. It has been said so far that a suitable operating system must support both high- and low-level languages. More specifically it must be possible to write 'drivers' for hardware device level handling and also run a series of high-level language processes. These processes may well call assembly routines for efficient execution of particular functions. A single address space system lacks the easy debugging capabilities afforded by firewall violation detection. However this need not be a problem. As experience with Cedar and several other environments shows, a process written in a 'type-safe' language is easy to develop and is largely prevented by compiler checks from corrupting other processes. The driver routines will still be capable of wreaking havoc, but nobody who has ever written a driver for any system (firewalled or otherwise) will be under the illusion that this is a job to be undertaken without extreme care. The art of writing drivers is to make them as simple as possible, strictly limiting their functionality to that which needs to be handled efficiently. Exactly the same applies also to assembly level language routines called from a high-level process.

The reason why so many words have just been spent in comparing operating systems is that other projects currently looking at Integrated Services facilities based upon LANs are using the firewalled approach. For example the work at Columbia University, using MAGNET as a transmission medium, is based upon SUN workstations. These are intended to cater for all a user's office requirements, including voice handling. However quite considerable extra hardware and software is needing to be 'built onto the side of' each workstation to implement the real time components (since UNIX gives no real time guarantees), and still more software is having to be provided within the SUN's UNIX kernel.

We are not alone in feeling a dissatisfaction with large cumbersome kernels. For example, the desire to be able to write low-level programs efficiently in a way that is not possible in UNIX is a major motivation for the MACH project [Tevanian 86]. MACH attempts to rebuild the UNIX kernel functions as a message passing rather than a single-threaded structure. MACH owes many of its ideas to the TRIPOS operating system (described below). It uses a single address space and an object oriented programming language. Above the message passing system, the MACH user is then given a UNIX-lookalike interface and firewalled environment for general applications. The designers of MACH believe that a firewalled environment is essential for running the many popular applications programs which are written in unsafe languages. However for embedded applications, such as the ISLAND servers, there is no need for such a firewalled environment to sit atop the structure.

6.2.1 TRIPOS

We have already discussed the advantages of partitioning the voice system into a series of small servers each of which has a simple function. The operating system used for each can clearly be chosen to suit the requirement of the particular service. A different system may possibly be used for small voice servers from that for workstations or for GOD. For voice servers the TRIPOS operating system was chosen. TRIPOS [Richards 79] is well established in Cambridge as an operating system. In particular it is the basis of the processor bank. It is not the only system of its type in existence and part of the reason for its choice was its availability and support locally. TRIPOS has several features that make it more appropriate for voice servers than other systems that were also available. It is a multi-tasking message-based operating system, designed as a run-time environment for BCPL, which is an untyped systems programming language. TRIPOS supports a variety of different languages and in particular much programming has been done on TRIPOS in Modula-2. Modula-2 [Wirth 82] has a 'type-safe' subset in which most programs can be written, so that the unprotected single address space of TRIPOS need not be an impediment.

TRIPOS also supports 'devices'—assembly level drivers which use the same message passing format for inter-process communication as do high-level processes (which are called 'tasks' in TRIPOS). The devices also allow for interrupts from hardware: devices are mostly hardware device drivers, but there need not actually be an interrupt routine handled directly by the TRIPOS device itself. Devices can be mounted and dismounted from the running operating system during testing and development (or in production systems for that matter) under control from a 'command line interpreter' (CLI), the TRIPOS interactive command interface. This facilitates the development of drivers under TRIPOS, compared with an environment where modifications to drivers require a kernel release for example.

Object oriented languages are currently fashionable as an efficient means of producing reliable software. An object-based system was available at Cambridge, but was a garbage collected language system without an asynchronous garbage collector. It could not therefore be used for time-critical voice servers. To produce the relatively simple software required for voice servers, a type-safe language is perfectly adequate. The very much more complex GOD is being constructed in an object oriented style.

In a discussion on choice of an operating system the Cedar programming environment should be mentioned. Although this was not available for ISLAND work, it was however used as a basis for the applications work described in chapter 8 of this dissertation. The basic characteristics of Cedar are of interest and relevance. Cedar features a single address space, a type-safe (and asynchronously garbage-collected) language and very lightweight processes—lightweight in that the overheads for process creation, deletion and (monitor-based) communication are very small. Cedar is built upon the 'PrincOps' instruction set [Xerox 85], a language-specific instruction set which has been implemented on several microprogrammable machines. All of these machines have a set of microcode to run the operating system (i.e. to implement its instruction set). The instruction sets to support various peripherals are separate sets of microcode, so that each has its own optimised instructions for high performance [Lampson 81]. (The designers found that this gave better performance and a far better performance/cost ratio than using separate dedicated hardware for each peripheral.) The handling of a LAN and of for example local discs is thus implemented in microcode entirely separate from that used to support the main operating system. This means that, in contrast with the discussion to follow about our servers and their interfaces, it would be feasible to use the standard LAN handling microcode for voice streams and still obtain good performance, without writing any voice-specific code in microcode.

6.3 A Structure for Voice Servers

6.3.1 LAN Access

Although TRIPOS was already available running on the machines in the Cambridge processor bank, there were reasons why a new generation of machines was needed for the ISLAND servers. This section discusses characteristics common to both types of machine and the architectural differences which distinguish the new generation.

The processor bank machines all contain two processors per system. The 'main processor' runs the TRIPOS kernel (which largely consists of a scheduler and message passing mechanism), all TRIPOS tasks and the devices handling the clock, local terminal, local disc etc. as fitted. The 'interface processor' handles the transmission and reception of basic blocks across the Cambridge Ring. The main processor contains a very simple device representing the Ring, whose function is to send requests for reception, transmission etc. to the interface processor and to receive interrupts from the interface processor when requests are satisfied or have failed.

The interface processor runs a small amount of assembly code to transmit and receive basic blocks and also to load system images from the Ring for both processors. The code also allows for simple control (resetting, halting, resuming and reading/writing the memory) of the main processor. The reason that the Ring handling primitives were partitioned off from the rest of the TRIPOS software was that they were an easily identifiable and simple section of the software that accounted for a significant amount of the execution time. In early ring-based implementations of TRIPOS (for minicomputers) predating the processor bank and also on the very early processor bank machines, there was only one processor; the full ring handling software ran as a TRIPOS device. The dual processors brought a significant performance improvement, particularly for the processor bank. Most of the processor bank machines have the Ring as their only physical peripheral (apart from a 50Hz clock), through which all file and terminal access occurs. The performance of the Ring device is therefore crucial to overall performance.

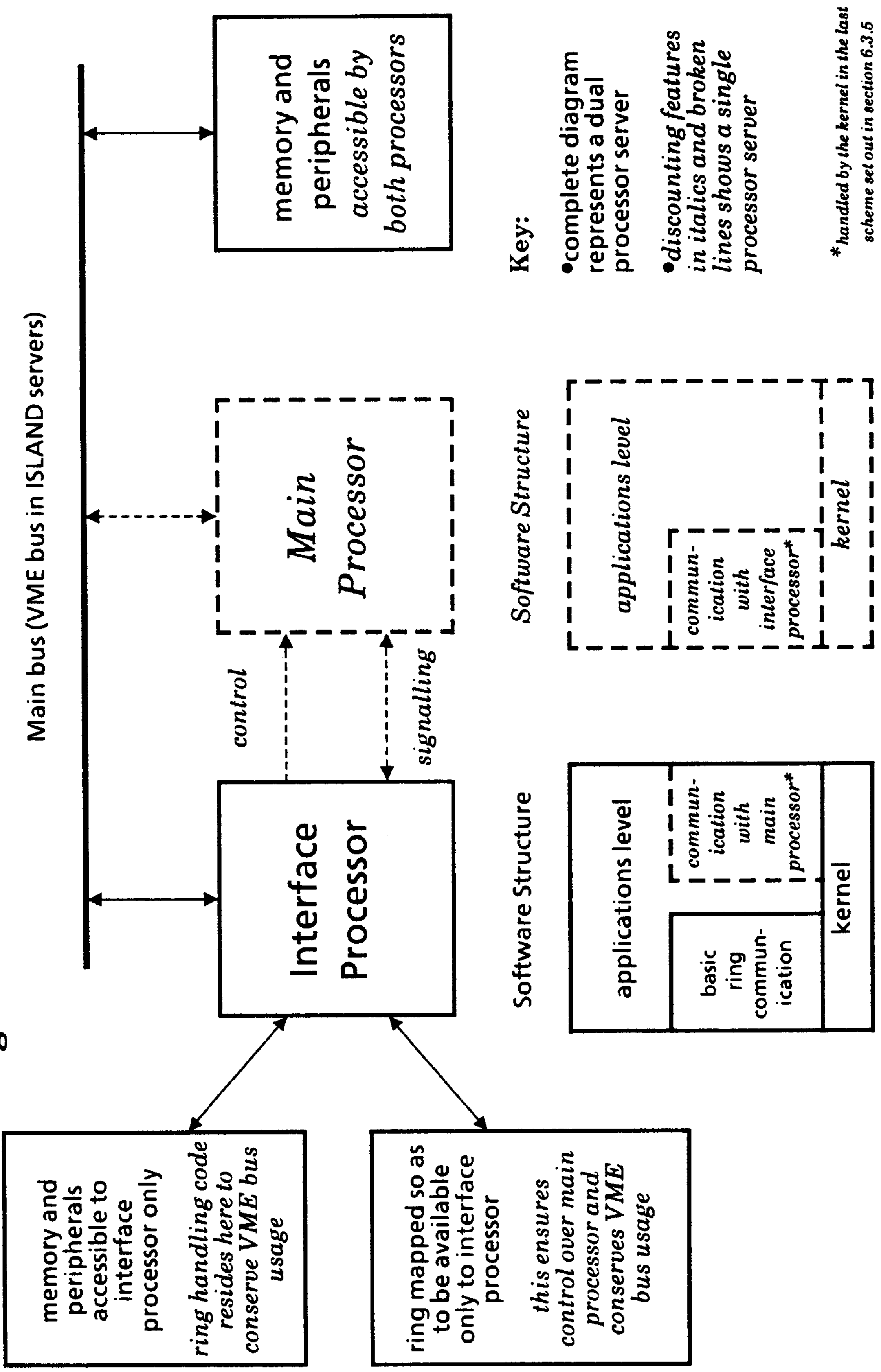
Because the Ring is the only peripheral on a processor bank server, complete control of the server must be possible from the Ring. Therefore the interface processor must be able to control the main processor. It is clear that the interface code must be trusted and hard to break if control of the server from the Ring is always to be possible. The reception of commands to control and debug the server implies some cooperation on the part of the interface processor. In the processor bank machines, the interface processor is not programmed by users: one of a very small number of trusted ring handler programs will be running in it. The main processor is fully programmable by the user and hence may run wild. For this reason the interface processor executes out of its own local memory (RAM and/or ROM), which is not accessible from the main processor. Similarly the Ring hardware appears as a local peripheral to the interface processor and is also not accessible from the main processor. This prevents the main processor wreaking havoc on the communications channel being used to debug it. All main processor memory can be accessed by the interface processor—not only for data transfers during normal running, but also to allow full debugging.

The interface processor uses its own local bus to execute code and to access the Ring. It accesses the main processor's bus in normal running only to deliver data from or copy data to the Ring, and so hardly affects the bandwidth available to the main processor on its own bus. The code running in the interface processor handles only block reception and transmission, except in the case of one system where the processor also handles a stream protocol. This addition was made on grounds of performance, but required a complete rewrite of the interface code—the addition of arbitrary extra facilities in the interface processor is not a simple task, or to be entered into lightly.

6.3.2 Limitations of the Processor Bank Servers

The major problem of the old style servers is that the interface processor is not easily programmed and that, viewed from the main processor, it appears as a 'black box' which provides a fixed set of ring handling primitives. These primitives are well suited to data protocols but not to voice stream handling—they cannot respond fast enough to support the transmission or reception of even one ISLAND voice stream. An implementation of the voice provider needs to lie above the basic transmission and reception primitives at a much lower level than that of the TRIPOS Ring driver seen by the main processor. There needs to be a way to bypass the standard data handling procedures, having distinguished voice and data traffic at a very low level; this can be accomplished only if code can be added to the interface.

Figure 6.1


Main bus (VME bus in ISLAND servers)
memory and peripherals accessible by both processors
Main Processor
Interface Processor
control
signalling
memory and peripherals accessible to interface processor only
ring handling code resides here to conserve VME bus usage
ring mapped so as to be available only to interface processor
this ensures control over main processor and conserves VME bus usage
Software Structure
applications level
basic ring commun-ication
commun-ication with main processor*
kernel
Software Structure
applications level
commun-ication with interface processor*
kernel
Key:
●complete diagram represents a dual processor server
●discounting features in italics and broken lines shows a single processor server
*handled by the kernel in the last scheme set out in section 6.3.5

Part of the reason why the Ring interface processor is not easily programmed it that is a much lower-level object than the main processor. The newer servers in the processor bank are based on Motorola 68000 main processors running TRIPOS and Motorola 6809 interface processors running assembly code. This means that software cannot easily be moved between the processors. Also the 6809, being an 8 bit rather than 16 bit machine, cannot directly access all of the address space of the 68000. Access to this space is therefore accomplished using specially designed DMA hardware which is complex, expensive and obscure in structure compared with a 68000 used as interface processor.

6.3.3 New Server Hardware

For ISLAND servers it was therefore desirable to move away from the idea of a main processor with a small assisting processor to handle Ring traffic. ISLAND servers should contain one or more processors, all of equal stature. The processors should all run the same operating system and should all be programmable in the same way, so that code would be easy to move from one processor to another. Since the servers would have the Ring as their only standard peripheral, facilities for debugging and controlling the whole server from the network would still need to be provided. This might best be done by preserving the feature of exclusive access to the Ring from one processor and also retaining the ability of that processor to maintain control over the other(s). This processor would always contain the code to drive the Ring, plus any code which needed to interface to the Ring driver in a very tightly-coupled manner.

We built servers with one and two processors for ISLAND use. Their detailed design is not relevant here, but some architectural features will be mentioned. The servers used Motorola 68000 processors, on cards linked together on the industry standard VME bus [VME 82].

There are applications for servers having one and two processors: in the dual processor types, one processor would handle ring i/o and functions tightly coupled to it whilst the other performed the server's 'true function'. A potential dual system application is the conference server, in which the interface processor might drive the Ring and run the voice provider, whilst the main processor did the work of combining streams and of handling communication with GOD. In other servers there may be less useful parallelism in the server's function or the whole service might consist of performing ring transfers. The latter is true of the translator, whose purpose is to accept one style of packet, buffer such packets and then retransmit the data in another format. Control is a very small part of the operation. Other than by using two separate processor-driven ring interfaces in the server (one for reception and one for transmission for example) there is no obvious parallelism to make the use of two processors clearly useful.

It was also required that the single processor systems could serve as the basis of the phones. The fact that both dual and single processor systems were to be produced and that common software (such as the voice provider) would be run in them emphasised the usefulness of having only one processor type and only one operating environment running on it. The structure of the servers is shown in figure 6.1. If all broken lines and italic text in the figure are ignored the structure seen is that of a single server. The complete figure represents a dual server. It will be seen that for the dual servers the Ring is a local peripheral of only one processor. As stipulated above, the interface processor therefore has complete control over the other processor. Each can interrupt the other for inter-processor communication, but the interface processor can also halt, resume and reset the main processor. The interface processor has full access to all of the main processor memory, which is all sited on the VME bus. However the interface processor can also execute code from its local memory. This gives the same advantages for debugging and of bus availability for the main processor as was the case in the processor bank servers.[1]

6.3.4 The Reef System

The Reef system is the version of TRIPOS implemented for single processor ISLAND servers. Insofar as it differs from previous versions of TRIPOS servers, its design was the work of the author. TRIPOS and its Ring Driver were ported by Jay Kistler, as a Computer Science Diploma course project. The Ring Driver drew heavily on software from the Atoll system, which is described below, and was developed on top of the RIP (Ring Interface PROM) firmware produced by the author.

The aspect of the Reef relevant to this discussion is the way in which the Ring handling software is layered. Instead of being a 'stub', as found in the processor bank servers, the Reef TRIPOS Ring device contains all the code equivalent to the interface processor of a processor bank machine.[2] That code accepts requests to transmit data around the Ring and to await reception of data across it on particular reception 'port'[3] and transmitting station combinations, placing the received data in nominated buffers. The code notifies TRIPOS when requests are satisfied, fail or time out.

The device in the Reef invokes similar functions provided by the RIP. It is the RIP which actually handles transmission and reception of data across the Ring as well as the queueing and dequeueing of descriptors of expected receptions. These descriptors specify how the received data should be handled. The Reef Ring Driver layer on top of the RIP combines RIP facilities and handles the timing out of requests, to provide higher-level primitives to drive the Ring for data applications.

The reason for making this split and for defining the RIP interface is so that other software can access the RIP directly and build upon it another style of access. The prime candidate to do this is of course the voice provider. Both the Reef and the voice provider call the RIP to transmit packets and to queue requests for packets to be received. The facilities which the RIP interface provides are

i) queueing reception requests according to port number—the Reef uses one set of port numbers and the voice provider another. The RIP looks for a suitable reception request each time it starts to receive a packet from the Ring: it is at this very low software level that the handling of voice and data can be separated into environments appropriate to each.

---

[1] The ISLAND servers are based upon standard commercial VME processor cards. Since the original server design was implemented processor cards have become available offering dual ported memory (locally accessible from on-board and globally accessible from the VME bus). Use of such cards would enable more than two processors to be used in a server as sharing bus bandwidth between them would no longer be a problem. Retaining the Ring as a local peripheral of one processor would still be of benefit since this enables debugging and control of the system via the interface processor.

[2] at least equivalent to the simpler version of bank interface code, which does not handle a stream protocol

[3] the term for a logical sub-address on the Ring

ii) mid-packet 'upcalls'. It was noted above that reception requests are queued by the RIP in the form of descriptors. These descriptors can specify routines to be called on success or failure of reception and also after the reception of some number of data bytes. The routine called after a given number of bytes has been received will usually compute where in memory the remainder of the packet should be placed. In the case of a voice packet, the head of the packet will specify a sequence number: using this the voice provider can calculate where to put the data from each incoming packet. The use of an upcall for this calculation produces greater efficiency—the voice samples can be written straight into the correct place as they are received instead of being copied after the whole reception.

These two features, together with a simpler routine to transmit a packet, are the RIP components which the voice provider requires.

6.3.5 Dual Processor Software

The first TRIPOS system that was implemented on the ISLAND servers was the Atoll, principally the work of Jeremy Bennett. This dual processor system closely mirrors the processor bank TRIPOS. In particular the interface processor cannot be programmed. Inter-processor communication occurs between the Ring Driver 'stub' (a TRIPOS device) in the main processor and assembly code in the interface processor, which provides the same functions as the Ring Driver in the Reef.

Only a little of the work on producing a better environment for dual servers has been completed. The intended plan will be outlined here. It can best be explained by analogy with the two varieties of processor bank system currently in use [Garnett 83]. The simpler processor bank systems contain a TRIPOS Ring Handler *task*, which converts ring i/o requests into a low level format suitable for sending to a Ring Driver *device*, the stub in the main processor which sends requests to the interface processor. The more sophisticated version of interface processor software provides all the functions of the Ring Handler: for systems using this version the Ring Handler in the main processor is simply a stub device. This version also has an additional level of sophistication: the interface processor handles a stream protocol. The main processor thus loses its stream protocol handler task and instead has a stub device for this also. In fact the stream and ring handler stubs are the same device—the requests sent to it bear separate sets of request codes to distinguish the two handlers' functions.

To convert the Atoll into a server suitable for voice applications, the interface processor software should first be replaced by a Reef. The Atoll would use a Ring Handler stub and its Ring Handler would run as a normal Reef task.[4] The Reef would have a device to relay requests from the stub to the Ring Handler. The stream protocol handler could equally well be moved from Atoll to Reef, as could other standard tasks, such as the Filing Machine interface task. Which tasks ran on which processor could be varied in order to balance the two processors' loads.

[4] This Ring Handler would not be the same as the Reef's own handler—practical details would require a conversion of TRIPOS message formats. Either there would be a converter task or, more likely on grounds of efficiency, two similar, complete and independent Ring Handler tasks would run in the Reef, each invoking the Reef Ring Driver.

This Reef/Atoll combination would be suitable for voice services. The time critical voice provider could run in the Reef just as in the single processor systems. It might well be manipulating buffers on behalf of a client task in the main processor—the loose coupling of the voice provider to its clients becomes important in such a case. A simple Reef/Atoll combination could make the task of moving code between processors much easier than before, but it would still not be trivial. The separate task and address spaces of the two processors mean that the details of how two processes interact will depend on whether or not they run on the same processor.

The second part of the software restructuring planned is aimed at making the movement of tasks between processors effortless: the Reef and Atoll TRIPOSes would share a single task and device number space.[5] TRIPOS has a fixed number space for both tasks and devices and a convention could easily be established as to which numbers corresponded to each kernel. Instead of there being a pair of 'stub' devices to communicate between the processors, each TRIPOS kernel would send a message to the other one only if it was for one of the other's tasks or devices. Since not all memory used by each of the processors would be accessible to all tasks, there would need to be a separate free store chain in global space for allocation of buffers and TRIPOS messages to be sent between processors. However this would be the only way in which tasks would need to be different if they might communicate with other tasks in another processor. The movement of interacting tasks between processors would now become trivial.[6]

The RIP provides configuration information so that an Atoll/Reef or a lone Reef processor can know the constitution of the server in which it runs: by exploiting that information the single Reef, the dual Reef and the Atoll kernels could be made identical. In the single processor case the Reef would not attempt to send messages to another processor and would use a single free store chain for both local and global free store requests.

6.4 Loading and Booting Servers

In addition to mechanisms for transmitting and receiving data across the Ring, the RIP provides several ancillary functions needed to support the development and running of TRIPOS and other systems in the ISLAND servers. The functions used to load and reload system images into servers are of relevance to the ISLAND infrastructure. Following the practice in the processor bank and its supporting servers, several of the RIP's facilities are provided by a stub in the server's PROM itself plus programs in a remote machine. The RIP is written in assembly language, as is required for most of its stub-level functions. Remote programs are written at a higher level. For example the RIP provides a debugging stub, which handles processor exceptions and allows a remote system to read and write memory and registers; to perform block moves with local validity checking and to single step or resume code execution. A remote high-level program uses these primitives to provide single line assembly/disassembly, assembly level stack tracing and source level debugging for example. Similarly the RIP provides a stub to enable software to be loaded into a server from a remote machine, using a modified form of the UNIVERSE loader

[5] The hardware clock might well be duplicated and would then have the same device number in each processor.

[6] For example, only one Ring Handler would be needed for clients in both the Reef and the Atoll, because request packet formats would be the same for all clients. The global task/device numbering scheme could be extended beyond two processors provided the servers were equipped with global memory.

protocol [Wilbur 83], and to force either processor to be halted, resume execution or be reset. These facilities are used in a number of different ways.

In order to support ISLAND applications, the way in which servers are loaded and controlled has to be different in several respects from the processor bank style [Needham 82]. In the processor bank, the interface processor is regarded as a stable entity. When it is powered up, it approaches a bootserver and requests its interface code to be loaded. That code contains all the facilities for loading and controlling the main processor and also provides the ring handler/driver facilities that the main processor will require. Requests made by users to load system images into the main processor result in a 'Resource Manager' server interacting, through an intermediate server known as an 'Ancilla', with the interface processor to halt the main processor, load its code and then reset/start it.

For ISLAND the whole of a dual or single processor server should be loaded as one system image. In the case of a dual processor server running an ISLAND application the twin operating systems will each have applications tasks and drivers as part as their initial load images. It follows that the two must be loaded together. In the case of a single server there is only one processor to load. Since the applications to be (re)loaded may run wild, the code to halt them and to load a replacement system should be in ROM. It is therefore provided as a part of the RIP. (The RIP does need a small amount of RAM as its working space. However it is much less likely that a rogue program would destroy this working space than that it would subtly or completely affect a wholly RAM-based RIP. As an example, to invoke the RIP's catch-all debugger requires only the single exception vector concerned to be intact—the RIP then sets up all the working space that it needs to contact a remote debugging machine.)

Dual ISLAND systems have ROMs in both main and interface processors. The RIP, in the interface, is able to determine when a machine is powered up how many processors there are and whether there is any global memory.[7] The VSOP (VME Second processor Origin PROM) in the main processor is a very much smaller piece of software which simply enables the RIP to control the main processor where it is present. On startup, the VSOP initialises the main processor. It then tests to see if there is a system loaded for it to execute. If there is such a system loaded the VSOP will enter it, otherwise it will halt awaiting further instruction from the RIP. (The RIP can halt, reset, interrupt or resume the main processor via the VSOP and simple hardware.) When the RIP starts up, it resets the VSOP and during VSOP initialisation can guarantee to set the VSOP system start vector to 'no system loaded'. Thus the main processor will be guaranteed to be inactive whilst the RIP is loading a system for both processors to execute. When a dual processor system has been loaded by the RIP the interface processor immediately starts execution. It typically initialises all of its own data structures and then calls the RIP to reset the main processor, which now finding a loaded system will enter it.

The control primitives in the RIP were designed and implemented together with an 'ISLAND Ancilla'. When a new system must be loaded into a machine already running, the ISLAND Ancilla requests the RIP to

i) halt the main processor (a null routine where the RIP knows there is no such processor)

[7] This is the basis on which it can supply information to a TRIPOS system about the server configuration, as mentioned above.

ii) stop interface processor execution and await a request to be loaded.
The Ancilla then fires up a loader. The system when loaded will start executing in the single/interface processor, starting up the main processor where present just as above.

The RIP loader stub allows machines to be loaded in 'either direction'. Loading may occur
i) when the machine is started up, initiated by the RIP
ii) when the machine is running, initiated by an application running in the server, which calls the RIP with a request for loading
iii) initiated by external systems, when permitted to by software running in the server. (The server can limit loader privileges required for invocation from outside.)
iv) initiated by the Ancilla, which halts the server by using RIP primitives and then interacts with the RIP to set suitable loading privileges for use by the loader which it is about to fire up.

Combinations of these possibilities can be used to preload a partial system image and then add further components before running it. For example suppose that all ISLAND servers run under the same Reef/Atoll combination. A complete system image could be linked for each server, containing both the operating system and the server-specific software. Alternatively when the server is not in use it can be loaded with the common parts of the operating system. The code entered when this preliminary load is complete would simply set the external loading privileges ready to accept some more code and then wait for a second load of server-specific portions of the system image. (Use of an altered loading privilege will prevent a 'second load' into a server which has not been suitably 'primed'.) The advantages of this are simply speed benefits: loading a system takes some time. We discussed in chapter 4 the requirement to be able to start up new servers very fast. If most servers run a common base system then all of the common base code can be preloaded. When one server fails and another has to be started up in its place, GOD can then cause just the applications code to be loaded in and the whole system to start execution.

6.5 An Example Application—the Conference Server

To complete this chapter, implementation of the conference server is discussed. This is an example of a service for which the ISLAND servers were designed. The important parts of the conference server are the voice provider and a set of routines to perform the conversion of input streams into output streams. The voice provider and its use by the conference server have been discussed. It was noted in chapter 3 that a multiple stream voice provider has not yet been implemented. Conversion routines, for the cases of three and four party conferences, have been designed by the author and implemented by Stephen Marshall as a Computer Science Diploma project.

It is clear that to obtain good performance from the conversion routines they need to be written in assembly language: careful and ingenious use of instruction set features coupled with data structures tailored to the instruction set will enable the server to use minimal processor time per conference. The three and four party conference handler routines implemented do not represent a large amount of code, consisting of 300 and 600 assembly instructions respectively, plus comments.

The routines are written to be invoked from a Modula-2 program: their use of registers is compatible with that by the TRIPOS Modula-2 run time system. When GOD sets up or closes down a conference call, a process in the conference server will create or delete the buffers, voice provider channels and a pointer for that call. The Modula-2 program repeatedly inspects all such pointers and invokes the conferencing routines with parameters to indicate what calls are in progress and how many participants are in each. Each call and the position so far reached in processing the voice buffers for that call can together be specified using a single pointer. The conference routines will process a few packets' worth of every call and then return control to the Modula-2 program, which will check the data structures and repeat the whole sequence.

In order to maximise the processing speed of the conversion routines, the circular buffers supplied to the voice provider for each of the streams in a conversation are of a fixed and equal length and must also be located at a fixed offset from one another. This means that one pointer suffices to specify both a call and how far processing of that call has proceeded (using the Modula-2 program to perform wrap-around). This is also tailored to exploiting the fastest applicable addressing mode ('constant plus register contents') offered by the 68000 for accessing voice samples. For each phone connection, the same circular buffer is used for both reception and transmission. (The pointers used by the voice provider for transmission and reception will be different but will both point into the same buffer.)

As discussed in chapter 4, the conference server combines voice streams by performing addition and subtraction of PCM-encoded values using lookup tables. Use of lookup tables implies that these operations will be diadic—triadic operations would require extremely large tables. The effect of the conversion routines is that every telephone is sent the results of combining the outputs from all other phones in the same conference, but not its own output. A very simple way to combine voice streams is to add pairs of samples repeatedly to produce each output. For each output in an n-way conference n–2 additions would be performed, so that $n(n-2)$ additions would be required in all. If one subtotal may be stored during the calculation, an alternative method is to add all samples (using n–1 additions) and then obtain each output by subtracting the corresponding input, giving a total of $2n$–1 operations. The following table compares the number of operations used in each of these approaches for n = 3 to 8.

Number of Talkers n	Add only method $n(n-2)$	Add and subtract $2n-1$
3	3	5
4	8	7
5	15	9
6	24	11
7	35	13
8	48	15

The second method is faster for n>3. Of course if intermediate sums could be saved when using the first method fewer steps would be required. For a general value of n the optimal order of combining samples to take advantage of this cannot easily be determined. For the case of n = 4 however this order is simple and requires six additions—less than either of the figures in the table above.[8] Therefore addition only is used in these implementations of the three and four party combination routines.

The task of the conference server is not quite as simple as repeatedly combining samples in regular patterns. The voice streams sent to the server have silence detection flags denoting which speakers are active. For most of the time in an audio conference there will only be one speaker active at once. In such a case the server should not add together the background sounds of all inactive speakers, for when combined they may well represent a significant disturbance against which the active speaker will not be so easily heard (particularly if more than one participant in a conference is in a noisy room). Therefore for one active speaker all other participants should hear that speaker and the speaker should hear only one person's background noise. For two speakers each of the speakers should hear one another and the remaining participant(s) should hear the two sounds added. For more speakers the sequence continues logically from there.

The other problem which the conversion routines have been implemented to handle is that at any time a packet on one of the incoming voice channels may have been lost/delayed. To detect this, the compounding routines test whether the silence word for each voice packet is set to *true*, *false* or a third value defined by the voice provider, indicating 'no packet arrived'. That third value is written into the buffer after each packet is transmitted from the buffer. The compounding routines detect this condition when chosing a background sound to send out to a single current speaker or when there are no active speakers, so that they do not chose some invalid data which may contain a large glitch if played in isolation. In the (unlikely) case where none of the buffers contains incoming packets, all outgoing buffers are filled with D.C. level sample values.

The three and four party conversion routines use a selection tree to determine which streams in a conference are currently active and which inactive and to chose a routine suitable for them. The tree walk takes inactive streams and missing packets to be equivalent and then cases of missing packets are handled separately where necessary. There are 8 and 16 leaves in the 3 and 4 party trees respectively. Two alternative methods were considered for implementation of the compounding routines called after traversing the tree. At each leaf of the tree registers could be set up to point to the different buffers and then one of a few generic routines would be called at the end of the tree walk (such as a generic routine to handle 1 active, 3 silent streams). Alternatively a routine could be provided for every leaf. The former course seemed most attractive initially. However the details of implementation on the 68000 meant that the second method ran faster—the addressing modes required in the first case turned out to be considerably slower. The routine-per-leaf approach called for duplication of code, but macro expansion facilities were available to duplicate it quickly and without introducing errors.

There are many other ways to implement the handling of sound and silence in addition to this selection tree approach. It was used since it was the fastest method of all those considered. The speed of the conversion routines as implemented by this approach clearly depends on the number of active speakers. The slowest case is where all speakers are active, as might be expected. Processing 2ms of a fully active conference call takes the 68000 just under 0.3ms and 0.5ms for the cases of three and four party conferences respectively.

[8] The order is to form sums of inputs $s_1=i_1+i_2$ and $s_2=i_3+i_4$ and then to form outputs $o_1=s_2+i_2$, $o_2=s_2+i_1$, $o_3=s_1+i_4$ and $o_4=s_1+i_3$.

7. Ring Protocols and Hardware Support

This chapter considers the lowest level of Ring protocols required in order to support the infrastructure and the applications set out so far in this dissertation. It then discusses hardware for efficient implementation of those protocols.

Most of the discussion on protocols so far has concentrated on transporting voice on an Integrated Services LAN, for this is a subject which has called for new work. A tacit assumption has been made that the LAN has the properties expected of a conventional computing LAN and that transport of data across such a LAN is well understood. The level at which the *combination* of voice and data, as opposed to protocols suitable for each, must be studied is at the level of packet transport across the network. Means must be found to guarantee low delay to voice traffic at the same time as high performance for data. If the need to support high bandwidth image traffic is also added, the problem is compounded further.

This chapter starts by identifying potential problems: it reviews the natures of voice and data protocols and then considers how a network interface will handle them. Most of the performance problems concern the high speed sorting by a network interface of incoming packets. These problems are compounded in the case of a mixed traffic interface that has to handle two very different types of traffic—high speed bulk data transfers and frequent small voice packets. The shortcomings of various existing types of interface are discussed in this context.

A packet transfer protocol is proposed, including a criterion for sorting incoming packets. A software implementation of the reception procedure is outlined. We then propose a hardware support scheme tailored to this protocol, to increase throughput of the network interface. Its applicability to the Fast Ring in general, as well as to Integrated Services in particular, is discussed. The hardware design was intended to provide only very basic support for critical parts of the protocol and to support variations in the design of protocol: the chapter ends by considering the extent to which flexibility of protocol is possible.

7.1 Problems of Data Protocol Interfaces

In computing LAN systems, performance of network interface hardware has long been a limiting factor on performance of the whole system. Whilst there have been LANs giving around 10Mbs^{-1} of bandwidth for about a decade [Metcalfe 76, Wilkes 79], few interface nodes perform within an order of magnitude of that level. For example the Logica VMI-1 interface [Logica 81] claims to handle the maximum point-to-point bandwidth of the Cambridge Ring, but does so only in half-duplex mode and under very specific traffic conditions. The nature of computer protocols makes a high performance interface hard to design. It is possible to achieve significantly higher bandwidth by using very different protocols—predominantly circuit-switched traffic does not require much intelligence per switched byte and hence will give better throughput.

This problem of performance of intelligent interfaces is emphasised by

i) the increase in bandwidth available from LANs and the desire to use a significant proportion of that bandwidth in a distributed computing context

ii) the emergence of Integrated Services systems such as ISLAND, where there is a need to handle a variety of protocols simultaneously in one interface. For example, the integrated workstation must interact with the apparatus of a distributed computing environment, so that its interface must support flexible and intelligent switching. At the same time if the interface is to support either video or concentrated voice traffic it must provide high throughput. The translator is another example from ISLAND where this problem will also be felt. The *simultaneous* combination of requirements is the greatest problem for the interface designer.

Protocols for voice have already been discussed in some detail. Their features of interest here are that they are lightweight and involve streams of frequent, small packets of voice samples. Although video traffic is of interest it is a considerable subject on its own and will not be dealt with here. The main topic that remains to be considered before Ring protocols suitable for Integrated Services are proposed is data protocols and the workings of interfaces that handle them.

The two basic types of data protocol were outlined in section 2.1. These are

i) the single shot protocol, in which a request goes out from one station to another and a reply returns along the same route. Although if no reply is received the request may be re-transmitted up to some prescribed number of times, the protocol can be regarded as a single exchange and from the point of view of the interface is not associated with any other requests for network transactions

ii) the reliable data stream protocol, in which two stations send one another a series of packets in order to transfer a quantity of data correctly and exactly once. All activity on a single stream can be regarded as part of the same transaction from the standpoint of clients' requests for network i/o.

An Integrated Services packet level protocol must support these two types of protocol and must also support unacknowledged streams for voice and other lightweight uses (such as that outlined in [Waters 84] and discussed above in section 2.3).

7.2 Demands on the Network Interface

These protocols place most demand upon the receiving side of a network interface; transmission is fairly straightforward. The client of the network interface is generally, as in the case of the ISLAND servers, some computing engine running for example a multi-process operating system. At any time, various processes in the client may be expecting to receive data across the network. Such receptions can be divided into three categories:

i) a client process offers some service (e.g. it may offer to field SSP requests as to what the time is). The service will be advertised to customers by a nameserver. The process can expect requests for its service from anybody at any time. It will normally have some buffer available in which to receive incoming requests from the network. It will have notified the interface of a logical sub-address within the node on which requests will appear and of a buffer available in which to place the next request received.

ii) a process has sent a request to some SSP service and is awaiting a reply. The request packet will have specified some logical sub-address for the reply: the interface will have been notified of the sub-address and of a buffer as above.

iii) two stations have set up a reliable or an unacknowledged stream. Associated with the stream are a series of reception buffers allocated by the client process. The client will have notified the interface of the expected sub-address. The buffer into which data at the tail of any given stream packet should be placed may well depend on the sequencing information in the head of the packet. (This is true in the case of the voice provider. It is also generally true of reliable data stream protocols.)

In all of these cases, a packet should not arrive unless there is a reception request for it from a client process. (If one does, it should be discarded.) Also, at least in the first two cases, a buffer location for the data is known when it arrives. For efficiency, data should be placed straight into the correct buffer as it arrives.

The interface will maintain a reception request list. When the start of a packet is received its header information, for example source address and destination sub-address, will be checked against the requests. Once a suitable reception request is identified the data will be transferred into the associated buffer and then the client process in question will be awoken or otherwise notified of the arrival.

For a stream protocol, in a simple system the whole of each packet (header and data) may be transferred to a standard buffer. Some stream handler process will then be awoken to decide from the header information upon another buffer into which to copy the tail, before awakening the process which is actually consuming the stream data. A more efficient way to handle stream reception is for the interface to be supplied with an upcall routine together with the reception request. In the description of the RIP it was explained how the interface receives the header of an incoming packet and calls a supplied routine, the purpose of which is to determine where the interface should then place the remainder of the packet directly. This mid-reception interaction will clearly make stream handling more efficient. (This is the approach used in the more complex processor bank systems described in section 6.3.5 [Garnett 83]: the simpler approach corresponds to the way in which TRIPOS systems using a stream handler *task* behave.)

This may all seem unnecessarily complex: the same efficiency could be achieved far more simply if the interface were to keep a series of buffers of its own. (This approach is used in one commercial telecommunications research project.) Each packet received would simply be placed in the next available buffer. After reception was complete the reception request list would be searched for a client, which would be passed the address of the relevant buffer.

This scheme is simple and attractive at a low level, but is incompatible with many operating systems and programming environments. If user-supplied buffering is not used directly, as expected by higher levels of software, data must be copied from the interface buffer pool into client buffers, with an obvious performance penalty. (A general-purpose programming environment may support i/o requests to read a very large amount of data into a portion of the heap representing an array: that array will be assumed by the client to be contiguous and the data must be delivered in a manner reflecting this.) An equally strong objection occurs where the data reception requested forms part of a video stream destined for a framestore buffer. Although this dissertation does not deal with video in any depth, chapter 9 will discuss design of a workstation terminal. One of its properties is that the video display framestore is directly accessible to a network interface at full bus speed. It is important that images can be transferred from network to the correct portion of the store directly: a requirement for additional copying would affect performance intolerably.

From the point of operating system structuring, it is undesirable for the event notification upcall structure to call up as far as the application requesting the Ring i/o, which is what would occur if received data were supplied to an application in buffers allocated by the interface. The upcall structure discussed above is intended to span only the layers between the lowest level Ring packet handler and the reliable stream handler, which should then communicate with the application consuming or producing data in terms of buffering declared by that application. Bad experiences with upcalls spanning right up to the application level are reported from the Swift project [Clark 85a], since all applications written in the system need to know an undesirably large amount about the i/o structure underneath them. There are however some circumstances where passing the address of a buffer up to the client is attractive: these will be discussed towards the end of this chapter.

Transmission of packets is of comparatively little concern in this discussion because it is serialisable: there is no equivalent to several reception requests outstanding which may be satisfied in any order. A simple 'get rid of this packet' interface is all that is required, although it is convenient to be able to specify that a packet lies non-contiguously in a series of buffers (e.g. which contain the header and data portions of a stream packet) in order to avoid copying data.

7.3 Examples of Typical Interface Hardware

A computing LAN will generally connect a series of computing engines. The architectures of both processor bank and ISLAND computing engines have already been discussed: these embody the characteristics of a large proportion of all computer network interfaces. The important characteristics may be summarised as follows. An engine may contain one processor, with network interface hardware mapped into its address space, as in the case of the ISLAND single processors. The interface software will run as a process or as a device (i.e. in a high level or low level language) as part of the processor's operating system. However many systems employ a peripheral processor specially devoted to handling the network interface on behalf of the 'main' processor. The main processor passes transmission and reception requests to the interface processor and receives in return notifications when requests are satisfied (or fail in particular ways). The interface processor may also handle for example the whole of a reliable stream protocol, the main processor seeing only the data which is layered on top of that protocol.

The interface processor may be either a general purpose microprocessor or a lower level engine such as a bit-slice system specially optimised for its limited set of functions. A general purpose microprocessor may be assisted by for example DMA hardware. The series of microprogrammable computing engines which implement the PrincOps instruction set[1] was mentioned in section 6.2. These systems run independent sets of microcode to support a general-purpose processing system and high speed peripherals, such as the network interface. This type of system is not an option for most designers since microcodeable systems are generally expensive to develop and maintain. However they look very little different for the purposes of this discussion from a general processor plus a bit-slice interface.

There are drawbacks to using either a high level processor or a bit-slice system to implement a network interface. In most computer programs a small fraction of the code can be identified which is executed for the majority of the time that the program runs. Network interfaces are no exception to this rule. There will be a small tight loop which shovels the data between memory buffer and network, and which runs for most of the time. The remainder of the code, which runs only once per packet transaction, will be that (e.g. for reception) to decode the packet header; search the reception request list; check the validity (length etc.) of the incoming packet; then when reception is complete to notify a client process of the arrival. Similar checking and notification will surround the tight loop for transmission.

A general purpose microprocessor will make the code for queue searching, header checking etc. fairly easy to produce but will run the data transfer tight loop very slowly. A microcoded engine can be designed to make the bulk transfer process very much faster. However writing the rest of the code needed for the interface as a microprogram is not only unpleasant to contemplate as a writer but also expensive both initially and for maintenance or upgrades.

The combination of a microprocessor and a DMA controller (DMAC) can sometimes solve this problem. Some CSMA LANs present a complete incoming packet in a FIFO buffer: the processor can decode the header information and then set up the DMAC to transfer the bulk data into an appropriate client buffer. Afterwards the processor notifies the client process that data has arrived. This will give a significant advantage where packets are large. This is not the case for the Fast Ring, where data arrives in a series of small slots, each of which needs to be decoded before the data can be transferred appropriately. The time to process the packet header in such an arrangement would dominate the transfer time for the (very small amount of) data, so that the simple addition of a DMA controller on its own will achieve very little. It has been established that a small packet size is essential for Integrated Services. We must accept as a consequence that the design of the network interface will be a harder task.

It was suggested in section 6.3.3 that the model of a server as a series of processors of equal stature, one of which drives the Ring, was more appropriate for ISLAND use than a main processor with a peripheral processor for ring handling. Whereas DMA facilities on their own would be of little assistance to the Ring driving processor, hardware which both decoded and sorted most incoming packets and then automated their transfer could be of considerable benefit if allied to a suitable protocol definition. The Ring handling processor would still run most of the interface code, but execution of the 'tight loop' would have been removed from it for efficiency. This is the approach which will be developed in the rest of this chapter. It has some commonality with the approach to network interfacing used in DCS [Mockapetris 77], in that it partitions the interface into a 'semi-intelligent' component to handle critical parts of a protocol and a more

[1] For example the 1108, 1132 and 6085 series machines by Xerox Corporation

general computing engine.

7.4 Access Procedure for the Cambridge Fast Ring

Details of both the Fast Ring and the older 10Mbs^{-1} Cambridge Ring [Wilkes 79] have already been given in this dissertation. The access procedure for both is the same. A transmitting station specifies a destination address and loads data into the Ring node. The receiving station may be interrupted when data arrives or poll for the event, and then reads both transmitting station address and the data. A receiver may select to listen to all stations, one station or no stations.

The older ring carries only two data bytes per slot; the new one carries 32. This is necessary, as noted previously, on grounds of performance when the data rate of the ring is increased.[2] It also improves the throughput of a network interface. During a transfer lasting a series of slots the interface must either (i) poll for the reception of another slot of data or the availability of the transmission buffer or (ii) be interrupted for each such event. In either case the larger amount of data accommodated in a slot decreases this overhead.

A more important advantage is that with 32 data bytes it is possible to implement a protocol using just one slot. In the older Ring, almost no protocol could be implemented in two data bytes, which is why the 'basic block protocol' [JNT 82] was designed, in which a train of slots is formed into a long data block containing header, sub-address, data and checksum. Interface performance dictates that it is not feasible for a station to be receiving a basic block from more than one address at a time, for otherwise each reception of two data bytes would require a search through the list of current reception requests.[3] A ring node may select to listen to only one transmitter at a time: this hardware mechanism is used to produce acceptable performance from the interface.

This same problem is not present in the Fast Ring. Studies of data traffic patterns [Temple 84] have shown that computer network traffic divides into a U-shaped distribution—large packets for example for file transfers and small packets for terminal traffic and simple requests (e.g. for the current time). Most packets will fit into a single Fast Ring slot. Nevertheless one could operate an analogous 'listen to only one sender' protocol for the duration of any packet, however long or short. The longer slot length, together with other improvements in the ring design [Hopper 86], would produce a much higher throughput than for the older Ring, given the same complexity of interface hardware.

[2] As well as by the destination delete schemes mentioned previously, performance could also be increased by allowing a transmitting station to use more than one packet per Ring revolution. As well as destroying the simple bandwidth sharing guarantees of the Ring, this would also not solve the problem currently under discussion. The network bandwidth available might increase, but the network interface could not act fast enough to make use of that bandwidth.

[3] A very simple scheme for digital telephones was once implemented on the older Ring [Leslie 81], in which to minimise the complexity of the phones they continually sent two data byte long voice packets to one other station—either to another phone or to the control server. (The phones had conventional dials and the digital interface was so simple that the controller received voice streams containing dialling pulses and decoded requests to call other phones by processing these 'voice' streams.) The severe limitation of this system was that it was impractical for the phones and controller to communicate using standard protocols with other stations on the Ring.

This approach is not suitable for the voice/data traffic mixtures to be encountered in the ISLAND architecture. Voice must be sent in small, frequent packets. These packets are of a length that will fit into one Fast Ring slot. Some nodes will be handling both voice traffic and longer packets simultaneously, the latter consisting of long sequences of slots. For these long block sequences, the exact transmission time and the queueing delay before transmission are not critical. For voice, transmission must occur within a very small range of queueing delay. (The juxtaposition of these two facts is illustrated by the functions of the translator.) It is therefore necessary that voice slots can be handled interleaved with the reception and transmission of long data blocks. It is probably sufficient to handle several single slot transfers during a single multi-slot block transfer, as opposed to handling many multi-slot blocks at once. If video, voice and data traffic are all to be mixed this may not be true (since video transfers in for example a video conference will need to be made in real time like voice but in large quantities and in long blocks like data) but in many cases it will be.

7.5 Candidate Protocol Structures

A block transfer protocol must be able to handle a variety of sizes of block, although we noted above that most will be either very short or very long. The block size used in a file transfer protocol is usually set by the network: the client generally has a large amount of data to transfer and divides it up into the smallest number of blocks that the network allows. If we design a multi-slot protocol with some fixed maximum number of slots per block, the client will split the data up into blocks which in turn will be sent each as a series of slots. This may seem to be an unnecessary layer in the middle: we could conceive of sending extremely large quantities of data as a single stream of individual slots (for example on the Fast Ring transmitting up to 1.75 Mbyte in one go using a very simple format) and eliminate the intermediate blocking level. A reliable block transfer protocol might entail sending requests at the end of the entire transmission for any slots not correctly received, this giving better throughput than conventional short revolving window number schemes. This the basic form of the ARPA Blast protocol [see Clark 85b].

This may appear attractive in that it is conceptually simple and elegant, removing an unnecessary intermediate protocol layer. However, apart from the sizes of buffers called for in some circumstances, it suffers in terms of efficiency. In a reliable sequenced data stream, the block is generally regarded as the unit of acknowledgement/retransmission. In the context of this work it is more significantly the 'unit of thought'. In the Blast-like scheme above each slot would bear a sequence number. Upon arrival of each 'packet' (which for the rest of this chapter we now define to mean a slot full of data, in accordance with Fast Ring parlance), the correct reception request would need to be identified and a buffer address computed based on the request and the sequence number. This overhead is liable to limit the throughput of a network interface built along these lines.

To avoid this requires the use of a multi-packet block, and in particular a block structure in which very little need be done per packet. The approach which will be put forward here confines nearly all protocol activity to the header and tail packets of a block, minimising the work of handling a mid-block packet. Since the problems of a high speed interface are largely at the reception end, discussion will concentrate there first. It will be assumed that the number of receptions of blocks greater than two packets in length (which will be termed 'long blocks') in progress simultaneously is normally small, although it may well be more than one. An efficient means is required to identify mid-block packets and associate them with one of the receptions in progress (or discard them otherwise).

This association will be made by checking some particular criterion or criteria: candidates include the

i) transmitting station address

in which case only one long block transmission can be allowed at a time between any given pair of stations. If each packet is tested for a 'mid-block packet header' then short blocks can be interleaved with long ones, thus satisfying the voice/data transmission concurrency requirement.

ii) address of and (logical) sub-address within the transmitter

would remove the above restriction, at the expense of one more parameter to check per packet. (Looking for a match with one of several variable sub-addresses will almost certainly be more costly than testing for a fixed mid-block header.)

iii) sub-address within the receiver

The attractive feature here compared with the schemes above is that the sub-address can be allocated by the receiver on a per-block basis. The receiver therefore has control over the values of sub-address it will be expecting to test for. This is significant because the receiver will need to look up the sub-addresses it is presented with at high speed. If *it* allocates them then it can ensure that they are well ordered, forming the indices of a small array for example, so that lookup is a simple operation. If the numbers to be looked up are a function of the transmitter instead then the numbers which are valid at any time will form a large and sparse array. Checking will be slower, at least as a software operation.[4]

It should be noted that this option requires that a transmitter always be allocated a receiver sub-address before it can send a long block: this is probably obtained by a single shot exchange with the receiver, consisting of single packets in each direction.[5]

iv) receiver sub-address and transmitting station address

This removes the need for a preliminary exchange before all long block transfers. Logical sub-addresses may for example be obtained from a nameserver. Of course two transmitters may happen to use the same sub-address for a long block transfer simultaneously. This option again involves checking against a sparse array, requires extra checking per block and has none of the advantages of the previous case.

[4] A hardware implementation can use associative stores, but this entails more hardware complexity than e.g. accessing an element of a small array.

[5] The Fast Ring design suffers from certain receiver contention problems. Discussion of these problems is not relevant here but receiver-regulated timing of long block transfers might well alleviate them.

The first and third schemes are the most attractive. The first will be explored in this discussion since whilst imposing some restrictions it is simple and self contained, unlike the third which assumes the need for a supporting level of protocol.[6] It was an aim of the interface hardware design to be described in this chapter that it should be flexible enough to support several protocol schemes: it will be seen below that it does indeed support all the options set out above.

7.6 A Protocol Definition

In accordance with the discussion above we will define a block structure thus:[7] the 32 bytes of data space in a packet are used as follows.

byte 0:	bit 7	set if the packet is the start of a block [SOB]
	bit 6	set if the packet is the end of a block [EOB]
	bits 5–0:	tag (explained below)
byte 1:		continuation of tag
	(bytes 0 and 1 constitute a 'packet header' dibyte)	
bytes 2–3:		length of block data in bytes if SOB, user data otherwise
bytes 4–5:		sub-address in destination if SOB, user data otherwise
bytes 6–31:		user data

16 bits are assigned to the length field a little arbitrarily, but it is unlikely that blocks longer than 64kbytes will be required. (In particular, the block will be the unit of retransmission requests.) Since the length is in bytes, some of the data space in a packet with EOB set may not be used.

It should be obvious from this definition how blocks containing one, two or more packets will be composed. The purpose of the tag field is to avoid error conditions of the following type. A transmitter transmits two consecutive two-packet blocks to the same station. The network loses both the second packet of the first block and the first of the second. Without the tag, the receiver will believe it has received one valid block. The tag is simply a revolving number incremented by the transmitter each time it sends a block longer than one packet. It is assumed that the tag numbers will not be recycled before the receiver would time out an incomplete block. Otherwise a larger field needs to be assigned. For each byte in the head and tail packets used for protocol information, however, the number of bytes of data which can be sent other than as a full long block is reduced: it is very desirable to maximise this number.

Tags are not required or used in packets where neither or both of SOB and EOB are set: in these cases the tag field is defined to consist of all zeros. Note that there is no checksum field in the block structure: the Fast Ring handles checksums at the slot (i.e. hardware) level. This is a major advantage of the new Ring over the older one: software implementations of the basic block protocol are estimated to spend 40% of their time on checksum calculation [Swindon 83].

[6] The third scheme is preferred by other researchers working on the Fast Ring for different styles of use, under the auspices of the Alvey funded 'Unison' project.

[7] This block structure was defined as a result of brainstorming sessions involving members of the University Computer Laboratory and Acorn Computers Ltd., in which the author was a participant.

7.6.1 Software Implementation of This Protocol

Transmission is not a complex issue: it need be noted only that single packet block transmissions must occur immediately they are requested, despite any long block transmissions already in mid-block. An implementer of the protocol may or may not decide to interleave long block transmissions: if so, the transmitter software must prevent two transfers being interleaved to the same destination.

Reception is more interesting. At the 'start of day', reception requests are queued up by clients just as explained in section 7.2. Behaviour of the interface can then be considered in terms of the arrival of a

i) <u>packet with SOB set</u>

A reception request that matches the packet will be identified. Checks e.g. for sufficient buffering to accommodate the data in the incoming block will be made. The data in the SOB packet will be copied into the buffer. If EOB is set then the client process requesting the reception will be notified. Otherwise, a 'block reception in progress' descriptor will be set up, containing
  the transmitting station address
  the number of bytes of data outstanding
  the position in the buffer where the data in the next packet should be inserted
  the tag expected in the last packet
  a pointer to the reception request concerned.
A timer will be started, to signal errors for blocks not completely received within some total time.

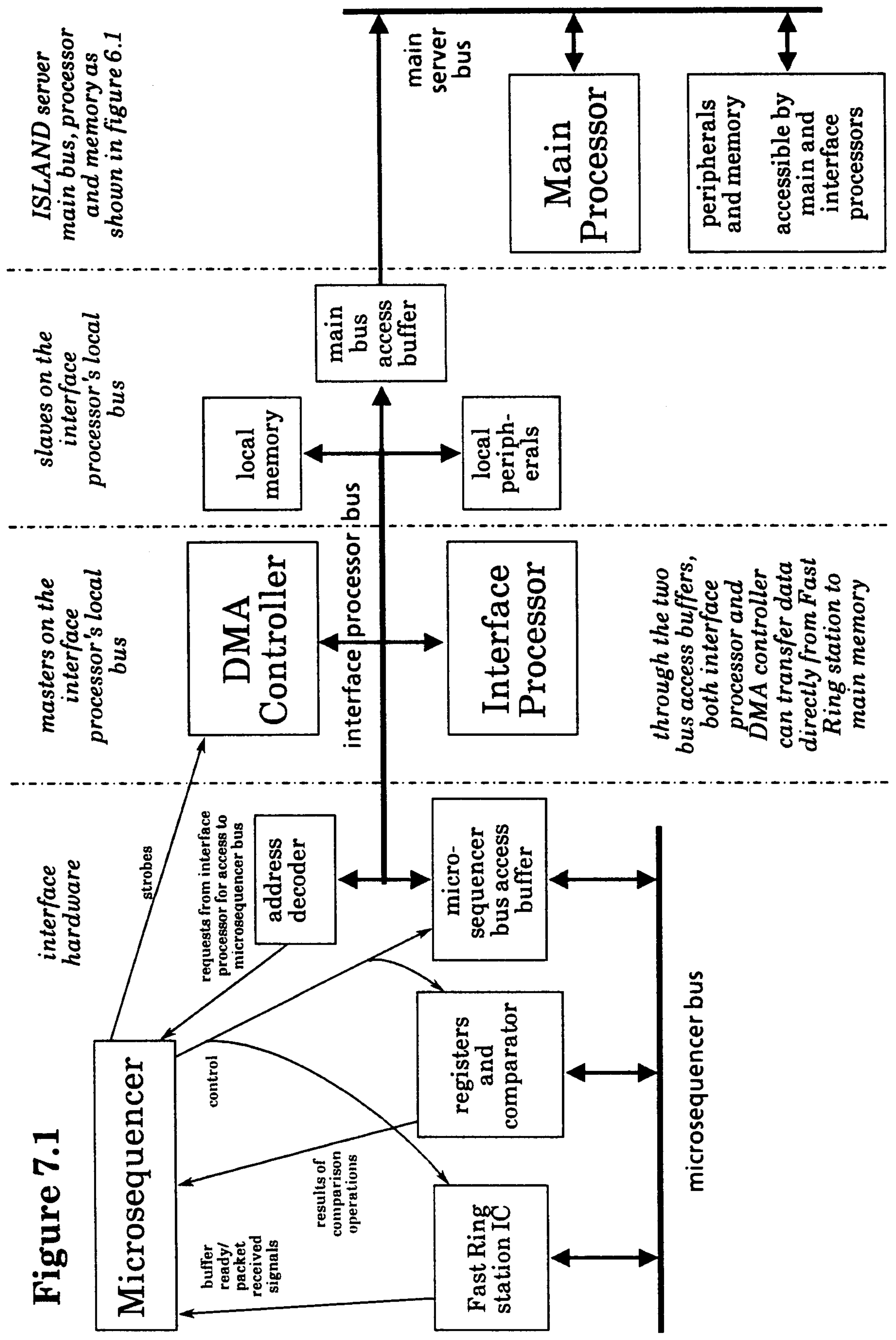
ii) <u>packet with neither SOB nor EOB set</u>

The 'block reception in progress' descriptors will be searched for a match of transmitter address. This can be done fairly efficiently, assuming that relatively few long block transmissions will be in progress at a time, with a fixed size array of descriptors, overflowing into a linked list as occasionally necessary. When a match is found, all that is required to handle the packet is to adjust the count of bytes of data outstanding, checking that it does not fall below one as a result of receiving the packet, insert the data into the buffer at the current position and then adjust the buffer pointer.

iii) <u>packet with only EOB set</u>

A 'block reception in progress' descriptor needs to be identified here also. The number of bytes outstanding must be checked, in case any mid-block packets have been lost, and the tag must also be checked. In the absence of errors the valid data from the packet will be transferred to the buffer. Success or an error will be signalled to the client and the 'block reception in progress' descriptor will be cleared down. (Timeouts and other mid-block errors clearly lead to error signalling and the clearing down of the descriptor in the same way.)

Figure 7.1


ISLAND server main bus, processor and memory as shown in figure 6.1
slaves on the interface processor's local bus
masters on the interface processor's local bus
interface hardware
main server bus
Main Processor
peripherals and memory
accessible by main and interface processors
main bus access buffer
local memory
local periph-erals
interface processor bus
DMA Controller
Interface Processor
through the two bus access buffers, both interface processor and DMA controller can transfer data directly from Fast Ring station to main memory
strobes
requests from interface processor for access to microsequencer bus
address decoder
micro-sequencer bus access buffer
registers and comparator
Microsequencer
control
results of comparison operations
buffer ready/ packet received signals
Fast Ring station IC
microsequencer bus

7.7 Hardware Support For This Protocol

The software realisation above was outlined mainly as a basis from which to suggest how hardware of modest complexity might strategically be used to produce a much faster interface. The obvious region of the protocol to consider is the handling of mid-block packets, both in transmission and reception. The hardware should identify and handle fully all mid-block packet receptions. The 'block reception in progress' descriptors, or their logical equivalent, should be implemented completely in hardware, with DMA facilities to deal with mid-block packets when received. Under program control the head and tail packets could then be handled in parallel. Similarly a microprocessor should transmit the header packet of a long block and then set up the support hardware to transmit the mid-block packets, finally transmitting the tail itself. Whilst long blocks were being handled, the microprocessor should be able to send and receive short packets in parallel.

The Cambridge Fast Ring IC contains a receiver FIFO buffer large enough to hold only one packet. Reception of header, tail and complete-block packets would take significantly longer to process than hardware-manipulated mid-block packets in the scheme above. If when the interface is in the middle of receiving a long block a single packet block arrives, this latter block needs to be pulled out of the ring buffer as soon as possible, to prevent other packets from being rejected on account of the buffer being full. The support hardware should therefore DMA all SOB/EOB packets into a well known location and *then* notify the microprocessor to begin processing them, leaving the node and support hardware available to handle further packets.

7.7.1 Overall Hardware Structure

We designed a high speed interface for incorporation into the ISLAND servers built for Integrated Services applications. The lack of a stable Fast Ring has to date prevented assembly and testing of this interface. Low level details of the interface will not therefore be presented here. However an outline of its behaviour will be presented: the design has advanced to a stage sufficient to make this appropriate.[8]

The interface is designed for use with servers of both single and dual processor types, each suiting a different range of applications. According to the philosophy of chapter 6, the hardware is built as a peripheral local to the interface processor. The interface hardware is based around a commercial DMAC, chosen for its several and flexible operating modes, plus two microsequencers (of 64 and 16 states respectively) to handle the identification and separation of mid-block packets for reception and the automatic transfer of packets for both transmission and reception (using the DMAC). The interface hardware abuts to the interface processor in such a way that the DMAC can access the processor's on-board bus and hence from it the main server bus, using a minimum of hardware. A block diagram showing both the interface hardware and the way that it fits into an ISLAND server is given as figure 7.1.

The DMAC has four channels: channels 1 and 2 are used for mid-block packet reception, 3 for reception of other packets and 4 for mid-block packet transmission. Each channel is set up in peripheral-to-memory transfer mode (or vice versa for channel 4). Each channel operates by

[8] The design proceeded as far as circuit diagrams and microcode programs ready for construction of a prototype.

handling a series of 'transactions', each transaction being the complete transfer of a single packet on channel 3 or the transfer of all the mid-block packets of a single block on the other channels. For each transaction the relevant channel is initialised, after registers in the DMAC have been loaded with the start address of a memory buffer and a total number of words to be transferred. The main microcontroller strobes the DMAC each time a word is to be moved, until the programmed number of words has been transferred, when the DMAC disables the channel in question and interrupts the interface processor with an indication of the event. The microsequencer will detect the event (which will occur only on the final transfer within a packet) and disable itself from further use of that channel.

7.7.2 Reception Procedure

The reception procedure is best explained by following the same path as was used above to outline the software implementation. The hardware is initially set up to handle a single packet—channel 3 is set to transfer the packet's source address (a dibyte) and all 32 of the packet's data bytes from the ring to some memory location. When an SOB packet arrives, the microsequencer first transfers the first two data bytes (which constitute the packet header as defined in section 7.6) and the source address dibyte into comparison registers. On finding only channel 3 to be set up, the sequencer then strobes that DMAC channel to transfer to memory first the two dibytes already loaded into registers and then the remaining data straight from the ring. On the last strobe, channel 3 will be cleared down and the interface processor interrupted.

If the packet is a complete block, the interface processor will assign a new buffer for channel 3 and set the channel up again. It will then handle the block. If the packet is the start of a long block then, assuming channel 1 or 2 to be free, the interface processor will set up one of these mid-block reception channels to handle the exact amount of data expected before the EOB packet. The source address will be written into the appropriate comparison register. (The mid-block packet header value will have been written into the 'expected header' comparison register of channels 1 and 2 when the interface was first set up.) The interface processor will also set up a descriptor for use when the EOB packet arrives and a timer for the whole block. Finally it will allocate a new packet buffer for channel 3, re-initialise that channel and then copy the data from the header packet to the head of the block's buffer. Note that the mid-block channel must be initialised before channel 3 is re-initialised, even though this can mean an incoming packet unconnected with this block being held up or rejected by the Fast Ring station. This is in order to prevent the first mid-block packet being handled (incorrectly) by channel 3.

When a mid-block packet arrives its source address and header dibytes will be transferred to comparison registers and will be compared against the header value and source address expected by each of channels 1 and 2. If there is a match, the sequencer will strobe the relevant DMAC channel to transfer all the data straight from ring to memory. (If there is no match the packet will be handled by channel 3 and the interface processor: this allows more than two long block receptions at once, albeit at reduced throughput.) If the packet is the last mid-block packet of a block the DMAC counter will expire at the end of it, clearing down the channel and interrupting the interface processor. The interface processor will mark the block descriptor 'EOB expected' (just as it also does upon receiving the first packet of a two packet block).

When an EOB packet arrives it will be handled by channel 3, being transferred into a temporary buffer before the interface processor is interrupted, just as in the case of an SOB packet. After being interrupted, setting up another temporary buffer and re-initialising channel 3, the interface processor will check that the whole block has correctly been received and if so transfer the valid data from the EOB packet to the tail of the block buffer, finally notifying the client which requested the reception of its success or failure.

7.7.3 Transmission Procedure

In one sense, block transmission is simpler to explain than reception, because there are no unpredictably ordered events from the outside world to cope with. To send a long block, the interface processor first moves the header packet to the ring itself.[9] It then sets DMAC channel 4 into 'auto transmission mode', to transmit some number of complete mid-block packets to the single destination. In this mode each time the ring station signals that the previous packet has been transmitted the sequencer will load the destination address and mid-block packet header dibytes from registers to ring and then strobe the DMAC to transfer a mid-block packet's worth of data directly from memory to ring. When the last mid-block packet has been sent, the DMAC will disable channel 4 and the auto transmission mode, before interrupting the interface processor, which will transfer the final packet of the block to the ring itself and then notify the client requesting transmission of its completion.

This is straightforward compared with reception. However, as mentioned above, during transmission of a long block it may be necessary to interleave a short one. This short block will be transmitted by the microprocessor. Unlike the case of reception, where the interface processor never pulls data directly from the Ring, logic must therefore be provided to arbitrate rights of transmission between sequencer and interface processor (since interleaving within a packet is not useful!). In this design the interface processor is required to poll for transmission rights before filling the transmission FIFO. When the processor is not polling and the interface is in auto transmission mode then the FIFO will be filled continually by channel 4 as above. By polling for transmission rights, the interface processor sets a flag to indicate that the next empty FIFO occurrence should be reserved for it: when the FIFO next becomes empty polling returns 'transmission rights granted' and channel 4's access is held off until the following 'FIFO available' signal.

7.8 Concentrated Voice Streams

The capabilities of the hardware outlined above are suited to a traffic pattern where a small number of long blocks must be interleaved with a possibly larger number of small blocks: overall throughput will be good since most of the data will be transferred in long blocks. An incoming short block will be removed from the reception FIFO very quickly, so that a high speed stream of mid-block packets will not be badly affected by the arrival of a single SOB/EOB packet.

The interface will not perform so well when confronted with a large number of packets, each of which requires individual handling. (This is not surprising, since this was the reason for

[9] Transmission is accomplished simply by loading a station address dibyte and then 32 bytes of data into the ring.

defining a long block.) Such a load will be offered in two cases in an ISLAND Fast Ring environment—the first is that of a station, such as a conference server, handling the transmission and reception of large numbers of voice streams in parallel and the second a ring-to-ring bridge. The first is clearly relevant to the main theme of this dissertation. The second is less so, but it influenced the design of the interface and so will be considered briefly.

A key reason for selection of the particular DMAC used in this design was that it has a 'chaining' mode. In the procedures described above, the DMAC performed peripheral-to-memory transfers between the Ring and a contiguous buffer of specified size. When the correct amount of data had been transferred the channel would be disabled and an interface processor interrupt would occur. In chaining mode, a linked list of buffers is specified to the DMAC. Each buffer in the list consists of a link pointer, the length of the data space in the buffer and then data space following immediately after. As it is strobed by the microsequencer, the DMAC follows this linked list, performing transfers until it exhausts the buffer which has a zero link pointer, whereupon it disables the channel and causes an interrupt.

It was suggested above that there are some circumstances in which it is reasonable for the interface to place the data it receives in one of its own buffers rather than one nominated by the client, and then to notify the client of the address of that buffer. This would be acceptable to the conference server whose interface sees a client requesting streams of single voice packets from several transmitters simultaneously: the client will be satisfied if handed pointers to the packets as they arrive. The DMAC needs a long linked list of buffers from the heap, with enough data space in each to hold the source address plus all of the data from a single packet. Filled buffers are sorted into lists for individual streams and handed to the client, which performs compounding operations on a set of buffers and then passes them back to the interface for transmission, still in the format (destination) address plus data. Such a technique, based upon linking and unlinking of lists, is much more efficient than one calling for copying of the data.[10]

Transmission of a linked list of individual blocks requires the interface design to support an additional, slightly different auto transmission mode: rather than transmitting destination address and packet header from registers followed by 15 dibytes of mid-block data directly from memory, the interface design is also able to move 17 dibytes to the Ring straight from memory. (It also has to be able to follow a linked list whilst transmitting of course: this is only a matter of setting the DMAC into an appropriate mode.)

7.9 Other Applications

As already noted, many programming environments require that long blocks be delivered in single contiguous buffers nominated by the client: the delivery of a linked list of bytes would not enamour most high level programmers. However this design of interface enables linked lists of packets to be converted into contiguous format where appropriate. In such a case channel 3 is

[10] Note that this is a somewhat different environment for the conference server's compounding routines to work in from the voice provider base that was described in chapters 5 and 6. An analogous service to the voice provider would be provided here to re-link buffers from the single reception list onto queues each representing a single voice stream. A significant part of the compounding routines as described in section 6.5—the routine selection tree and table lookup—would be unaltered. The changes would reflect the need to follow pointers for each stream along a linked list instead of around a circular buffer.

used to transfer all incoming packets to one linked list. The processor sorts that list into individual lists of complete blocks and then channel 1 is used in the DMAC's memory-to-memory transfer mode to copy data from list format into contiguous buffers. (Channel 2 would perform the inverse function for transmission.) This scheme would have the advantages that

i) packets would be pulled from the reception FIFO at an approximately constant rate. In the original scheme there is an undesirable pause when particular packets are being processed, such as the delay between DMAing the first and second packets of a long block. However a similar pause in the transmission can be arranged at that point, to avoid this problem at least in part.

ii) for large numbers of simultaneous long blocks, this may give greater throughput.

Large numbers of simultaneous long block receptions are likely to be rare. This scheme has the drawback that an extra DMA copy is required for all packets. We have alluded already to applications where the interface would be copying images directly from the Ring to a memory-mapped frame buffer: in such cases the extra DMA copy would have a serious effect on performance. A worse overhead still would arise due to the interface processor processing every mid-block packet (even though only to sort it onto an appropriate queue), something which our interface design specifically set out to avoid.

Chaining mode is thus most useful for applications where the client will actually be happy to receive a chain of buffers. Another application where this is true is in a Ring-to-Ring bridge. It was remarked in chapter 3 that the Cambridge Fast Ring node IC can function either as a station or as a bridge node. The Fast Ring addressing scheme supports flat addressing across a mesh of Rings. A bridge is made from a pair of nodes, each with a table of destination addresses of packets which should cross the bridge rather than continuing on the same ring. When a packet appears at the reception FIFO of a bridge node, the data is preceded out of the FIFO by the source and destination addresses of the packet. Both of these addresses will be required on the destination ring. The transmitter FIFO of a node run in bridge mode expects the same format, so that a bridge can be built almost entirely from two node ICs back to back.

Such a simple bridge will not perform very well under many load conditions. Details of the problems are not relevant here, but the bridge will become congested for all traffic if a fast transmitter attempts to transmit through it at high speed to a slower receiver. Provision of additional FIFOs between the two Ring ICs will only delay this effect. If voice traffic is passing through the bridge it will delay that traffic also, which is unacceptable. One way to avoid the problem is for transmitters to take heed of the capacities of their corresponding receivers. This may lead to establishment of transmission rate negotiation as a part of a long block protocol. Such an approach closely resembles the NETBLT protocol [Clark 85b] and has been proposed for the Fast Ring.

Like the designers of NETBLT, we believe that such an advisory protocol should be accompanied by a hardware protection mechanism, to prevent a malicious (or malfunctioning) transmitter from disrupting the whole network. The chained reception scheme of our interface can readily be applied for this. Incoming packets would be placed in a single list and then re-linked into for example queues for particular destinations. The opposite half of the bridge might then transmit a packet at a time, monitoring the time taken to transmit to each destination and using the information to determine the relative frequencies of transmission to be attempted for different addresses, possibly also deleting particular packets from overlong queues.

The non-chained reception and transmission modes of the interface can also be used in a bridge. In parallel with chained mode operation on channel 3 the interface can test for packets with source/destination address combinations which are being handled specially by channels 1 and 2 and being placed in long contiguous buffers. These channels would only be used for specific destinations known to be able to receive from the bridge at high speed. Channel 4 would be used in two modes here, just as for a station, to transmit either a long block of packets to one destination or single packets to various destinations.

To use a long buffer the interface would monitor for SOB packets of long blocks: upon finding one it would then set up a buffer to hold all of the rest of the block. It does not matter if a packet with the same source/destination address pair but not part of the long block becomes placed in that buffer.[11] The other half of the bridge would then temporarily change its transmission mode from transmitting sequences of single packets to transmitting this stream of packets to the single destination.

This scheme is potentially attractive for high bandwidth connections through a bridge, as might be required for video traffic. It would need to be used with considerable care in an Integrated Services environment, however, to ensure that voice did not become unacceptably delayed. Voice packets caught up in a long buffer might be held up considerably. Furthermore the bridge should not transmit to a single destination for a long time either, in case voice streams for other destinations became delayed too much. Nevertheless short bursts of transmission of sections of a long block might produce a worthwhile performance advantage. For Integrated Services traffic, the chained reception approach on its own seems to offer most benefits without the complex precautions needed for operation in long block mode.

7.10 Protocol Restrictions Imposed By The Hardware

The different modes of operation of the interface have been outlined in some detail because a key part of the design aim was flexibility and general applicability to an Integrated Services environment. It was desired that the hardware should be useful in long block station mode (for video or data transfers), chaining station mode (e.g. for voice servers) and in a bridge. Another major part of the design aim was to place minimum constraint on choice of protocols which could be used with the hardware.

Complete flexibility of protocol is possible using the scheme of chained transmission/reception on channels 3 and 4 with channels 1 and 2 used for memory-to-memory (blocking/unblocking) transfers as necessary. The interface processor's software then defines what blocking strategy, if any, is used. This does not offer very good performance, not surprisingly since it does not exploit the special features of the interface. For high interface performance a protocol definition needs to be exploited to some extent at the hardware level.

[11] Care must of course be taken to schedule transmission such that all packets in transit between a given pair of source and destination addresses leave the bridge in the sequence in which they arrived.

The long block scheme was designed to be exploited at the hardware level, without constraining protocol choice too tightly. In the approach described in section 7.7, incoming packets are sorted according to source address and packet header. The header value is programmed into comparison registers at interface set-up time rather than being hard-wired into the interface to allow flexibility. A long block could be defined with a receiver sub-address in place of the packet header—such a scheme would be compatible with the hardware.[12]

Any scheme for matching packets against requests can be accommodated by the hardware in which all packets to be handled by a single DMA channel can be distinguished by a unique combination of source address and the first dibyte of the packet. Greater variations than this, such as matching the first two dibytes of the data field, could be accomplished only by microcode changes. The freedom allowed by the hardware design as it is, however, gives much scope for different protocols to be tried.

The limit of two mid-block handler channels was set in order that the interface could be realised using a modest number of standard components (DMA controller, PLAs and PROMs) as building blocks. These components were chosen partly because they lend themselves to implementation in VLSI. If this 'semi-intelligent' interface were to be implemented in VLSI, following the manner of [Mockapetris 77], it would be feasible to use more mid-block channels if desired. This would not, however, alter the behaviour of the interface from that described here.

[12] as are all of the schemes in section 7.5: this is scheme 3

8. An Integrated Editor

The last four chapters have set out a design for Integrated Services, starting at the level of voice functions which comprise an integrated PABX and then progressively working down as far as the level of packet transport across a network. As well as forming the basis of a PABX, the voice services outlined in chapter 4 are intended to serve as the basis for higher level Integrated Services—services offered to the user at a workstation and based on the capability to manipulate several media together.

The next two chapters of this dissertation concern these higher level Integrated Services. This chapter describes the incorporation of voice editing capabilities into a general purpose document editor. It is presented for two reasons. First it stands on its own as a study of the style of interface that we believe an integrated system should be able to offer to the user. Secondly it raises issues of how multi-media documents should be supported by a system and holds up to scrutiny the suitability of the ISLAND design for this purpose. The latter issues are discussed in the next chapter, followed by a more general discussion on the multi-media workstation and its capabilities.

The editor described in this chapter was implemented in the Etherphone environment. Its design attempted to find suitable solutions to two sets of conflicts. The first is the desire for fast and convenient use of a voice editor, against the wish for a powerful, flexible and extensible interface. The second is between the benefits from exploiting uniformity in handling different media and a recognition that there are some operations on voice which are best presented as being different from operations on other media.

The environment in which the editor was implemented is discussed. The voice handling capabilities are then described in two parts—facilities for simple verbal annotation of documents and a full voice editor. Finally facilities are described which provide the capabilities of an idealised dictation machine.

8.1 Background

The current capabilities of commercial workstations were outlined in chapter 1. It was observed there that the workstation increasingly plays a part in most of the activities in a working day. The combination of the facilities available on current workstations with those provided by the facsimile machine and the phone satisfies all of a user's communications requirements. Given an integrated architecture, which offers capabilities to control a variety of media in one coherent way, one may therefore usefully ask the question as to what new, useful services could be offered if these three objects were to be combined. Facsimile features could become part of the workstation's capabilities and the existing functions of the office phone could be controlled from the workstation. These two topics will be discussed in the next chapter. The coupling of phone and workstation might also be used to incorporate voice into documents. Work has been reported on design of multi-media documents in a more general sense than this. Particularly notable attempts at unified document architectures to date have been the Open Document Architecture (ODA) [Horak 85], and Diamond [Forsdick 84, Thomas 85]. The former includes plans to incorporate voice into its generalised structure [Campbell-Grant 86]. The latter already handles voice as a part of documents: experience with that system will be discussed below.

This chapter will be concerned not so much with the architecture of documents that contain voice as the interface by which that voice can be manipulated. This contrasts with the ODA, which is concerned with standards for the exchange of documents. The emphasis chosen here stems from a belief that the voice manipulation interface will have a crucial bearing on how voice is incorporated into a document architecture, since the interface will dictate to a large extent how voice is used in that architecture. We will attempt to justify this view both in the description of the editing interface and later by discussing the demands that the interface places on a multi-media system that supports it.

8.2 Conflicting Requirements upon a Voice Interface

A number of systems which combine voice and visual media have been built. In trials using two of these systems [Nicholson 85, Poggio 85], it was found that people will not make use of voice annotation unless it is a simple and fast process, encouraging spontaneous and off-the-cuff comments. Many of the existing voice/visual systems are products that, recognising this need, include a set of fast and convenient facilities for voice annotation.

As an illustrative example one such product, the Centerpoint System [Centerpoint 86], supports the manager-and-secretary paradigm. The secretary is provided with a keyboard and a touch-sensitive screen. The manager, who is presumed to have neither the time nor the skill to use a keyboard, has only the latter and dictates letters, reports and memoranda for the secretary to type. Subsequently, the manager can mark typed copy with further vocal annotations that the secretary then uses to correct the documents.

There are many other scenarios in which uses for a simple voice annotation system can be envisaged. Members of an engineering design group could send drafts of reports or documentation to other group members, who would then annotate them with voice messages and return them. University tutors might use annotation for marking essays. The tutor would receive completed work by electronic mail, add comments in spoken form and then return the whole thing to the student.

Vital though it is for spontaneous use, on its own a simple annotation interface cannot provide a number of important capabilities, including the ability to make arbitrary modifications to voice annotations and to provide voice facilities in different document types as part of a general architecture that can be offered as a basis for future applications.[1]

Recently, some general-purpose systems incorporating voice have been built. Produced by computer science research teams, these are intended to be flexible, extensible, and to provide unified user interfaces linking a variety of different media. The most notable effort to date is a collaboration of several research organisations, built around a common multi-media document

[1] In fairness it should be noted that for this reason most of the existing systems that support annotation, although designed for specific purposes and strictly limited in the capabilities they provide, do include some voice editing facilities.

architecture [Garcia-Luna 84, Reynolds 85, Thomas 85]. In such systems, graphical objects are typically placed in text by opening a graphics 'sub-window' within the text. Further text may be placed inside such a graphics sub-window by placing a text sub-window within it, and so on—the structure is general and flexible. These systems allow voice to be inserted in an equally general manner. In one such system [Thomas 85], the user opens a voice sub-window (consisting of a voice 'icon' and a textual identifier for the speech) and then talks into a microphone. If (s)he wishes to edit the resulting annotation, an energy profile can be made to appear in a separate window, showing a linear plot of the speech energy against time. Editing is accomplished by specifying editing operations upon the profile that are then interpreted as edits to the actual voice.[2] These editing capabilities tend to support exacting, detailed modifications to the voice, rather than the simple operations that are needed for annotation. The interfaces as a whole tend to remove the possibility of using the editor for off-the-cuff remarks, by presenting facilities generalised across all media but too cumbersome to be used easily for voice.

The point of citing these two types of example, typical in general of systems currently available, is that there is a conflict between user interfaces that are convenient for simple dictation and document annotation, and those that allow more general-purpose manipulation of voice and other media.[3] A satisfactory trade-off between the desire for generality and the need for a user interface with the right degree of spontaneity is hard to achieve. The following description of a voice editor aims to show that an acceptably wide range of voice editing applications can be presented through a user interface that permits both simple annotation and more elaborate manipulation of recorded voice. Whilst one goal was to retain flexibility and generality, careful thought has been given to the user's model of the effects of all commands provided in the interface. If an operation could not be expressed in terms of a reasonably simple user's perceptual model, or if the result of some manipulation might not be clear to the user under all circumstances, then it was not included in the interface, regardless of its utility in some contexts.

8.3 Incorporating Voice into an Editing Interface

This section discusses how the conflicting requirements for ease of use and for powerful and flexible voice editing capabilities should be balanced. First the power and the limitations of a uniform approach to editing all visual (or visually represented) objects are discussed. Then the approaches to voice annotation and to voice editing used in this voice editor are explained.

8.3.1 Use of a Uniform Editing Interface

An important characteristic of a good integrated editor, and in general of a good integrated working environment, is uniformity. Uniformity means that the user learns a consistent set of methods for selecting a variety of visual objects and a consistent set of commands, each of which will have a similar effect on the selected objects. This common interface hides from the user any implementation differences due to the different natures of the various visual media.

[2] This may seem a rather strange distinction to make. As will be discussed later, however, there is a difference between a window full of text, where the actual text being manipulated is displayed in the window, and a voice energy profile, which is only a token representation of the voice itself.

[3] This conflict is not limited to the handling of voice: the same type of conflict between providing simple methods for common operations and allowing maximum flexibility and power for more elaborate usage arises in text editors, illustrators, and many other interactive applications.

An example which illustrates the concept well is the editor supplied with the Xerox 8010 (Star) workstations [Smith 82]. Star allows the same sequence of keystrokes and mouse button clicks to be used to move a portion of text from one textual window to another, to relocate a geometric figure or a digitised photograph within a graphics window or to move an entire document from one 'filing cabinet' to another. Deletion of a selected document can be accomplished by moving it to an iconic representation of a waste paper basket.

The basic approach of the voice editing interface to be described here, in common with most other voice editing systems, is to extend use of the concepts of the uniform editing interface beyond visual media, to encompass voice as well. However we believe that it is as important to identify those editing functions and patterns of operation that cannot usefully be applied uniformly to voice, and to add some specialised operations dealing with characteristics of voice that are unparalleled in other media. We believe that application of these guidelines has produced an interface with some important advantages over earlier integrated voice editors.

Extension of the ideas of uniformity to voice handling does not need special justification. The next two sections give examples of cases where over-zealous application of uniformity is not beneficial.

8.3.2 An Approach to Voice Annotation

A common approach used in integrated editors for the insertion of a graphical illustration into a textual document was sketched above. The user creates a graphics sub-window within the text and then creates and modifies the illustration directly within that sub-window. Textual annotation of the illustration may be added by creating a further text sub-window within the graphics sub-window; and so on. This model is perfectly natural and understandable to the user and is reasonably straightforward to implement.

We assert that it is a mistake to extend this model to voice. Creating a window is a heavyweight operation which goes against the 'spontaneity of use' requirement for vocal annotations. Also no graphical representation can convey the actual content of a voice annotation: only playback of the voice itself will accomplish that. We prefer instead, therefore, to indicate the presence of voice by decorating the region of text to which the annotation pertains with a simple pictorial marker. It is important that this should not alter the layout or structure of the visual medium to which it has been appended. If the editor is being used for typesetting, a reviewer might wish to comment vocally that "this paragraph looks too long and thin". Such a comment refers to the form rather than the content of its target. The annotation becomes meaningless if its inclusion in the text causes a voice sub-window to appear or if the editor otherwise distorts the layout of the text to which the annotation is attached.

8.3.3 An Approach to Voice Editing

Straightforward annotations to text can be made simply by recording single utterances from a microphone. Their presence can then be marked pictorially. Playback of complete annotations is equally straightforward. At some stage, however, the user will wish to extend or modify a voice annotation. It may also happen that the amount of voice to be included in an annotation exceeds that which can easily be dictated in one go. At this point there will be a need for voice editing capabilities.

Nearly all systems that support voice annotation also include some form of voice editor. Most of these voice editors function by presenting the user with a visual representation of the voice and with operations on this representation that allow manipulation of the voice. Using such a static, visual representation of what in reality is an audible and dynamic medium, the user can access any part of an utterance; select certain parts of it; reorder selected segments etc. much more conveniently than would ever be possible using conventional telephone-based voice editing facilities [Gould 82] or a simple dictation machine. Our voice editor design adopts this approach.

A visual representation of voice implies little about the actual editing interface that should be used to support voice editing: on this issue existing systems diverge. A summary of the experience with this voice editor is that useful voice editing primitives have come more from observation of the differences between voice and visual media than by heavy use of the principle of uniformity. When editing text, one often deletes and inserts material character by character, for example to correct spelling mistakes. If the uniform interface model is stretched too far, it will be concluded by analogy that voice ought therefore to be edited at the word level or at the phoneme level. This is a mistake, because the ways in which voice and text can most easily be generated are very different. When looking at written text, one can easily correct it character by character, whereas the idea of replacing a complete sentence on account of one incorrect character is both absurd and tedious. On the other hand, most people are adept at speaking entire words or sentences, but find it difficult to edit a single word in a spoken sentence without losing the natural flow. At an even lower level, replacing one vowel sound with another is practically guaranteed not to produce satisfactory results [Allen 83]. User interfaces that operate at this fine level of detail are slow and tedious to use. In one case where an over-detailed editing interface was provided [Poggio 85], the implementers observed that it was hardly used. A voice editor could much more usefully provide operations that support the modification of voice documents at the phrase (or sentence) level, using naturally-occurring pauses in the speech to denote phrase boundaries.

8.3.4 Visual Cues for Marking Voice

There is one more important philosophical point that underlies the design of this editor. Even the best possible graphical representation of a recorded utterance will fail to convey most of its content. Therefore it is easy to lose one's place when dealing with such representations. A good editor should incorporate methods for marking significant points on the visual display. Simple temporary markers would be useful for keeping track of important boundaries during editing operations. Permanent markers are needed to mark significant locations within extended transcriptions in a way that will be meaningful to the user on re-examination of the voice after a long interval. It will be explained below how this interface tries to provide markers which reflect the meaning (or linguistic content) of a segment of speech rather than portraying mathematical artifacts such as energy profiles.

One of the forms of marking developed as a part of this editor is believed to be unique to this work. It seems useful to draw a distinction between 'stale' and 'fresh' voice. 'Stale' voice is voice that is already in a document that one has received and is editing. When one is speaking into a microphone, that voice is 'fresh'. The difference lies in the speaker's awareness of what has just been said; it is easy to produce a new version if a sentence is fresh in one's mind. The user is more likely to want to modify the fresh voice than the stale—to correct the choice of words or phrasing; eliminate coughs, hesitations or other blemishes; change the tone of voice used in a selected phrase; and so on. This is how an idealised dictation machine would be used. Our editing interface incorporates automatic marking of the freshness of voice and supports a set of fast and convenient operations to support the functions desired of a dictation machine, i.e. reviewing, replacing, and extending the fresh phrases in a voice recording.

8.3.5 Characteristics of Existing Systems

There is already a wide range of special-purpose and general-purpose systems for editing voice and annotating documents. Among these are several product systems—the Centerpoint system mentioned above, the Wang Audio Workstation [Wang 82] and the Sydis Voicestation [Nicholson 83]. The last is also now known as the British Telecom *Mantis* system. Other systems were produced as research projects—two related voice editors produced at AT&T Bell Laboratories [Maxemchuk 80a, Wilder 82], the Intelligent Ear [Schmandt 81] and Phone Slave [Schmandt 84] projects from the Architectural Machine Group at MIT, a Master's Degree project by Mirrer [Mirrer 82], BBN's Diamond system [Thomas 85], the CCWS system (which is a multi-media information system and incorporates a multi-media editor) by SRI [Poggio 85] and a voice and telephony project at IBM San Jose [Ruiz 85].

All of these systems except the Phone Slave and the IBM system provide voice editing facilities. The Centerpoint system, Diamond, CCWS, the Intelligent Ear and Mirrer's voice editor present a graphical energy profile for editing, whilst the others use simpler representations similar to the one chosen in this work, as will be explained below. The AT&T systems, the Intelligent Ear and Mirrer's system restrict editing to phrase boundaries, or at least to significant silence periods. The Wang, AT&T and Sydis systems permit forms of textual annotation of voice to serve as visual markers. (In the case of the AT&T systems, text and voice can simply be mixed in a linear progression.) The Intelligent Ear makes novel use of speech recognition for producing markers, as will be noted below.

The Centerpoint, Sydis, BBN and IBM systems permit vocal annotation of text documents: of these, the Sydis and IBM systems use a form of annotation that does not affect the formatting of the documents, except for markers in the margins. The Phone Slave is a special-purpose system exploring the integration of interactive answering machines and electronic mail in a number of novel ways. As such, its characteristics do not fall neatly into either the annotation or the editing categories. The message browsing facilities of this system do however include a visual representation indicating the duration of each message segment.

The BBN 'Diamond' system is based on a general multi-media document format. That format is very restricted in its structure, document representation being limited to 'shallow trees'. For example an image window can be placed within text, but any text then placed within the image will be scan-converted and thereafter not manipulable as text. Diamond is also capable of communicating with other systems also using the format mentioned in Section 8.2 [Reynolds 85].

CCWS is heavily oriented towards automatic presentation of information in response to queries. Finally, it has already been mentioned that the ISO standards body is defining a multi-media format, the ODA [Horak 85], which it is intended will eventually include voice.

8.4 The Cedar Editing Environment

The voice editing interface which will be described below was built using the Cedar programming environment [Swinehart 86]. Cedar was developed as a research prototype by the Computer Science Laboratory at Xerox PARC. The detailed design choices for the voice interface can best be understood given a little background knowledge of the editing facilities in Cedar.

Tioga is the standard text-editing program in Cedar. Tioga is in essence a high-quality galley editor; it supports the creation of text documents and allows control and parameterisation of type faces, type styles, paragraph and more general layout formats etc. Tioga is unusual amongst text editors in that its documents are tree-structured rather than consisting of plain running text. This structure makes possible such operations as displaying only the section headings or key paragraphs of a document, which means that searches through a Tioga document for a particular section can be performed quickly and effortlessly. Finally, Tioga includes capabilities to incorporate illustrations and scanned images in its documents. Tioga can create both monochrome and full-colour documents.

Given such a general structure, it might be expected that manipulation of a Tioga document would be cumbersome. On the contrary, the editor interface has been constructed very carefully, using menus of commands; keystrokes and a three button mouse, to afford very fast and fluent editing. In particular

> care has been taken to determine which editing operations are required most frequently and to make those operations very fast.
>
> the use of pairs of selections, specified using the mouse, is the basis of many of the most frequent editing operations: operations such as copying and amending text are thus made very fast.
>
> the command interface is enriched by *accelerators*, which are (usually) invoked by simple key combinations (such as control-shift-S) and perform common sequences of operations. These accelerators can be bound to single keystrokes or other input actions by declaration in binding configuration files: the individual user can create his own bindings and various well-loved sets of bindings are available generally on the system.

In summary, Tioga has a general and versatile structure combined with a user interface that was designed for fast and fluent operation.[4] We have tried to follow that pattern in the voice editor. Whilst sections above have dwelt on instances where relentless application of the uniformity principle ought to be avoided in a voice editing interface, many of the Cedar editing primitives were appropriate to such an interface and have been used. This will be discussed

[4] Compare with the very similar philosophy for the design of the ZOG user interface [McCracken 84].

further in following sections.

Cedar was designed such that many applications employ the capabilities of the Tioga editor. These applications include the electronic mail system; the system command interpreter; manipulation of permanent text files as well as temporary text windows such as compiler error logs; plus any other tools that require the entry and manipulation of text by the user. Such extensive use of a single editor interface lends considerable unity to the Cedar user interface, since in all of the different types of application in which Tioga is used identical keystrokes will perform identical functions. Wherever Tioga is used, all of its formatting and multi-media facilities are available. Thus by adding voice annotation to Tioga we have made it available in a wide variety of Cedar applications.

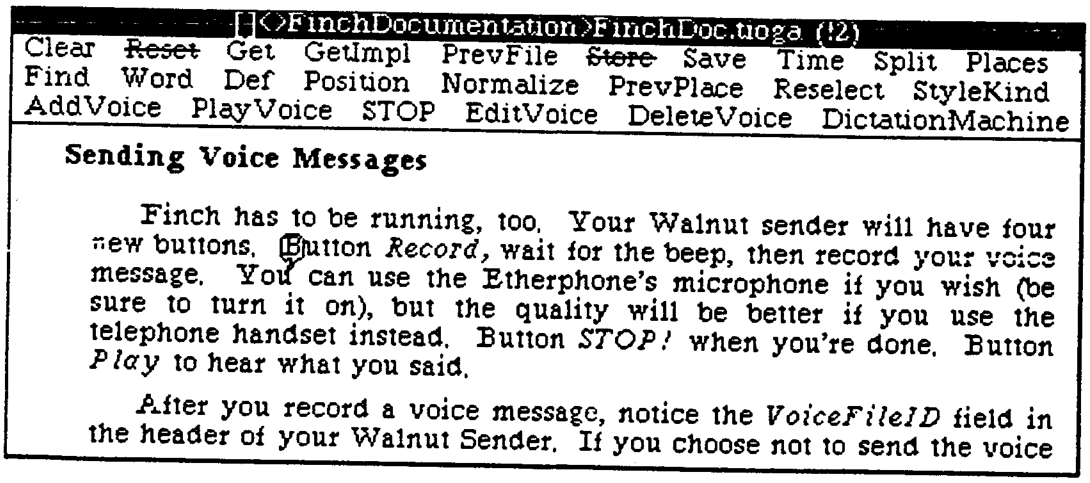
8.5 The Voice Editing Interface

This section presents the capabilities of the Cedar voice editor prototype in substantial detail. First simple methods are described for adding voice annotations to standard Tioga text, and for listening to such annotations in their entirety. This is followed by a description of more general-purpose techniques for editing voice annotations, all based upon manipulation of visual representations of the recorded voice.

8.5.1 Basic Annotation

Figure 8.1 shows a text document window, or *viewer*, from a Cedar workstation screen. The top region of the viewer is a system region that includes an identifying label and three rows of *menu buttons*. A portion of the document to be edited occupies the remainder of the viewer. The user interacts with the document by using the mouse to select objects of interest and then positioning the mouse-driven cursor over one of the menu buttons and clicking a mouse button. (The latter two actions are referred to simply as *clicking* a menu button). Tioga also allows the more common editing operations to be performed on selected objects by using keystrokes from the standard keyboard.

Figure 8.1: a typical Cedar viewer, containing a formatted document with a voice annotation in it.


Clear ~~Reset~~ Get GetImpl PrevFile ~~Store~~ Save Time Split Places
Find Word Def Position Normalize PrevPlace Reselect StyleKind
AddVoice PlayVoice STOP EditVoice DeleteVoice DictationMachine

[]<>FinchDocumentation>FinchDoc.tioga (!2)

Sending Voice Messages

Finch has to be running, too. Your Walnut sender will have four new buttons. Button *Record*, wait for the beep, then record your voice message. You can use the Etherphone's microphone if you wish (be sure to turn it on), but the quality will be better if you use the telephone handset instead. Button *STOP!* when you're done. Button *Play* to hear what you said.

After you record a voice message, notice the *VoiceFileID* field in the header of your Walnut Sender. If you choose not to send the voice

Any single text character within a Tioga document can be annotated with an audio recording of arbitrary length. To add an annotation, the user simply selects the desired character within a text viewer and clicks AddVoice in that viewer's menu. Recording begins immediately, using either a hands-free microphone or the user's telephone handset, and continues until the user clicks STOP. As recording begins, a distinctive iconic indication of the presence of the voice annotation is added to the document, as a decoration of the selected character. In the prototype system, this *voice icon* is an outline in the shape of a comic strip 'talk bubble' surrounding the entire character, as appears in the second line of text in the first paragraph shown in Figure 8.1.

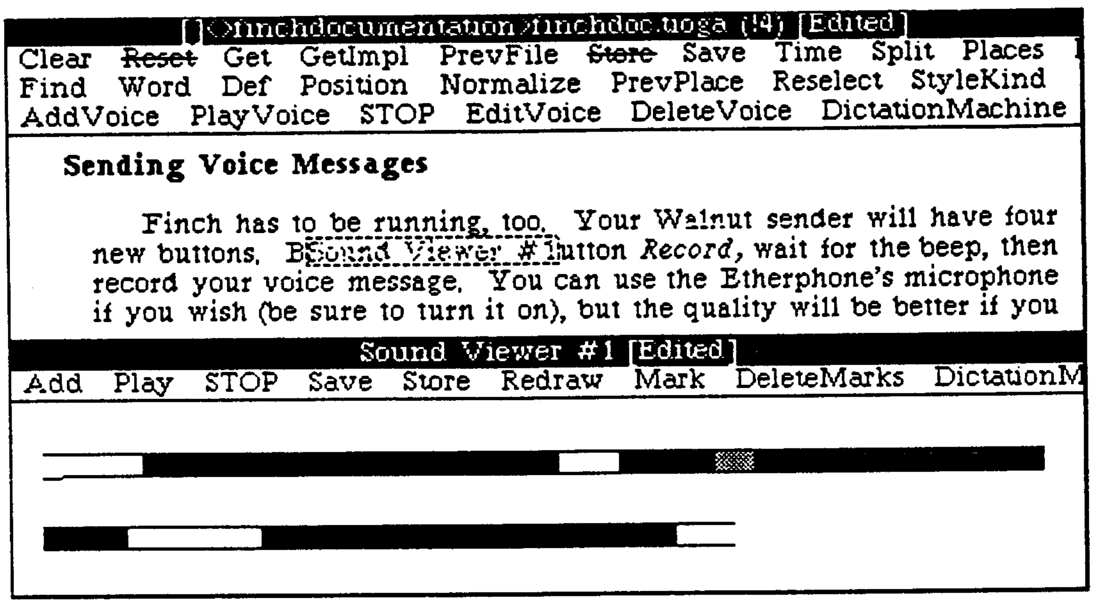
Adding a voice icon does not alter the layout of a document in any way. Thus voice annotations can be used to comment on the content, format or appearance of formatted text. Also programs such as compilers can read the text, ignoring voice icons just as they ignore the font information in Tioga documents. Voice annotations may be used, for example, to explain portions of the text of a program without affecting the ability to compile it. Like font information, voice icons are copied along with the text they annotate when standard Tioga editing operations are used to move or copy that text to other locations, either within the same document or from one document to another.

A voice annotation becomes a permanent part of a Tioga document. Copies of the document may be stored on shared file servers, or sent directly to other users as electronic mail. To listen to voice, a user selects a region containing one or more voice icons and clicks PlayVoice. Since playback takes some time, the user may queue up additional PlayVoice requests during playback. These will then be played in sequence. The STOP button can be used to halt the whole process at any time.

8.5.2 Voice Viewers

The procedure outlined above is highly satisfactory for short annotations, but for more complex annotations the user will need facilities to edit portions of voice. To keep things simple for the user performing rapid annotation, all that appears in the text is an icon representing each complete voice utterance. To perform editing operations the user selects a region of text and clicks EditVoice. One *voice viewer* opens up for each voice icon within the selection. Each of the selected voice icons now changes its appearance to that of an *opened voice* icon—it now displays a number that identifies the corresponding voice viewer. Each voice viewer is labelled with its number, so that the user can easily see the associations between voice viewers and icon positions in text viewers. Figure 8.2 shows what the voice icon in figure 8.1 looks like when opened and also shows the corresponding voice viewer.

Figure 8.2: the voice icon of figure 8.1 has been 'opened', producing the voice viewer just below it.


[]<>finchdocumentation>finchdoc.tioga (!4) [Edited]
Clear ~~Reset~~ Get GetImpl PrevFile ~~Store~~ Save Time Split Places
Find Word Def Position Normalize PrevPlace Reselect StyleKind
AddVoice PlayVoice STOP EditVoice DeleteVoice DictationMachine
Sound Viewer #1 [Edited]
Add Play STOP Save Store Redraw Mark DeleteMarks DictationM

Sending Voice Messages

Finch has to be running, too. Your Walnut sender will have four new buttons. BSound Viewer #1utton *Record*, wait for the beep, then record your voice message. You can use the Etherphone's microphone if you wish (be sure to turn it on), but the quality will be better if you

We have already expressed a desire to steer the user towards phrase-level editing and away from editing at a finer grain. For this reason, we have avoided the type of display that many other designers have used [Centerpoint 86, Mirrer 82, Schmandt 81, Thomas 85], which shows a graphical profile of the sound energy plotted against time. Such a display encourages the user to identify visually the individual consonants, phonemes or words within voice. Tioga voice viewers in contrast look essentially like 'capillary tubes'—long horizontal rectangular tubes filled in to indicate portions of sound and unshaded where there is enough silence to identify a phrase or sentence break. As with other voice editors, the length of each region of a capillary tube corresponds linearly to the duration of sound or silence that it represents. The appearance of voice viewers is very similar to the graphical depiction used in the Sydis Voicestation [Nicholson 83].

As can be seen from figure 8.2, open icons *do* disturb the format of the text in which they appear, to make room for the voice viewer number. The whole viewer in the figure represents about twenty seconds of voice. The small grey rectangle in the figure is the playback 'cue' (to be explained in section 8.5.4).

Although the capillary tube format heightens the tendency to think in terms of phrase-level editing, it is sufficiently uninformative that users will not remember which capillary segment represents which portion of speech. When editing text, one tends to focus in turn on several portions of text, then locate them again to make a specific edit. The user must be able to do the same with voice. This seems in part to be what has led other designers to the energy graph approach, even though this approach is only moderately successful as a means of providing context. In particular it entails the user associating energy shapes with particular phrases, associations which the user will only be able to retain for short periods, and represents an undesirably low level identification. This editing interface therefore supplies instead a number of marking techniques that it is believed can provide this context more successfully, without the unwanted side-effect of encouraging the user to focus on an undesirable level of detail when editing.

8.5.3 Voice Editing

How are voice editing operations accomplished? In Tioga, the user copies text from one position to another by making two selections with the mouse, then pressing a particular key on the keyboard. The uniform editing philosophy says that exactly the same actions should be used to copy voice from one position in a voice viewer to another. In fact, in this editing interface, all of the standard Tioga text-editing operations are identically used for editing voice also—operations such as deleting, moving, copying or transposing portions of voice within or between voice viewers.

Some operations have analogous rather than identical meanings within voice and text viewers. 'Selection granularities' are one example. We have already referred several times to 'making a selection'. Tioga operations are mostly executed by selecting some region of text using the mouse, and then indicating a function to be applied to it. For convenient and fast editing, Tioga offers easy selection of different units or granularities of text—single characters, complete words, whole paragraphs or entire documents. A selection, once made, can be extended to include contiguous units at the same granularity. Now in voice viewers we are trying to promote as units not characters and words, but phrases and sentences. Therefore we have set voice selection granularities to phrase-sized units and larger, so that the user is steered towards making edits at an appropriate level. A finer-level selection is available providing resolution to a quarter of a second, but its use is discouraged for ordinary editing. (It was remarked above that edits at the phoneme level, if attempted, invariably sound bad.)

Inserting new voice into a voice viewer is analogous to generating new text using a keyboard in conventional documents. The user selects a position within the voice representation, clicks AddVoice and then speaks into a microphone. Whilst the new information is being recorded, a series of arrowhead symbols is gradually inserted into the capillary tube, beginning at the insertion point. New arrowhead symbols are added at the proper rate to indicate according to the time scale of the capillary tube display how long the speaker has been talking (as shown in figure 8.3 below). Once STOP is invoked, the arrowheads are replaced by a normal capillary display (but possibly enhanced visually using the techniques to be described in section 8.5.4).

Finally, the user can save a series of voice edits by clicking Save to preserve them as the new 'contents' of the corresponding voice icon (if there is one), or by using Store to add the voice as an annotation at any selected position in a text viewer. The use of these buttons is analogous to their interpretation in text viewers to store edited text in permanent files. When the Store button is used, an open voice icon is created at the new position to represent the changed information in the voice viewer. The voice annotation at the original text position (if any) retains its original value: its open icon reverts to the standard 'bubble' form.

8.5.4 Marking Voice

How can the user know which portions of the voice to edit, given only black regions in voice viewers representing 'sound' and white ones representing 'silence'? The next few sections describe a number of methods which were developed to help the user establish the needed context.

8.5.4.1 The Playback Cue

No pictorial representation of a voice segment can truly express its content: the only way completely to determine what sound is represented by a capillary tube is to listen to it. It was described above how the user can select an area of text in a Tioga text viewer and play all the voice contained within its voice icons. In a similar way, making a selection in a voice viewer and clicking Play causes all the voice represented by that selection to be played. A position indicator, or *cue*, moves along the capillary display, indicating exactly the position of the voice being heard at that instant. (A cue is shown in figure 8.2, being the grey rectangle in the capillary). Playback requests can be queued up in any order from voice icons and from voice viewers, but the visual cue appears only when the voice being played was requested from a voice viewer. A variant of the Play command, selected by clicking on a different mouse button, plays all of the voice in a single voice viewer. We will also describe below some additional playback operations that are useful for dictation.

Editing is permitted while playback is in progress, so that the user can readily select voice regions for deletion, replacement, or relocation as the cue passes over them and their contents are revealed audibly. This has turned out to be a very convenient way to edit the voice. Alternatively, the user may mark points of interest for future attention as the cue passes over them, using any of the methods to be described below. Once a selection of voice has been queued for playback, this playback will proceed, regardless of whether a portion of the queued selection is meanwhile deleted from a voice viewer. This was the most coherent and instinctive behaviour we could find for this situation. The cue indicator will disappear during playback of voice segments that are no longer part of the viewer's contents, reappearing when 'visible' voice is reached again.

The complete mechanism for providing all the capabilities described in the last two paragraphs was a complex portion of the implementation of the voice editor, particularly since several operations can be accomplished simultaneously and therefore must be carefully interlocked. However the effort seems to have been well justified, since 'editing behind the cue' has turned out to be a fast and effortless method for performing simple voice editing.

8.5.4.2 User-Supplied Marks

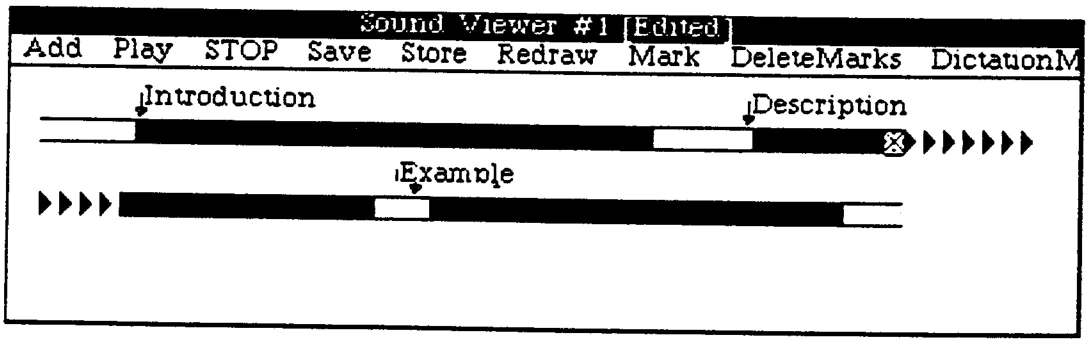
In Tioga, editing operations such as replacement require the user to specify two selections. Making selections is simple for visual media, but when these operations are to be applied to voice viewers it is harder for the user to identify the portions to be selected. If identification involves listening to the voice, then this in turn will involve making selections. There needs to be a way, therefore, temporarily to mark each selection once it has been located.

The user can select any point within a voice viewer and click Mark to place a distinctive diagonal cross within the capillary at that point (or DeleteMarks to remove one). Selecting a range of voice rather than a point causes Mark to place a cross at each end of the selection. When future phrase-granularity selections are made, the selection mechanism will treat a mark placed in the middle of a phrase just like a phrase boundary. This enables user-defined points of interest quickly to be selected as boundaries in addition to silence/sound transitions. These diagonal marks may be used for any purpose, but they are particularly intended for use as an aid for making multiple selections. The marks are temporary and lightweight—in particular they are not retained as part of a voice annotation when it is Stored or Saved and are destroyed when the voice viewer containing them is destroyed.

More permanent marks are also provided in this interface, in the form of textual annotation of the recorded voice. Selecting a position in a voice viewer and then typing some text causes that text to become associated with the selected voice. The text appears above the selected position, with an arrow pointing at the voice marked (as shown in figure 8.3). These textual annotations are more permanent than the cross markers in that they are retained along with the voice when the contents of the viewer are saved or stored and will appear again the next time the particular voice icon is opened. Very simple editing of these textual annotations is supported, being limited to appending further characters to the end of the annotation, together with deleting single characters and complete text annotations.

Through textual annotation of voice the user can mark any phrase with, for example, some key word or words that occur within the phrase.[5] This style of textual annotation is believed to provide much more useful cues for searching within a voice viewer than would an energy profile or other pictorial representation of the voice signal.

Figure 8.3: the voice viewer from figure 8.2 has been organised by the addition of permanent annotations. The user set a temporary marker (the diagonal cross within the capillary tube) while listening to the middle section, locating a point where new voice was to be recorded. The arrowheads shown indicate that this recording is in progress.



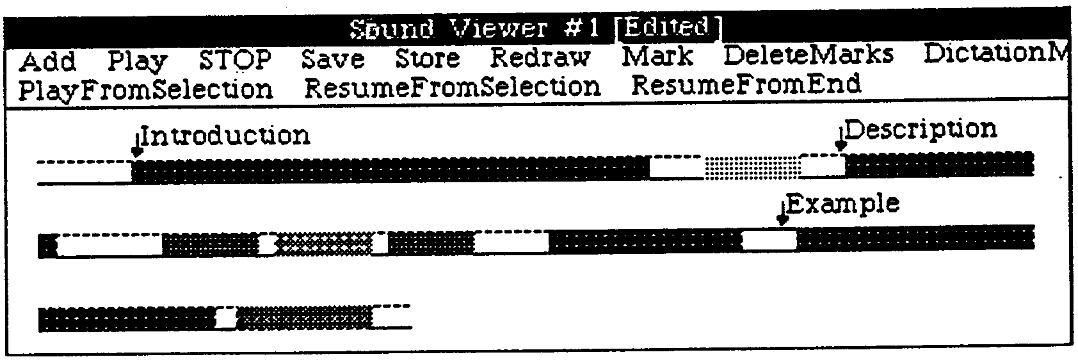
[5] Automatic speech recognition, however imperfect, might provide such keyword annotations automatically. Occasional mistranslations would be harmless in this context. This idea has been tested experimentally by the MIT Architecture Machine Group [Schmandt 81].

8.5.4.3 'Fresh' And 'Stale' Voice

Section 8.3.3 introduced the notion that regions of voice that have been inserted or modified recently, i.e. 'fresh' regions, are different from the remaining, 'stale', ones. The difference lies in the ease with which the user can re-input fresh regions. The Cedar voice editor makes novel use of colour to test this idea.[6] When a voice viewer is opened, the capillaries representing the voice appear in a dark red colour (i.e. red in places where they have so far been described as appearing in black: the white portions remain white). Suppose that a copying operation is now performed. The copied voice appears in its new position in a bright yellow. If another copying operation is then performed, the most recently copied portion appears in this bright yellow, whilst the previous copy changes to a less vivid colour.

A series of such colours was selected, ordered so as to suggest an aging process. The most recently added text appears in bright yellow, the next in a deeper orange shade, and so on until all voice of more than a certain 'age' will appear in the dark red. Like the temporary cross-shaped markers, these age indications are a volatile property and disappear from a voice viewer when it is destroyed. In early use of the prototype editor system, these colourful indicators of the locations of recent editing activity have reduced the need for user-supplied markers. Figure 8.4 attempts through the use of various shadings to portray in black and white the effect of these automatic freshness indications during an editing session.

Figure 8.4: several edits have been made to the contents of the previous voice viewer, resulting in this view. Shadings are used here to denote the colours that appear when the voice is displayed on a colour monitor. The most recently added voice is shown in the lightest shading, which represents a bright yellow. Voice unchanged from figure 8.3 is shown in the darkest shading, representing a deep red.



Sound Viewer #1 [Edited]
Add Play STOP Save Store Redraw Mark DeleteMarks DictationM
PlayFromSelection ResumeFromSelection ResumeFromEnd
Introduction
Description
Example

[6] Many Cedar workstations are equipped with a colour as well as a monochrome bit-mapped display. On systems with a colour display all voice viewers appear by default in colour. It will be explained below how the voice editing interface available for users with only the monochrome display differs from that described in this section.

8.5.5 Dictation Functions

8.5.5.1 Specialised Playback and Recording Operations

In a voice editor, "what you see" is very definitely not "what you get". It is what you hear that counts—the display only *represents* what is there. This is why playing back voice segments is such a commonly required operation during voice editing. In particular, the dictation of new, 'fresh', material produces some special requirements. The user may make a mistake while dictating or be uncertain whether the phrase just uttered sounded clear. It would be convenient to be able to replay what had just been spoken from some point a little way back, then to be able to resume recording either from that point ('recording over' what had been there previously) or from the end depending upon whether the existing version turned out to be satisfactory.

All of this can be done using combinations of the Tioga-style voice editing operations that have so far been described, but not always rapidly or conveniently. This voice editor therefore provides three operations to make it easy to use the editor as if it were (a rather enhanced version of) a conventional dictation machine. These operations are accelerators that combine simpler editing operations in specialised ways.[7]

The three functions are presented as an additional, optional set of menu buttons (shown as a second row of buttons in Figure 8.4). Clicking PlayFromSelection cancels any recording or playback already in progress and then plays all the voice from the current selection to the end of the 'freshest' section of voice in the viewer.[8] RecordFromSelection cancels any activity in progress, deletes all voice from the selection to the end of the freshest section, and then begins recording from the position of the selection. RecordFromEnd does the same thing, except that it does not delete anything and begins recording from the end of the freshest section.

These three functions are all that is needed to provide a high speed dictation facility. Typically the user records until (s)he makes a mistake or wishes to listen to what has just been dictated. In the former case, repeatedly selecting the approximate point with the mouse and clicking PlayFromSelection allows the user quickly to find the end of the last phrase which (s)he wishes to retain. Then RecordFromSelection is used to replace the remainder of the previously-recorded segment with new dictation. Where the user was not sure whether a phrase was spoken for example clearly enough, PlayFromSelection can be used to locate the phrase and play it back, followed by RecordFromSelection if it is to be rejected and replaced. RecordFromEnd would be chosen instead if the replayed passage turned out to be acceptable.

Since the version of the editor from which all the figures in this chapter were taken, a fourth button has been added to the dictation menu. CompressSilences scans a selected portion of voice for all silences of a greater length than l and replaces them with silences of exactly length l. l can be set to any value using the Cedar command interpreter tool. This accelerator is useful to

[7] It was explained above that Tioga includes many other such accelerators and also provides ways for users to define their own.

[8] More precisely, the region chosen by this command and the other two specialised dictation operations is from the current selection to the beginning of the next segment less 'fresh' than the one containing the selection, or to the end if there is no such segment. It is intended that the user generally selects the end of the freshest segment, usually the one just dictated.

a secretary who wishes to remove all the long pauses from a dictation before typing it out. A suitable value of *l* can transform isolated sentences into pleasant-sounding continuous speech.

8.5.5.2 The DictationMachine Button

This menu button, available in both text and voice viewers, is an accelerator that makes it easy to begin a dictation (for example, to dictate a letter that a secretary will later type). Clicking DictationMachine creates an empty voice viewer and immediately begins voice recording in it. The characteristic recording arrowheads appear in order to indicate progress. Once created, this viewer behaves just like any other voice viewer, except that it is not associated with an open voice icon until the user specifies a location by clicking Store.

Normally simple annotations to text will be entered as described in Section 8.5.1, without the need for voice viewers or editing operations. However, if during input a mistake is made, the user may find the dictation functions useful. Instead of stopping the recording and then opening a voice window, the user can click DictationMachine. This produces a new voice viewer, as above, whose only contents are the arrows representing the voice already recorded. Recording continues. The new viewer is a fully-fledged voice viewer and all the voice editing operations can be applied to it, for example to replay a section and then record 'over it'.

The way in which dictation into an existing voice viewer is performed has already been described: the 'fresh' voice is distinguished from older segments by colour so that selection for editing is easy. Furthermore, the dictation functions can be used, automatically selecting the end of the freshest segment.

8.5.5.3 Dictation on a Monochrome Display

In order to show 'ageing' of voice and to exploit the idea of 'freshness' on a monochrome display, different shadings or stipple-patterns could be substituted for the colours on the colour display.[9] In fact the Cedar device-independent graphics package does this automatically if the information to be displayed on a monochrome display includes colour specifications. However it is difficult to distinguish these patterns from one another, and more difficult still to use them to convey a progression of 'degrees of freshness'.[10] In addition, the combination of these shadings and the playback cue, cross markers and denotation of the current selection will confuse the user—this will be an overuse of the capabilities of monochrome shadings and symbols. Therefore it was decided not to show the freshness markings at all when using a monochrome display. Newly added voice fades immediately into its surroundings. The three specialised dictation functions always treat the voice viewer as a single segment of constant age, operating between the selection and the end of the entire viewer. This may seem confusing in that voice editing functions do not therefore behave identically in all viewers. However the behaviour of functions always corresponds to what is seen (for on a monochrome display the end of the newest segment will always be the end of the viewer). It is felt that the reduction in clarity caused by trying to display ageing information on a monochrome display is sufficiently serious to make this a better

[9] The monochrome display is of the one bit per pixel type and the possibilities are therefore limited to stipples and similar shadings.

[10] Figure 8.4 illustrates this point.

solution.

The DictationMachine button can be used within a voice viewer to prevent new material from merging indistinguishably with the old, when the user at a monochrome display stops recording to correct a mistake. Clicking DictationMachine during recording into a voice viewer transfers the expanding set of arrowheads representing the new material into a newly-created voice viewer. At the insertion point in the original voice viewer a simple (diagonal cross) marker is left, so that the new voice can be returned (by hand) to that point once the dictation session is complete. This dictation session may include arbitrary edits and additional dictations, since here again the new dictation viewer is in all respects a standard voice viewer.

The possibility was considered of providing an automated command for merging the results of the dictation back into the original viewer, but we were unable to find an interpretation for such a command that had an unambiguous and readily understandable user model. The behaviour would be clear if the DictationMachine button had been used only once, but repeated use of the button might produce an arbitrary tree of related viewers: the desired behaviour in this case is not clear. In line with the design principles of section 8.3, a simpler, manual approach was chosen, since the alternative would have been to present a command whose operation would not be clear under all circumstances.

8.6 Summary

This chapter has presented a voice editing interface which provides a new and powerful way to add voice handling capabilities to visual documents. In the design of the interface, the uniformity principle has been exploited where useful (e.g. in the operations to copy, replace, delete, transpose etc. voice selections) but careful attention has been paid to respects in which voice should not be treated like visual media. In particular, care has been taken to make simple voice annotations very easy and not intrusive upon the visual structure of a document. Since voice cannot easily be portrayed graphically, a number of methods have been employed to help the user establish context when editing voice. Special facilities have been added to aid composition of longer messages on-line, using the model of an idealised dictation machine.

9. Support for the Integrated Workstation

It was stated at the beginning of the previous chapter that the ISLAND voice architecture was designed to support not only a PABX but also higher level Integrated Services. The remainder of the work to be described in this dissertation shows how these services have been provided for. It concerns various facilities which we propose should be available at an 'Integrated Workstation' and a miscellany of things that are required in order to support those facilities. These topics are covered in this chapter in two main sections.

The first half of the chapter deals with how the voice editor described above would be supported in the ISLAND architecture. Implementation of such an editor raises several issues. First there is the sheer volume of storage required for voice. This motivates the implementer to avoid copying voice files from place to place unnecessarily, which has implications for the way in which multi-media files are both stored and edited. Secondly, voice manipulation within documents imposes real time constraints upon the voice filing system. This places demands particularly upon the ISLAND translator. Access to a multi-media document as a single and complete entity is the final issue of interest: this has further implications for the way the document is stored.

The ISLAND architecture is intended to support a variety of user level Integrated Services, based upon the multi-media capabilities available in a workstation. Having discussed the important area of multi-media documents, the remainder of the chapter considers a variety of other services. After a review of various useful workstation facilities, hardware is described which supports applications that require image-handling capabilities. Use of this hardware has been investigated to support the display of complex document structures and as a basis for services involving image coding and transmission. The final area reviewed is control of the telephone system from an enhanced interface on the workstation. Underlying mechanisms to support this control are discussed.

9.1 Managing Integrated Documents

The voice editor interface was presented above without reference to the architecture which supports it. This was deliberate, for two reasons. First it is beneficial to consider the type of interface that would be most helpful to the user and only after that to address the issues of how to implement it; many interfaces suffer from the inclusion of facilities which were easy to implement but which destroy the coherence of the interface. Secondly the editor was implemented in the Etherphone environment. None of the facilities in the editor rely on that environment[1] and it is important to this thesis to scrutinise how well the ISLAND architecture would support the editor. The components which will support it have already been described in previous chapters. The concern here will be to piece the components together into one coherent support system.

[1] By the Etherphone environment we mean the facilities for recording, playing back, storing and managing voice files. The editing *interface* clearly did rely upon and was heavily influenced by the capabilities of the Cedar programming environment.

The following three facts have a strong influence on the way in which voice in documents should be handled.

i) Recorded voice represents a large storage requirement (compared with data).

ii) Most stored voice (e.g. in the form of simple annotations in documents) is recorded by one individual and played back by another. The individual who recorded it often does not wish to hear it back.

iii) If a person in the process of recording voice does wish to hear it back (when using the voice editor as a dictation machine for example), the voice should be replayable very soon after recording stops.

The impact of these three points should become clear in the following discussion.

9.1.1 Storage Requirements

As noted above, the first issue that confronts the implementer of a voice editor is the sheer volume of voice samples that have to be stored and manipulated. Whereas text is generally stored using 6 to 16 bits per character, a second of voice, encoded by the standard P.C.M. method, amounts to 64 kilobits. There are a variety of other coding methods that will compress digitised voice substantially below this density. They are attractive largely because of the cost of disc storage but naturally the greater the compression ratio the more expensive the hardware required to compress and reconstitute the voice. For acceptable sound quality, the storage requirement per second is still considerable. Many of the more efficient compression algorithms assume that they are operating on a single human voice and are hence unacceptable for the user who wishes to record a few seconds of music or even two people speaking simultaneously. Users may chose to annotate with voice rather than text because the results have a more personal feel: that feeling will be lost if the compression method severely affects the quality. In summary, use of heavy compression may restrict applications of our system just as in chapter 4 it was seen that it restricted the design choices available to the builders of the ARPA voice conference system. As storage becomes cheaper and people's expectations of sound reproduction quality increase, we may wish to use improved compression techniques as a means of storing better quality sound in the same space as before [e.g. Westall 85] rather than for using less space to store the same quality as before.

As well as the total volume of voice stored, the granularity at which stored voice is manipulated will be of relevance to the implementer of a voice editor. This will depend not only on the density of samples per second but also on the average duration of an utterance. The editing interface discussed above aimed to steer the user towards handling whole phrases rather than single words. This clearly increases the length of the average segment of voice samples involved in a move or insert operation.

9.1.2 Storing Voice Annotations

Most of the document structures reported in the literature which allow voice messages or annotations to be included within documents use a more complex representation for the document than a linear sequence of textual characters and voice samples. One common approach, used in Diamond [Thomas 85], MMM [Reynolds 85] and the ODA [Horak 85], is to define structure and content files. For each document there is one structure file, which defines the way in which a series of elements making up a document are related, including their relative positions when displayed or printed. The elements are held in content files each containing data representing only one medium, such as text, graphics, voice or bit-mapped images. The structure file contains only references, or pointers, to these files.

In Diamond the pointers are termed 'citations' and are used as links between all components of a document which are of different media. Some of the justifications given for their use are (i) the size of a multi-media document (particularly the image and voice components) compared with plain text documents and (ii) the tendencies of workstation users supplied with mail folders to retain individual copies of mail sent via distribution lists. Significant space in the filing system can be saved when a document is copied and edited by allowing only pointers to unchanged components, rather than the components themselves, to be duplicated in the new version of the document.

The systems cited above are all general-purpose multi-media document handlers. In a system where the capabilities were to be limited to the addition of voice annotations to simple textual documents a pointer structure would still be appropriate. Such a structure is typified by [Maxemchuk 80b], in which a document is represented as a sequence of text characters and pointers to voice segments. The reasons given for the use of pointers in this case are (i) that the voice samples themselves need to reside on a separate fileserver, due to real time manipulation requirements (ii) to avoid copying voice segments when editing documents.

Avoidance of unnecessary duplication of voice segments is important in any filing system, including that for ISLAND. Use of pointers to voice files has a further advantage for editing in an environment like either that of the ISLAND or the Etherphone designs. The workstation need never manipulate voice files directly. The only time that the voice itself needs to be accessed, as distinct from the pointers being manipulated, is for recording and playback. Consider how the simple annotation implementation interface described in section 8.5.1 might be implemented within ISLAND. Suppose that every recording operation produces a new file. The translator, upon each STOP command, returns to the editor running on the workstation a pointer to the file. The editor may need to be given a characterisation of the file, in addition to the pointer (for example it would need a list of the silence and sound periods if a display of the form described in the previous chapter were to be produced), but it will never need to access the contents of the file. All editing commands can be implemented as manipulation of pointers, as will be discussed below. The only time that the contents will be accessed (other than for remote document transfers, which will be discussed later in this chapter) is for playback: playback is accomplished by sending pointers to the translator, which then transfers the data from the voice fileserver directly to the phone.

This minimises the amount of data movement. The availability of access through the translator to a common voice filing system from any phone is an important part of this minimisation. A structure where users recorded and played back voice only via their own local storage would involve copying (and unnecessary duplication) of a lot of data. In the Etherphone environment workstations have local discs, on which copies of text files are held whilst they are being edited.[2] However a special global fileserver is used to store all voice messages at all times. A simple

[2] Such files are stored more permanently on global fileservers.

message is recorded straight onto that fileserver from a phone by one user and then replayed directly to another phone by another user, to avoid the transfer of voice from one local disc to another. (The Etherphones and their associated workstations are separate physical entities—there is no faster transfer path between a phone and 'its' workstation than between a phone and the voice file server.)

An obvious consequence of the use of files which point to other files is that garbage collection will be necessary. It may be simple or complex to implement, depending on the document structures allowed by the system [Thomas 85].

9.1.3 Support for Voice Editing

A simple system of pointers can avoid duplication of voice files which are copied wholesale from one document to another, but not of those which are edited in the manner that the interface in the previous chapter allows. For example, [Maxemchuk 80b] provides an extra level of indirection in the voice fileserver. Pointers held in text files are used to access linked lists of disc blocks and these in turn are used to access the voice itself. This avoids copying voice samples when voice segments are split or concatenated, which in itself is important. It does not, however, allow two linked lists to point to the same disc block, so that samples need to be copied when two documents access common voice segments but in different sequences.

One way to avoid any duplication of voice samples is to use a data structure like that employed in some filing systems or 'code control systems' [e.g. Tichy 83, Leblang 85]. Conceptually, there are two ways to implement any editor. The first is to make a copy of a file which is to be edited, alter the contents of the copy and then rename the copy in place of the original at the end of the session. The other approach is to maintain a permanent data structure. Each editing operation produces a new entry in the data structure which records a change to the file. The current state of a file can be obtained by amalgamating components of the structure.

Text editors use a wide variety of methods and data structures within their internal workspace whilst operating on a file, but these are usually not reflected in the files which are stored in the filing system. Usually all files are completely separate and independent entities. However a study of modifications to source files in a programming environment [Leblang 85] suggested that there is a strong case for storing different versions of the 'same' source file using one data structure. It was found that using this method the storage requirement was only increased by 1–2% of the size of the most recent version for each additional version of the file retained. Systems implementing these ideas use either one file containing both all the text in the different versions and some path information to distinguish them [Leblang 85] or combine a file containing all the text with a separate data structure of pointers into that file [Tichy 83].

It is clear that the efficiency of an external pointer structure compared with multiple copies of a file increases as the average size of a move, replace or other editing operation increases. It has been noted that text files are often edited character by character, whereas voice is replaced at the phrase level. This approach is thus particularly useful for voice files. One might hold an independent linked list of pointers for each occurrence of voice within a document, but it is more attractive to represent different edited versions as entries in a hierarchical data structure, representing each version as the result as operations on other versions. One might loosely term this data structure a 'database', but this is not to say any more than that it is a self-referential data structure to which access may be gained by presenting some token. In a code control system along the lines of [Tichy 83] the name and version number of a source file is such a token. In a voice editor the tokens would be pointers from text or structure files. These pointers would represent arbitrary sequences of voice segments and editing would be performed by requesting generation of a new pointer which is the result of combining other pointers in some way.

9.1.4 Real Time Performance

The major aspect of real time performance in the ISLAND architecture has been discussed in earlier chapters—that all voice-handling components in the system must be able to handle a voice packet per voice stream every 2ms. The translator shields the fileserver from such demands. It provides sufficient buffering per stream to compensate for fluctuations in the real time response of the fileserver and to allow communications with the file server using large block sizes.

Another real time demand, which arises in the context of the voice editor, is that the delay between stopping recording a file and playing it back must be kept small. [Nicholson 83] suggests that it should be no longer than 100ms for comfortable use, although our experience with the voice editor described in the previous chapter is that delays of 300–500ms are tolerable. The delay is most noticeable in the context of the dictation machine, where near instant turn-around has a great effect on the fluency with which the system can be used. If delays are too large, much of the advantage of using this paradigm is lost.

In chapter 4 it was stated that lightweight primitives for access to voice files can make fast response possible. For example fast turn-around from recording to playback can be facilitated if only the translator has access to a file whilst it is initially being written. That access can be made very free, for example allowing both reading and writing simultaneously without consistency checking. For fast turn-around the translator could replay parts of a file before other parts of it had been written through to disc, provided it was capable of spotting portions not yet written through and of playing these back from its internal buffers. This latter technique would not, however, be very useful in an implementation of the editing interface described in the last chapter. In that interface, the user is entitled to expect the arrows which indicate 'recording in progress' to be replaced by normal capillaries as playback occurs—this happens in the Tioga implementation. If the voice is not yet fully written to the fileserver then the information needed to draw the capillaries is not available, unless it is obtained from the translator. Such a data path is best avoided: whereas it might be reasonable for the translator to play back voice from its own cache, collection of information from different system components depending on fluctuating performance of the system is an unpleasant idea. (The information could of course be obtained every time from the translator. This adds to the transaction another function to compromise the performance of its primary task, but is a possible solution if no other approach gave adequate turn-around of files.)

It was partly the problem of obtaining sound/silence profiles in a clean way in the Etherphone environment which led to every record/resume request in the Tioga voice editor creating a new voice file. Resumption of recording in the same file when a dictation was paused or rewound and then continued was fraught with problems like the above. If a file is created for each record/resume request then once it is completely recorded and written fully to the fileserver, no further write access is expected on it (except if partial garbage collection of the file is performed) and a sound/silence profile can safely be delivered.

A similar approach can be pursued in the ISLAND design. When the translator is called upon to record some voice, it should request creation of a file by the filing machine, access permission for the file being set such that the filing machine itself cannot open it. (For fast response, a 'stock' of such files can be created in advance of their being required and be held ready at all times.) The translator can then write data to the fileserver directly using lightweight, fast primitives. As soon as recording stops or pauses and the data has been fully written out to the server, access from the filing machine can be reinstated. The translator will continue to use its lightweight access but only to read the file. Such access cannot be dangerous in parallel only with reading via the filing machine.

9.1.5 Garbage Collection

In any pointer-based document system there needs to be some method of garbage collection. If the editing facilities discussed above are implemented the task of garbage collection will not be trivial: it is important to outline the problems here.

Suppose that a trace-and-sweep collector is to be used. This implies first that it must be possible to identify all pointers to voice by scanning the document system. If all documents are based around identifiable structure files of well-known format then this is not difficult. If files may consist of text characters and voice pointers, or equally of binary images, and if these types are not easily distinguishable then scanning is harder and likely to be messy. In the latter case a database, to which all voice pointers refer, may be used to contain the problem.

Unlike the case of Diamond, the voice pointers in the system advocated here will not be references to whole files. Voice is never to be copied when an editing operation is performed. Also each AddVoice command will produce a new and separate voice file. It follows from these two facts that pointers will, either indirectly through a database or directly, be references to sections of files. Specifically, in the case of the Cambridge Fileserver they will be (PUID, subrange) structures. As the result of successive voice editing operations only certain portions of a file may be accessible; the rest will be garbage.

If only completely inaccessible files are collected then a large amount of spurious data may remain. This is especially true if the dictation operations described in the Tioga interface are used frequently. When RecordFromSelection is clicked the tail of the file just recorded will immediately become garbage, emphasising the need for partial collection of voice files. On the Cambridge Fileserver this can be divided into a special and a general case. The special case is where only the tail of a file is garbage. The Fileserver provides an operation to truncate a file. It is worth implementing this special case within a garbage collector on account of (i) the correspondence between occurrences of the case and the way in which dictation functions are used in the editing interface that we have extolled in chapter 8 (ii) the complexity of the general case.

The only general way to compact voice files which are accessible only in part is by copying these files. A copy will have a new PUID. Thus all pointers to the voice will need to be altered. In a simple scheme the subrange information will also need to be changed.[3] This is where a database, or some other self-contained structure giving an indirection between the voice pointers held in documents and (PUID, subrange) specifications, becomes an important requirement. Without such a device, whenever garbage collection took place any files affected would need to be altered, which is somewhere between unpleasant and impossible for most applications. Using a database approach, the whole process can be made transparent and 'atomic' by copying voice files into compacted forms, updating database entries to point to them and then deleting the old voice files.

One trap that must be avoided in devising the garbage collector is that of deleting portions of a document when it is temporarily held partially outside the filing system. This corresponds logically to holding a lock on a conventional file but needs to be implemented differently because of the way in which files other than directory files can refer to one another. If lock is held on a file containing pointers, each entity pointed to needs to be identifiable so that it can be regarded by the garbage collector as locked. This could be done by scanning each document as a part of taking a lock out on it, by scanning all locked documents as a part of garbage collection, or by holding the information in some independent data structure.[4] There are subtle problems in the case of pointers moved from one document to another and held only in the workspace of an editor, but these will not be addressed here.

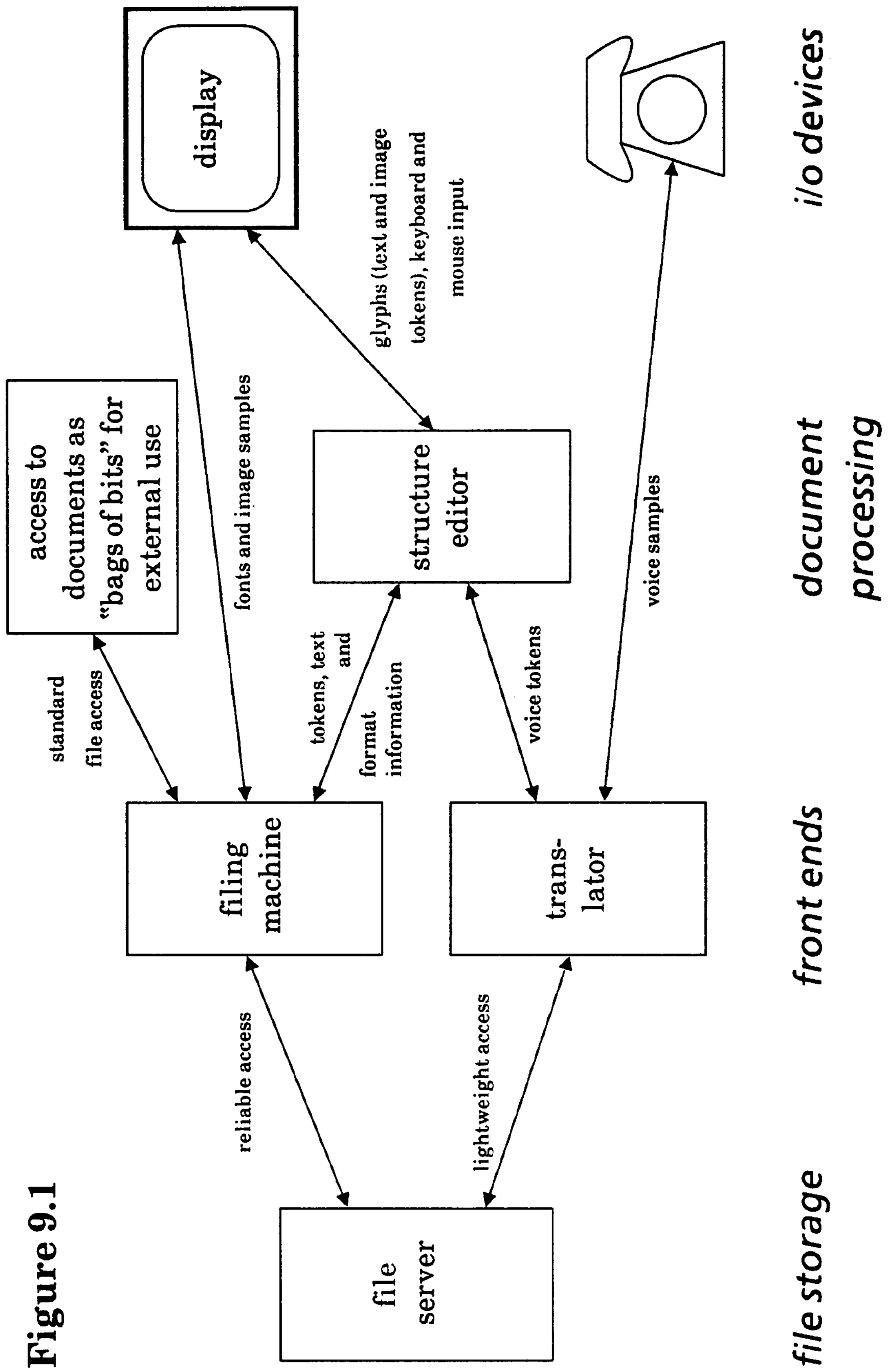
9.1.6 Access to Complete Documents

It was claimed in chapter 4 that a structure of one fileserver with two front ends has advantages over the separate voice and data fileservers used for example in the Etherphone system and by Maxemchuk. To review the considerations which led to such a structure, one of these two approaches is required because parts of a fileserver, most notably its network interface, would be unable to handle both the bursty data traffic and time critical voice traffic. Also two modes of file access are needed—reliable for data and fast for voice—and these can conveniently be provided by either two front ends or two fileservers.

The advantage of a single fileserver lies mainly in the ease of packaging a complete document. There are occasions when copying files containing pointers is not possible, so that a single, complete and self-consistent structure must be assembled. The most obvious scenario is that of two ISLAND environments located at different sites, each with an independent fileserver and complete with voice file database etc. If a document is to be transferred from site to site, it will need to be sent as one package. All pointers will need to be converted into the voice samples they represent.

[3] The Fileserver interface specification does have a feature which can be used to avoid the subrange information being altered. Files can be created with any size specification, but none of the data within them need actually exist. For portions of a file where data has not been written the fileserver reads back a pattern of data which was specified when the file was created. There is no command to discard data in a part of a file once it has been written, but files could be copied for compaction using the same total length of file and relative positions within it of valid data as in the original file, but with the parts being garbage-collected simply not written across.

[4] Strategies for solving this problem are discussed in [Terry 87].



display
glyphs (text and image tokens), keyboard and mouse input
fonts and image samples
access to documents as "bags of bits" for external use
structure editor
voice samples
standard file access
tokens, text and format information
voice tokens
filing machine
trans-lator
reliable access
lightweight access
file server
i/o devices
document processing
front ends
file storage

Figure 9.1

Standards for the interchange of multi-media documents are currently being developed: the document format of the Office Document Architecture, or ODA, already mentioned (which is a combination of ISO, CCITT and ECMA standards work) is set to include voice in its definition within the next few years. The details of how a multi-media document might be represented for transfer are not of concern here. The relevance to this discussion is that where multi-file documents have to be 'bundled' for dispatch as self-contained entities, it is a great inconvenience if only part of the document is accessible at some times. This is not so much a philosophical problem as a practical matter—a cause of unnecessary software complexity. The avoidance of software complexity is something which has been an important consideration throughout the ISLAND design. For example it was a motivating force behind the partitioning of the basic ISLAND voice system into small servers. It is well known that a large component of the software in a distributed system is concerned with handling error conditions in other parts of the system, and that comparatively little is actually devoted to handling error-free cases. The introduction of a new series of possible errors due to only half of a document being accessible should be avoided if possible.

The transfer of documents need not be performed in a bundled manner: some researchers advance the argument against it that the recipient may not have any facilities to handle voice, in which case (s)he will not wish to wait whilst large amounts of useless data are sent or for that matter wish to store them when they are. In the local area context of Etherphone many Tioga users had no voice facilities: the pointer mechanism was therefore arranged such that where such users accessed documents containing voice they would not see any trace of the voice at all. To avoid such users being sent the voice components of documents in a global environment the users who desired the voice could 'pull' it from the filing system whence the document had come. Assuming the recipient's environment knew whether voice playback could ever be possible, this could be done automatically when the file arrived, to avoid a long wait when it was first opened whilst bits of it were fetched, but this implies the existence of a detailed filing structure with a global name space. The schemes above for garbage collection could also no longer be made to work simply.

The isolation of separate domains for documents, with no cross references between them, seems the most practical approach. Each site can use its own local file representation, with a standard transfer protocol used between them. This is the approach used in the ARPA multi-media document experiments [Reynolds 85]. The approach parallels the hop by hop approach to voice protocols discussed previously, where the working environment for the local area is specifically tailored to that local area: this avoids temptations to compatibility with for example wider area protocols. In both cases such compatibility would compromise the aims for which the LAN environment is designed.

9.1.7 Structure of a General Editor

It is useful here to sum up and generalise the components which comprise an editor according to the discussion above: these are shown in figure 9.1. This figure shows logical decomposition of a multi-media editor into functional components and also the information which passes between these components. It is hoped that it will reinforce how the editor should fit into the ISLAND architecture.

The structure of a document can usefully be separated from its content, along the lines of the ODA, Diamond and similar designs. Developing the ideas in section 1.4.5, we define an object which operates on the structure of a document only. This is shown as the 'structure editor' in the figure. It accesses the structure of a document held on the fileserver through the filing machine. According to the structural information, the editor causes the components of the file to be sent to various i/o devices. Voice tokens held within the structural information are passed to the translator at appropriate times, causing sounds to be produced by the phone. Similarly textual and graphical objects may be represented as tokens which are sent to the framestore/display. (These tokens might well be ASCII characters in the case of text. However tokens may alternatively refer to files which contain text, to avoid merging the structure of a file with its content.) They may be displayed using fonts or digitised graphical objects which are picked up from the fileserver using the filing machine as a front end. Images to be displayed will also be referred to using tokens, as will be discussed in chapter 10. Input (in the forms of keystrokes and of mouse clicks and positions) will be sent from the display to the structure editor, mouse positions being specified either in relative coordinates for various windows or by reference to glyphs displayed on the screen. (The former is much more likely as it permits the structure editor to interpret how for example a position which locates several overlapping glyphs is handled.) The structure editor and display need not actually reside on separate machines, but they have been partitioned in this discussion such that bandwidth required for communication between them is low. This is in order that the ideas of the terminal concentrator might be applied to high resolution terminals, as discussed in section 1.4.5. The display would be a cheap machine located on the user's desk, whilst the structure editor ran in a more general-purpose and expensive server, located elsewhere, in a bank.

Figure 9.1 also shows a bundling function—a software module which assembles complete documents as single entities according to some document transfer standard for shipping to remote filing systems.

9.2 The Integrated Workstation and its Terminal

The previous sections have discussed how multi-media documents can be stored and accessed. The remainder of this chapter will address the broader subject of the workstation on which a multi-media interface is presented to the user.

Most of the important issues relating to workstations in the ISLAND environment were outlined in chapter 1. The current capabilities of commercial workstations are listed there. The hardware of such workstations normally includes a pointing device, such as a mouse; a high resolution bit-mapped display; a network connection and possibly local disc storage. We have discussed local and remote storage in this chapter. Section 1.4.5 proposed a split of the Integrated Workstation into processor and display, analogous to the split between the Cambridge processor bank and its remote terminals. The display would become principally an image handling device and its hardware design would reflect this.

The purpose of the Integrated Workstation, as stated in section 1.2, was to provide services which involved manipulating various media—voice, text, graphics and images—together in a coherent way. Discussion of this will concentrate on three main aims. The workstation should be integrated with the phone system in the sense of being used for the control of telephony functions. All of the above media should be integrated into a single document architecture. Integrated communication facilities should be provided, allowing two workstation users to communicate using voice, images and shared text files.[5] The next section will review these various facilities and their uses.

9.2.1 Workstation Applications

Multi-media document manipulation has already been discussed at length in this chapter. It has been noted that the workstation display is an i/o device for visual components of the document, just as the telephone is an i/o device for sound. It has been explained how instructions would be sent to the translator to play back a particular segment of sound, as specified by some pointer. In the same way instructions will be sent to display some portion of an image at a particular location or to display a character in some font at some position. The display will need to be able to fetch images and fonts efficiently from a fileserver across the network.

The workstation should support telewriting, or 'distributed blackboards', by which is meant that two remote workstation users, each armed with a mouse, can draw on a single screen image as if they were standing together at a blackboard and each held a differently coloured chalk. Section 1.4.5 referred to a study which suggested that the combination of a voice channel, telewriting and single image transfer formed a very useful means of communication. To support this the display needs to be accompanied by a camera or other scanning device. The ability to scan and transmit images of itself enables the functions of facsimile machine to be incorporated into the workstation.

There are a variety of different image transmission facilities which have obvious applications. As well as single frame transfers for document exchange etc., slow scan video is useful for surveillance applications. Slow scan links installed together with telephone links have been shown to be useful for example in business negotiations. There are of course also requirements for moderate to full frame rate video transfers, for video conferencing.[6]

The heading of integrated communication also includes distributed access to documents—there have been experiments where several users at different sites have combined, using audio conferencing and distributed editing capabilities, to edit a document [Engelbart 82]. Also a system for distributed access to a multi-user engagement diary has been demonstrated [Sarin 85]. Unified views of a common desktop, available from a series of separate workstations, is another topic which is currently interesting researchers. These applications, although they come under the heading of new services from integrated workstations, do not make special demands on the

[5] Some of the issues raised in trying to communicate using two different media simultaneously concern how much drift and difference between transit times can be tolerated for media being transmitted in parallel. We will not pursue these issues here. There is a lack of facts to back up such a discussion and opinions seem to be hotly contested. (For example some people hold that in a video conference a mismatch of one second between delivery times of voice and video is perfectly satisfactory, whilst others find it intolerable.)

[6] Many people, including the author, feel video conferencing to be overrated. People find it difficult to resolve conflicts in a video conference meeting: this has been complained of sufficiently frequently to provoke commercial psychological studies on the reasons for it. Video conferencing does however seem to be of considerable value to avoid pointless meetings, a manner in which it is used by several U.S. companies. (The video conference is used to ascertain whether there are enough issues to be resolved to require a physical meeting to be held.)

ISLAND system infrastructure and will therefore not be discussed further.

The final set of applications of interest here concerns controlling the phone system. The 12 or 13 keys on a phone instrument constitute a very poor interface compared with pop-up menus and other methods of control common on current workstation systems. Given powerful means of issuing commands and displaying information, sophisticated features such as call filtering can be made available to the user. The interface of a normal phone could not accommodate such sophistication in a sensible and usable way.

9.2.2 Hardware Support

If the workstation is separated into a processor and a display, the primary purpose of the latter will be high speed image manipulation. The next few sections discuss the design of hardware for this purpose. Most workstation displays of the current generation have a resolution of around 1000 by 1000 pixels, with one bit per displayed pixel. Such resolution is necessary to display e.g. a good quality A4 page of text. One bit displays are unable, however, to give an adequate rendering of good quality images, particularly of people or moving live scenes. The combination of such a large resolution and several bits per pixel represents the limit of technology at present. This technology was not accessible for ISLAND use. Since manipulation of high quality text is no longer an area of much research interest, we therefore designed a framestore of several bits per pixel and reduced resolution, as a vehicle to demonstrate various types of media integration.

The store was designed in collaboration with Seescan Ltd., a Cambridge company specialising in image capture and processing applications, who built the store. It has several architectural features of interest for the applications listed in the previous section: these will be discussed below. The store works at resolutions of 512 by 512 or 512 by 256 pixels. The number of bits per pixel is variable: configurations are available (and can be selected in software) with either 8 or 24 bits per pixel. Images are displayed in RGB format, driven through lookup tables (often referred to as 'colour palettes'). The store has two lookup table options. In 24 bit per pixel mode three 8 bit planes drive the three colour outputs, each through an 8 to 8 bit lookup table. In 8 bit mode any of the three 8 bit planes can drive the full 24 output bits (8 per colour) through an 8 to 24 bit lookup table. (We designed but have not built a 14 to 24 bit lookup table mapping from two planes to 3 colours: this would be a more generous size for the applications to be described below.)

9.2.3 Windowed Displays

The first lookup table configuration is useful for imaging applications, which are not relevant here. The second is designed to support combined windowed displays of differing media. For example when an integrated document is on display there may be windows showing text and graphics (probably each using one bit per pixel, although anti-aliased windows are a possibility) and also multiple bit per pixel ('n bit') images. The lookup tables can be used to display such a screenful correctly, in conjunction with the allocation of different bits within a plane to the text and images and the assignment of some bits as mask bits to indicate where the windows are located on the screen. The lookup tables can also be used to set which windows are in front of other windows.

To demonstrate these capabilities two TRIPOS tasks have been produced. (The framestore is a VME peripheral and was designed to be controlled by an ISLAND server. It can therefore be run under the TRIPOS operating system.) The TRIPOS tasks, which were designed by the author and implemented by Nick Brasier as a Computer Science Diploma project, are a window manager and a console handler. The window manager handles input from a mouse and accepts requests to create, move, resize, update and prioritise windows. Requests to create windows come from client tasks, each request being accompanied by a 'redraw procedure'. Subsequent requests from the client to update the window, as well as actions from the mouse resulting in movement of a window, cause the redraw procedure to be invoked. The behaviour of the redraw procedure is unknown to the window manager and peculiar to each type of client. The procedure will be invoked with parameters which specify a client-supplied data structure, the relative coordinates of a part of the window to be redrawn and the absolute screen coordinates onto which they map. All coordinates are supplied by the window manager.

This split between window manager and its clients is designed to

i) allow the manager correctly to synchronise requests from various clients to update the display

ii) avoid clients needing to know the absolute position and plane where their windows are located at any time

iii) enable the window manager to handle windows with any arbitrary content.

One client has been produced for the manager. This is a console handler which allows standard TRIPOS applications to use the framestore as a console, with one window per task. The manager is designed so as to be capable of supporting other clients, such as image handlers and more fancy text and graphics handlers. Where images are to be displayed together with other media, 8 bits per pixel becomes a fairly small number and the 14 to 24 bit lookup table becomes highly desirable.

The framestore lookup tables can be rewritten very fast, so that reordering of the display priority of windows occurs very quickly. There are also 16 completely separate lookup tables provided for each operating mode of the framestore, the table in use being alterable by a single register access. This can be used to make priority changes instantaneous. It can also be used for simple animation sequences, where for example a different image is stored in each bit of each plane. If more than one plane is fitted (as described below), one plane can be on display whilst the contents of the next are being calculated, to give potentially endless animations.

None of the software structure outlined above is particularly innovative—similar structures are used in various other window systems. The object of producing the two TRIPOS tasks was to show the window handling capabilities of the framestore.

9.2.4 Image Transfer Services

Many of the features of the framestore were designed to support image transfer services. The framestore will capture images from either RGB or monochrome video sources (and from several other types of source not relevant here) at full frame rate (i.e. one frame captured every 40ms). A maximum of six 512 by 512 pixel by 8 bit planes can be installed in one system, to support various services which require multiple storage planes, including those which call for successive frames to be captured into different planes.

All of the framestore image memory is fully dual ported. In other words, as well as video rate access to images for display and capture via a special-purpose high speed video bus, access to all image memory is possible across the VME bus at its full speed. One of the design features of the high speed Ring interface described in chapter 7 is that it can transfer data directly between the Ring and VME memory. The framestore was designed such that this interface could therefore transfer data directly between the Ring and image memory, to support high speed image handling. This is significant for fast manipulation of complex document structures. It is also very important for video transfer services.

The framestore design makes it possible, by the addition of a little extra hardware, to display one image whilst capturing another. In particular, the data capture and display paths on the video bus are fully duplex right up to the image planes themselves, to facilitate capturing a different plane from the one on display. This is a very unusual and powerful specification item for a framestore: its use can be seen from the following examples:

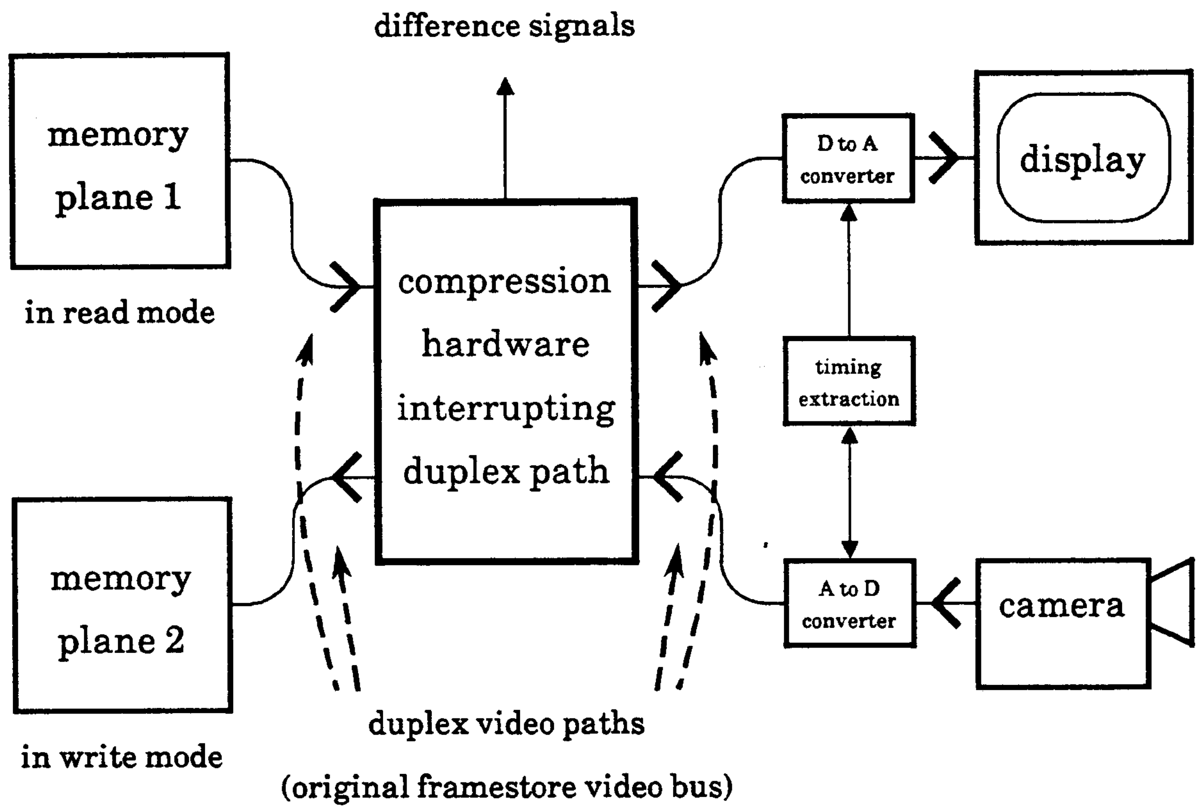
i) a fully duplex slow scan link between two sites employs two framestores. Each has four image planes fitted. At any time one plane will be displayed on the framestore's display. The second will be in process of reception of another image across the network. The third will contain a recently captured image or an image may be being captured into it. The image in the fourth will be in the process of transmission across the network. As each complete image is received or transmitted the functions of the first or second pairs of planes respectively will be swapped. In this application only one of our framestores need be installed at each end of the link, whereas nearly all other systems would require two per end.

ii) difference calculations are performed on a changing image in order to reduce the amount of network bandwidth used to transmit moderate to full frame rate video. Each new image captured by the store is compared with the previous and the differences (if above a preset noise level set for the camera or other imaging device in use) are sent along the link.

In a very simple scheme of type (ii) above the available link bandwidth per frame time would be used to send as much of the changed image per frame as possible.[7] This would clearly require special-purpose hardware, in order to perform video-rate difference calculations. The architecture of this framestore is specifically intended to make such hardware simple to add. In particular, the hardware can be placed in the duplex path between memory planes and camera/display unit, as in figure 9.2.

[7] In practice, particularly on WAN links, it is difficult to obtain a high, variable rate of bandwidth with sufficient grant-on-real-time-demand capability to allow transmission of a constant quality of image regardless of picture activity (although some researchers are now starting to address the use of networks in this manner). Many CODECs have therefore been built with an image replenishment rate which falls when the activity level increases [e.g. Chiariglione 82].

Figure 9.2



In this position the additional hardware can compare the previous frame, which the framestore thinks that it is 'displaying' (see figure 9.2) with the new image from the camera. It will write the new image into the second memory plane and show it on the local display. The difference information computed to be sent to the remote site will also be output, by a completely different route. It will be noted that the process as so far described has made no use of the VME bus. The VME bus is too slow for video-rate difference calculations. However information to be sent to the remote site will require a smaller bandwidth. The VME bus is available in this design to be devoted exclusively to transmission to the remote site, probably by means of high speed network interface hardware.

In the case where the image has changed too much for all the differences to be transmitted in one frame time, the hardware will output only a fixed amount of difference information. For areas of the picture where information is not being sent, unchanged portions of the old image will be written both into the second plane and to the video display so that both display and second plane give a true record of the image as held at the remote site. Thus the correct starting point will have been set up for operation on the next frame. Clearly each frame period the roles of memory planes 1 and 2 are reversed.

In such a simple 'conditional replenishment' scheme as the above, the perceived frame transfer rate would be a function of the amount of activity in the picture at any time. (For a typical video conference between two studios, in each of which a camera is focused upon three people sitting adjacent at a table, the percentage activity in the picture will be fairly small for most of the time. Most complex video CODECs, such as those produced by the European collaborative videoteleconference project COST211 [Chiariglione 82], rely on this for normal working.) An alternative approach for use with a fixed bandwidth transmission channel is to transmit a picture whose quality varies according to the activity therein. Conveniently, the eye is much less discriminating of moving images than of stationary ones. We have therefore experimented with various coding algorithms employing partial difference coding and trading resolution and quality rather than replenishment rate against picture activity. These algorithms call for a FIFO delay line to take advantage of the two dimensional correlations found in most images. They work on a line scan basis and therefore can be supported by the framestore hardware.

They show promise not only for full frame rate video conferencing but also for applications where for example a user with a display is accessing a remote image database and wishes to glance through the database as fast as possible in search of a familiar face or object. The reason they seem useful compared with methods like quadtree encoding [Samet 84] is that they present some fine detail of the image at the beginning of its reconstruction and this, even though accompanied by much noise, enables the picture to be recognised.

Not enough work has been done on refining these algorithms to merit their being presented in this dissertation. However they indicate that line scan methods which exploit correlations can produce worthwhile compression ratios. This is of interest since line scan CODECs are rather simpler to build than conventional (e.g. CCITT standard) CODECs. The framestore structure was designed to minimise the additional hardware needed to build such a CODEC.

9.2.5 Control of the Telephone System

The ways in which the ISLAND workstation display and the infrastructure beneath it support manipulation of multi-media documents and transmission of video between workstations have now been described. All that remains in this chapter is to consider briefly the control of telephony functions from the workstation.

It has already been asserted that the phone is entirely the wrong interface from which to control the sophisticated functions available from modern PABXs. A modern PABX typically comes with a thick instruction book explaining how many immemorable sequences of digits can be used to access a large number of potentially useful functions. Most people will either not read the book or forget most of its contents. The interface provided by a phone is a bad one because

i) it is limited to a numerical keypad, some tones and in special cases a few lights

ii) a command structure based on sequences of digits is unlikely to be memorable

iii) in general the caller who is moved through for example a sequence of call forwarding set up by some other individual(s) is given no control over and for that matter no idea of the transitions through this sequence.

In some PABXs attempts are made to provide a better interface by adding to the phone—usually adding dedicated buttons and display segments. Objections to this approach have already been explained. The phone becomes more complex; if features are bound into the phone then it needs to be modified every time the system is upgraded.

There are two ways around this problem. Simple demonstrations of each have been produced within ISLAND. In each case the demonstration was designed by the author and implemented as a student project in the Computer Laboratory. The first way is for the phones to prompt the user with options suitable to the action that (s)he is currently trying to accomplish. In most cases there are only a few relevant options that can be offered to the user at any one time, as opposed to a complete instruction book full. For example, if the called party does not answer, an intelligent PABX might well play the caller a voice message saying "please press 0 to try another extension, 1 to leave a recorded message or nothing to keep trying". In response to keystrokes it would obey the caller's wishes, in the case of leaving a voice message using further vocal prompts to explain the procedure to the caller. Such a system of vocal prompts not only avoids a caller needing to remember how to control the phone, but also means that it is the caller who takes the decisions for example about where a call should be forwarded. The caller who knows what is going on will probably be more satisfied with the service. The ISLAND demonstration system for this idea was a voice mail system. To collect or send messages users only needed to listen to instructions and push buttons. The system was completely self-explanatory to new users. For more experienced users instructional messages would only be played until a valid input was detected, making the system fast for use by experts.

Other similar systems have been reported in the literature [e.g. Karhan 86, Schiller 86]. These two examples are used for remote interrogation of a database. In the case of [Schiller 86] the caller would possess a special telephony instrument tailored to the system. Karhan's system is a service available to all subscribers on the telephone network, provided they are equipped with DTMF[8] dialling. In the U.S.A. most subscribers have DTMF capabilities. In the U.K. very few do. Work done by FERMA [FERMA 83] has shown that detection of conventional dialling pulses across the A.C. coupled national phone network is difficult to accomplish with a satisfactory success rate.

Dialling pulses are easy to detect in a local system, but for local use such message and keyed response techniques are rivalled by a better alternative. Advanced PABX facilities can be divided into two categories—those (such as leaving messages for absent people) which should be offered to outside users and others (such as redirecting all one's calls to another extension) which are only appropriate for local users. The message and keyed response approach would suit the first category. For a local user supplied with a workstation as well as a phone, a far preferable way to implement the interface to sophisticated PABX features is to offer menus and other command structures on the workstation screen. The Etherphone project has demonstrated some good examples of this. One is an improved alternative to offering short codes as a quick way to dial frequently used numbers. The workstation display contains a window showing people's names. Selecting a name and clicking the phone menu button causes the named person to be called. The workstation also shows (at least for internal Etherphone calls) the name of the caller, displayed on a 'phone ringing' icon. When the phone is being used to record voice (as described in chapter 8) this moving icon replaces the telephone bell.

The ISLAND demonstration features consisted of commands to control a phone from its associated workstation—to notify GOD that a user had moved (temporarily or permanently) to another phone, with an optional 'disturb only if urgent' flag, and a command to "phone <user

[8] Dual Tone Multi-Frequency, commonly known as 'touch tone'

name>". This demonstration system assigned phone numbers to users rather than to their telephones, a feature which we would dearly like to see in commercial PABXs! As the third part of this workstation demonstration, a call to a phone would be accompanied by a message on the corresponding workstation showing the origin of the call. These facilities demonstrated simple primitives between GOD and the workstation. It would clearly be possible to invoke them automatically from more sophisticated user interface programs—for example the phone command could be invoked from within a personal telephone directory browser.

It might be useful to be able to access for example the 'visit' command using short numerical codes typed at the phone, for use in a common room where there is no workstation. Provision of these facilities will not be addressed here. We believe that they should be regarded as substitutes for the workstation interface only under specific circumstances (such as there being no workstation in a room) and that the workstation interface is the better way for such facilities to be offered in general. (The whole object of this discussion is the replacement of existing cryptic PABX user interfaces.)

9.2.6 Message Paths for Telephone Control

Before the subject of control of telephony facilities from the workstation is left, the underlying structure of control paths will be discussed briefly. It was mentioned in 1.4.4 that the combination of a phone and workstation into one unit causes a major design problem. The phone must be highly available, whereas users require the capability to program (and therefore to crash) their workstations. (Even if the workstation is not user-programmable and runs a protected operating system, it will still fail more often than a phone should: experience with applications on the MAGNET system bears this statement out.) It is for this reason that the phone has been kept physically separate from the workstation. In the demonstration system described above, messages pass in parallel between GOD and phone and between GOD and workstation. Other researchers have also tried this approach. We believe, however, that it would be more elegant for signalling between GOD and phone to pass *through* the workstation.

In either case, GOD will send to phone and workstation complex information (e.g. about the caller), which the phone will discard and the workstation may interpret, e.g. displaying it to the caller. The workstation may also send GOD information in the same format as its accompanying phone would, e.g. to set up a call or a forwarding path. There are some circumstances in which the workstation may wish to intercept information before it reaches the phone. The phone may be in use for input of voice to a document, in which case the workstation should stop the phone bell from sounding and instead flash a message on its display. A user may have set up a call filtering option on the workstation, in which case some calls may be rejected by the workstation without the phone ever being notified.

The control path *through* a workstation is the more elegant for interception of such information. The problem that has to be solved in either approach is how to prevent disruption of the phone service when the workstation crashes. Because the workstation must be given the opportunity to prevent the phone from sounding its bell, in the parallel control approach simple incoming call notifications cannot be sent to the two at the same time—the workstation must be allowed time to reject the call or to suppress ringing. However if the workstation fails, the communication with the workstation must be timed out and the phone notified of the call regardless.

In the case of a parallel path scheme, the timeout must be set to be fairly short (for otherwise a caller may hang up before the phone bell has actually sounded). An alternative approach is to send messages of the form "ring your bell in 1 second unless the workstation tells you not to do so". The workstation will be required to act within a small time if it wishes to affect operation of the phone. In a 'routed through' scheme the implementer would utilise the fact that in ISLAND all communications between GOD and phone are SSPs. The low level SSP protocol mechanism would hold a flag saying whether communications were being routed through a workstation. If so, where an SSP reply did not return within a short time it would be retried both through the workstation and directly.

10. Summary and Conclusion

In this chapter the lessons which can be learned from the work presented in this dissertation are summarised in three sections, corresponding to the three main objectives set out in section 1.2. Before that, however, is a discussion of what has been achieved and how it supports the thesis. The ISLAND architecture was designed to run on the Cambridge Fast Ring but most of the work presented to support that design has been done using other networks. The implications of this are included in that discussion.

After summarising what has been learnt, this chapter concludes by discussing some areas where further work could usefully be done. The topic of extending the multi-media document work presented in chapters 8 and 9 is considered fairly extensively, and then briefer comments on some other topics follow, including PABX facilities and video transport services.

10.1 Status Report

Whilst much emphasis has been laid in this dissertation upon building a coherently integrated system, a single working system cannot yet be demonstrated. The work that has been done falls into three divisions. First there is the work on an integrated editing interface discussed in chapter 8, implemented in the Etherphone environment. The aims of the Etherphone and ISLAND projects are very much the same at levels above the transport and storage of voice. The work in chapter 8 is of equal relevance to the two environments. At lower levels the approaches and emphases of the two projects are very different. The lower-level design of ISLAND is intended to support facilities such as a voice editor, and part of chapter 9 examines how the ISLAND design presented in other parts of the dissertation would be applied to supporting the work demonstrated in the Etherphone environment.

The second component of work is that on the UNIVERSE voice protocol discussed in chapter 2. The lessons learnt from it are presented as the first part of chapter 3. Whilst this work may not seem explicitly to make much of a contribution to the research goals set out in section 1.2, the importance of the UNIVERSE experiments lies in the effect that they had on the ISLAND design. Techniques from the UNIVERSE which were rejected in the ISLAND architecture are of as much importance as those which were incorporated. An example contrasting the two designs, where improvement can be seen in the ISLAND design as a result of UNIVERSE experience, is the execution efficiency of the voice provider compared with the more cumbersome organisation of the UNIVERSE phones.

The final and largest portion of the work presented is the design and implementation of the ISLAND architecture itself. The aim of this dissertation has been to present a complete architecture for Integrated Services and to show that it coherently links a wide range of applications. The penalty for attempting to demonstrate a system whose scope is very wide is that not all of the components of the design have been implemented. The overall description of the design presented in chapters 1 and 4 is fairly assertive in style, but it is hoped that the rest of the work presented backs the assertions.

Chapter 3 outlined properties of the Cambridge Fast Ring which make it suitable as a foundation for the ISLAND system. However no components of the ISLAND system have been demonstrated on the Fast Ring, because of the unexpected delays before working Fast Ring ICs became available. It is appropriate here to consider once again what effect this has on the validity of the work which has been presented.

The voice provider design was optimised for the Fast Ring. Its implementation on the older 10Mbs^{-1} Cambridge Ring by Roger Calnan shows all the properties expected on a Fast Ring, except that

(i) the efficiency of the provider is less on the older Ring due to the smaller slot size.

(ii) the basic block protocol prevents the voice provider handling transmission or reception of multiple voice streams and long blocks simultaneously. This means that the translator, also implemented by Roger Calnan, is limited to handling only recording or playback at one time.

Otherwise the translator has all the properties which we have indicated the translator must have in the ISLAND design. The lower efficiency of the provider is also the reason why only the stream compounding portion of the conference server and not its network interface code has so far been built; it has therefore not been demonstrated handling voice streams originating from across the network.

Even though the voice provider is less efficient than it would be under the Fast Ring, all its important features have been demonstrated—its efficiency and structural advantages and the way it fits into the Ring handling software of a server. Only the different traffic handling capacities within one station reveal a strong contrast between its operation on the two Rings.

The hardware architecture and software structure of servers would be just the same on the Fast Ring as they have been made on the older Ring. The way in which the two Rings are handled at a low level is very similar. Exchanging the old Ring interface for a simple Fast Ring interface, such as the one designed by the author and built by David Tennenhouse, would not effect the issues described in chapter 6 at all. The location of the Ring interface hardware and the need for a level of access to the Ring handling primitives at the level provided by the RIP would be the same.

It is regretted that a working interface according to the design of chapter 7 has not been demonstrated. In particular more use could have been made of the features which are built into the framestore hardware described in chapter 9 if the close coupling intended between high speed network interface and direct access framestore could have been exploited.

What has been produced as a result of all of this work is a kit of parts designed to fit together as one coherent integrated system. Whilst it would have been pleasing to have fitted all the parts together, this would have ruled out working on the different levels of the system in parallel in the way that we did. This parallel approach is believed to have been of benefit for two reasons. First if the work had been pursued in a strictly bottom-up manner, experiments could not have done on enough levels of the system to show the intended extent of integration of the system, given both the years and the man-years available. Secondly it is felt that investigation of the upper levels of the system before the lower levels were complete has helped to improve the design of the lower levels, ensuring that they would be fit for the purposes intended. Examples from this work where one level of design influences layers below it are

(i) the work on the voice editing interface, from which much was learned about the required construction of voice and data filing systems

(ii) the partitioning of servers to make amongst other things the components of such filing systems, which had a considerable influence on the design of voice protocols, the voice provider and server hardware.

10.2 Summary of Lessons from this Research

The conclusions to be drawn from the work presented in this dissertation can usefully be summarised in three main categories, mirroring the three aspects of the objectives set out in section 1.2.

10.2.1 Integrated Protocols

This dissertation has set out an approach to voice and data integration at the network level using digital packets for the transmission of both voice and data. This approach is able to counter the traditional objections to packetised voice. The standard objections are that

i) the delay introduced is unacceptable

ii) the cost of packetisation hardware is prohibitive.

The fact that so many papers still being added to the literature are concerned with the use of complex hardware to reduce the bandwidth required for packet voice links at the cost of introducing heavy delays [e.g. Steele 85, Tada 86] only helps to fuel these objections. The current field trial by AT&T of a packet-based PABX [Burst 86] which uses large packet sizes and introduces tens of milliseconds of delay for local calls is also regrettable in that it helps to promote the argument that packet voice is satisfactory for self-contained systems but will never be suitable for interworking with national phone networks.

Much emphasis has been placed in this dissertation on the design of protocols to produce delays acceptable for interworking with national systems. Such protocols can be realised

a) by use of a packet network which is suited to the transmission of very small packets and in which the queueing delays for packet transmission are very low.

b) because in a local area context there is no need to economise on use of bandwidth. There is therefore no need to compress voice streams or bear the extra delays this would introduce.

A network which provides very much more bandwidth than its clients require is not necessarily an expensive option. On the contrary, it may be cheaper, because the cost of a network is heavily dependent upon the complexity of the client interfaces. A high bandwidth LAN, whose interfaces are much cheaper because they need not try to draw all available bandwidth from the LAN, is attractive. (For example the Orwell protocol [Adams 84] is designed to exploit all of the raw bandwidth of a LAN; interfaces which implement the Orwell scheme are much more complex than those required for the Fast Ring.)

The facts that transmission of 64kbs^{-1} streams on a 50Mbs^{-1} LAN does not require compression of voice streams and requires only a very simple interface and lightweight protocol help to counter the second standard objection to packet voice systems. At the conclusion of chapter 5 the realisation of a mass-produced ISLAND packet phone was discussed. It would use a small number of components, including a semi-custom IC and a simple microprocessor. Such a phone, including its ring interface, need not cost more than a conventional phone *together with its line card in a PABX*, the components which it replaces. Clearly for an ISLAND style phone to be financially viable it needs to be produced in large quantities: the cost of producing a custom IC lies largely in its development. On one hand we have tried to minimise the complexity of the IC required by simplifying the protocol that it is required to implement, but on the other hand there also need to be enough ICs produced to amortise development costs.

Similar arguments apply to conventional phone instruments. The complexity of a standard U.K. type 746 telephone is surprisingly great and its telephone dial is a complex piece of mechanical engineering which could never have been produced at a sensible cost had there not been the guarantee of large volume requirements. Modern phone instruments do not retain the extensive wiring harnesses of the type 746 and are hence less expensive to assemble, but typically these newer phones contain a microprocessor and various other support ICs. Measured against these phones the ideas on the design of a minimal packet telephone set out in chapter 5 do not look extravagant.

The above paragraphs may read like a voice crying in the wilderness, but there are encouraging signs that this voice and others like it are being heeded. Four years ago, most engineers from the PTTs who were introduced to the design of ISLAND would simply dismiss it. The stringent requirement of the 2.5ms delay for a PABX would usually be cited along with the impossibility of producing packetising hardware at a reasonable cost. The recent Oftel report [Oftel 86] which raises the amount of delay allocated in the U.K. to 'private networks' (the PABX no longer being regarded as the discrete unit for purposes of regulation) shows that there are moves to make it possible for new systems, including packet systems, to interwork with conventional networks. Recent conversations with strategic planners from the PTTs, including some from British Telecom, reveal that they are both working themselves and looking to experimental work such as that reported here to set standards of acceptability for packet voice network characteristics. The driving factor for this is that Integrated Services LANs are becoming attractive to replace conventional PABXs, partly due to problems of software costs and decreasing life expectancy of the latter, about which more will be said below.

10.2.2 Design of a Distributed PABX

One use that can be made of an Integrated Services LAN is to build upon it a distributed PABX. The ISLAND design of PABX can be said to be distributed in two respects:

i) the basic function of a star-configured PABX is to switch calls between pairs of telephone lines. A LAN is inherently a distributed switch: a series of ISLAND phones on a Ring provides the same functionality as the combination of telephone instruments and the PABX's line interfaces and switching units in a conventional star PABX. Only control functions are missing in either case.

ii) use has been made of standard ideas from distributed computing to add functionality to the PABX. A distributed computing system (such as [Needham 82]) typically consists of a number of machines on a network, each performing different functions which together constitute the capabilities of a large multi-user operating system. (Examples of such functions are authentication, printing, file storage on disc and tape, developing and running user programs etc.) Separable functions which have been identified in applying these ideas to construction of a PABX are a conference bridge, a means of storing recorded voice (subdivided further into a conventional fileserver and a special voice 'front end' to it), a clock synchronisation server and a control server (to connect calls etc.). The last function is distributed further, not as a means of splitting functionality but to increase the reliability of the system. Some other functions which could also be added as additional servers have been discussed in this dissertation.

The impetus for applying ideas from distributed computing to the design of PABXs was the desirability of building a PABX structure which is extensible and upgradable. Two very disturbing statistics quoted by PABX designers are that

i) 80% of PABX costs now lie in software

ii) the average installed life expectancy of a PABX has fallen from 20 years (in the days of Strowger systems) to less than five.

PABXs currently become obsolete more often because of their limited and fixed functionality than because of a need for greater capacity. The latter is nevertheless of some concern: there is an increasing trend in the U.K. towards 'Centrex' systems (where the customer hires a section of the town exchange rather than a separate PABX) because such systems allow a customer to alter the size of an installation in small increments.

The ISLAND design, based upon a LAN, allows for gradual changes in system capacity, but it is chiefly aimed at reducing the cost of altering and adding to the functions of the PABX. A series of small software modules running in separate servers can be easier to produce than one large system, not least because there is no need to worry about real time constraints in sharing a single machine. It has been stressed that the modules should have not only very well defined single functions but also a universal and simple interface structure, such as a single voice transfer protocol in the case of the phones, conference bridge and voice filing 'front end'. If no knowledge is built into the servers (and phones) about what they are connected to at any time, this ensures that new facilities can be added to the system (either by upgrading a server or by adding a new one) without the need to alter other servers to accommodate the change. Only the call connection (or facility allocation) server(s) need know when upgrades which provide for new facilities are made.

The benefits of this approach are that

(i) it becomes cheaper to produce and upgrade facilities.

(ii) it is possible to provide different sets of facilities for different customers. (To a PTT the idea of being able to upgrade a system by installing an extra server and its software and charge the customer on a per-service basis is attractive.)

(iii) both the set of facilities and the number of lines that a system supports can be upgraded without major changes to or complete replacement of the system, thus helping the problem of obsolescence.

The design of phone set out in this dissertation contains nearly minimal functionality. This is intended partly to reduce the hardware costs of such a ubiquitous component and partly to prevent changes in system functionality requiring replacement or alteration of the phones. Upgrades to the system should affect only the servers: the ISLAND design tries to minimise their hardware costs by creating one set of hardware powerful and general enough to support not only most services currently envisaged but also new and upgraded services as yet unplanned, which may call for more powerful resources.

A new series of servers based on the Cambridge Ring has been designed and built to support the various PABX functions. New hardware and software structures were primarily required in order to support a combination of data and voice traffic within the same machine. In particular the servers have a very different structure of network interface software than needed in conventional data handling systems. Parallel paths are provided in the machine by which frequent small voice packets in a very lightweight protocol can be handled whilst at the same time reliable data protocols are also being handled. This has ramifications for the choice of operating system, particularly that it must be able to handle the frequent arrival of small voice packets. The need, e.g. in a conference bridge, to process voice in small quantities and within a critical time also has implications for process scheduling in the operating system.

10.2.3 A Flexible Base for User-Level Services

Although a new design of PABX is enough justification for developing the protocols and distributed system discussed above, another major motivation for that work was the possibility of supporting new user level services which take advantage of different media available simultaneously within one framework and combined in one user interface. Most of the commercial work on integrated services (i.e. the ISDN) has been concentrated at the network level, with regrettably little attention devoted to user level facilities. The availability of high bandwidth links has made new video-based services possible, but these have generally been considered in isolation.

This dissertation has discussed extending the facilities provided by a workstation to include

i) control of the telephone from the workstation. As the range of facilities in a PABX becomes increasingly wide, the poverty of a user interface consisting of a dozen buttons, some response tones and possibly a few lights becomes more and more noticeable. Control of telephony functions from a workstation allows a much more convenient and powerful interface to the PABX, replacing extensive users' manuals which need to be read and memorised with a hierarchical, menu-driven interface and providing more feedback (e.g. information on the source of an incoming call or on the progress of a call forwarding sequence).

ii) a spectrum of services based upon image transport, ranging from single image transfers for document delivery or interrogation of a remote image bank to full frame rate video conferencing, and also facilities such as distributed blackboards.

iii) multi-media document handling. Documents containing text, graphics and simple scanned images can be composed on many commercially available workstations. The addition of voice, more complex images and video requires work not only on the user interface but also on the servers and techniques needed to handle such documents.[1]

[1] More will be said about multi-media documents at the end of this chapter, in discussion on further work.

A study has been presented in this dissertation on a distributed architecture for accessing, editing and storing multi-media documents. Voice and images require large amounts of storage space: this leads to a desire not to hold multiple copies of items in these media. This in turn leads to the decomposition of multi-media documents into complex structures of related files. Use of databases has been discussed as a means to hold the relationships between the files which constitute such documents, enabling amongst other things garbage collection of inaccessible files.

Access to multi-media documents also requires special attention: the very different natures of voice and data transport mechanisms calls for either separate fileservers or separate access protocols and routes to a common fileserver. The use of separate fileservers has been avoided as this would make the task of accessing 'one document' in its entirety more difficult than it need be. Access to a common fileserver is obtained instead via two different 'front ends', one for interactive access to real time voice streams and the other for interactive access to the data components and for assembling documents for transport as self-contained and complete entities. The two front ends are thus both used for editing documents. A multi-media editor can beneficially be partitioned into components for filing, for manipulating and updating the files which constitute the document and for handling i/o. This is more sophisticated than but logically equivalent to the manner in which editing is performed using the Cambridge Processor Bank and Terminal Concentrator. The phone is the i/o device for voice. For visual media a general purpose display is required. That display must be capable of fast manipulation of fonts and images.

The workstation must also be capable of handling general image transfer services. Special hardware for the workstation display has been built to meet this requirement, consisting of a flexible framestore system. The hardware is oriented to capturing, displaying and transferring across the LAN all types of image—real scanned images, as well as fonts and other graphics. The framestore-based system is intended to act only as a workstation *terminal*: for editing it must communicate with the processor which is handling the structure of a document. It must also interact directly with the fileserver (or its relevant 'front end') to access images and fonts.

Provision of video transfer facilities and access to data filing services from within a single display has led to problems of network interfacing. Network interfaces have traditionally been either circuit switched, simple and fast (e.g. a 'Megastream' link to a video CODEC) or packet switched, intelligent and slow (as in a typical distributed computing system). The combination of attributes required of the interface to support mixed traffic in and out of an integrated display is that it should be packet switched, intelligent and fast. A high speed intelligent interface design has been presented: it seems that for satisfactory throughput protocols must be designed specifically for high speed implementation and at least to some extent the lowest level of hardware must be tailored to them.

Work has been carried out on interfaces to present integrated facilities to the user, in particular an interface for incorporation of voice manipulation facilities into a structured text editor. More emphasis has been placed however on development of a generalised and flexible architecture to support other facilities. The experience gained from designing the voice editor commends this approach. Products such as [Centerpoint 86] satisfy particular needs for office facilities, but they lack flexibility and extensibility. They are subject to the same problem, i.e. of distressingly short installed life expectancy, as is a conventional PABX. It has been stated that the remedy for this in the case of the PABX is to design a framework which makes the introduction of new facilities easy.

The Cedar environment, in which the voice editor was implemented, illustrates application of this philosophy to mixed media user level services well. Cedar was designed with generality as a major goal. It was aimed at supporting a variety of different applications using common software layers, these applications including both those being designed at the time and where possible those then unanticipated. Determined pursuit of this philosophy by the designers of Cedar made integration of voice into Tioga, the standard Cedar editor, a remarkably straightforward task. The structure of Tioga consists of a series of linked layers. At each link it is possible to add parallel layers to perform slightly different functions or to handle different media or applications. The work presented in this dissertation has tried to follow the same approach in the design of basic software, for example that to make the framestore into a multi-media windowed display. It seems clear that if this approach is followed throughout the design of an integrated infrastructure then a large part of the work of implementing new applications in the future will already have been done and these applications will fit easily into the existing framework. The alternative, of designing each application in isolation, is as undesirable as throwing away a PABX every 5 years.

10.3 Further Work

There is clearly much scope for further work in implementing components whose roles within the ISLAND architecture have been described in this dissertation but which have not yet been fully built. However this section is concerned with future directions for work beyond that already outlined above.

10.3.1 Future Directions in Document Architectures

This subject will be discussed in more detail than the other areas suggested for future work. It includes several different topics of interest.

10.3.1.1 Incorporation of Speech Recognition

It is instructive to examine how conventional dictation machines are used. Executives use the dictation machine as an input device because they can speak faster than they can write or type. A dictation is given to a secretary to type up and no attempt is made to play the whole thing back before then, simply because one can read faster than one can listen to voice. Dictations on conventional machines often include such postfixed editing directives such as "in the first paragraph about artichokes, please change 'would like you to' to 'urgently require you to'", and so forth. The rather more manipulable approach to dictation set out in chapter 8 is aimed at encouraging direct editing of the voice instead of such postfixed amendments.

For dictation applications, voice editing tools such as those described in chapter 8 may eventually become obsolete. As voice recognition systems become more sophisticated, and in particular when they become affordable and generally applicable enough for incorporation into the personal workstation environment, it can be expected that voice input, translated instantaneously into text for fast reading, will become normal. Multi-media editors can then become integrated to the extent that alterations to textual portions of a document may be input vocally.

Accurate translation of continuous speech into text is unlikely to be possible for some time. However systems that instantly translate voice into moderately accurate text that the user can then correct should be available much sooner.[2] A secretary using such a system to transcribe other people's dictations would be able to work much faster by listening to the original voice and correcting a rough automatic translation than by typing the whole thing from scratch. Eventually it will become possible for the secretary to re-enter corrections vocally, giving the voice recogniser 'another chance' at a word incorrectly transcribed.

It must not be assumed that all applications to which voice annotation is currently suited would benefit from automatic translation to text. Inflection, emphasis, and the personal touch of a familiar voice can represent much of the value of a spoken annotation.

10.3.1.2 Extension of Document Architectures

Whilst incorporation of accurate speech recognition into a multi-media document system is still far from being a reality, the idea helps highlight the need for a more generalised document structure and editing interface than has been set out in this dissertation. There are several ways in which document architectures could usefully be extended beyond the work which has been done so far. The scope of the Tioga interface described above was limited to vocal annotation of documents and simple textual annotation of voice, with no further nesting of structure. There are three obvious directions for extension of this scope. First there is no need to restrict the number of levels to which an annotation structure is limited. A unrestricted tree structure of annotations is a possibility. Secondly there is no need to restrict those entities which can be annotated to simple textual characters. On one hand they might be single objects in a graphics window and on the other compound objects could be the target of annotations. Finally, there is no reason to restrict annotation to vocal annotation.

Pursuing the last idea first, the Tioga editor allowed voice to be 'stuck onto' portions of text and then 'popped up' as a visual entity for further manipulation. In a similar manner portions of text could be affixed to other portions of text, using an icon to indicate the presence of an annotation which could be 'popped up' for viewing as required. Simple uses would be to allow the equivalent of 'scribbling in the margin'. Comments are often affixed to printed documents using either a paper-clip or a sheet from the currently fashionable 'post-it' pads. These could be replaced in an electronic document by these annotations. One could pursue a tree-structured approach and allow a document to be regarded as a hierarchy of entities, each entity being a visual or audible component with further entities potentially attached as annotations. Note that the ODA [Horak 85] describes documents as tree-structured, but this is in the very different sense of their organisation and layout upon a flat visual page.

[2] An interesting trend is that the newest continuous speech recognisers, such as [SSI 86], while opening up exciting new possibilities for this application, upon getting a word wrong get it totally wrong. Unlike discrete recognisers, which match templates and therefore tend to produce as errors words that sound similar to the one spoken, these systems are based on complex rules of sentence structure etc. and generally produce as errors words whose connection with the correct one is not at all obvious.

Another, more complex possibility would be a directed cyclic graph structure. The 'whiteboard' graphical database system [Donahue 86] uses graphs of documents for applications such as help systems. A similar structure could be incorporated as part of a document architecture. This crosses the boundary from single documents to document management systems, with associated problems for garbage collection of document components held on a fileserver. If documents are annotated in a tree-structured manner it is still clear what constitutes a single document. If a more complex graph of links is used then the links will probably be used to connect related information held within the filing system: the objects which are linked together in the graph will have to be regarded as separate documents, unless different and distinct types of link are employed to represent reference and ownership. The FRESS system [Yankelovich 85] distinguishes 'tags' (tree-structured links) and 'jumps' (database entries) but published reports on the system do not address matters of ownership and administration of the contents of the filing system.

In addition to vocal and textual annotations, documents could also include computer animations and sequences of video frames. Timing information could be added to control the presentation of documents. A possible application is a document able to 'give an audio-visual presentation of itself', the timing and sequencing information being used to scroll specified areas into view and initiate sound playback or animation sequences. Several variations on the ideas above have been demonstrated by other researchers: as well as whiteboards, there are the Notecard [Halasz 87], Web [Knuth 84b], Neptune [Delisle 86] and scripted document [Zellweger 87] systems, all variously in part conforming to the definition of a 'hypertext' [Carmody 69]—a semantically directed document structure.

Most of the architectural extensions suggested here could be accommodated within a framework of existing uniform user interfaces—the user would not have to learn large numbers of new concepts. It should be expected, however, that new media or structures will occasionally introduce the need for specialised interface concepts and operations, just like those discovered in building the voice editing interface. For instance, the behaviour of a voice annotation still needs to be slightly different from that of other 'pop-up' annotations, because there is a need for both a simple playback capability and the more heavyweight operation of opening of a voice viewer to edit the voice. There is no equivalent to these two parallel requirements in the case of visual media.

10.3.1.3 Management of Images

Section 9.1 dealt with the problem that it is impractical to include digitised voice directly in annotated documents because of its bulk. Only references to the voice were included. Incorporation of scanned images into documents is a desirable extension of the capabilities discussed in this dissertation. Images are another medium that involves a substantial amount of data, and that will be manipulated in large units. This suggests that images should also be represented in documents by embedded references and edited by manipulating these references, to reduce the time and storage required to copy documents. The database facilities needed for managing voice references could be extended to handle images, possibly including animation sequences and video frame sequences.

10.3.1.4 Attachment Regions for Annotations

The issue of attaching a voice annotation to a general area of a document was deliberately avoided in chapter 8. Annotations could only be attached to individual characters. It might be desirable to be able to attach voice or more general annotations to regions of a document—areas of scanned images, graphics, voice or combinations thereof. This raises problems of clarity of the user interface. The reason why Tioga permits attachments only to individual characters is that we were unable to see a way for the user to interpret the meanings of more generalised attachments. This area may merit further work: we believe that any new scheme should continue to offer to the user a clear and unambiguous model for all the functions in the interface. For voice moreover it is important to offer an interface which preserves the balance of generality and extensibility with speed and convenience: the latter are vital to those applications for which voice is most suited.

10.3.2 Control of PABX Facilities

Very little mention has been made in this dissertation of how the distributed PABX should be controlled. Although both the need for GOD and its interaction with for example phones have been discussed, the way in which GOD is realised has not been considered, since this is the separate research work of Roy Want. Neither the progress of that work nor how it might be followed up are relevant here, but it is important to compare the software complexity of GOD with that of the voice servers.

Partitioning the PABX into a series of servers each handling a single function facilitates upgrading, altering and extending the range of features offered. The use of standard interfaces between those servers also facilitates offering varying configurations of PABX to different customers. However this does not address another, major, aspect of the problem. Manufacturers of telephony switching equipment typically quote lead times of a year or so to introduce a new 'facility' into their systems (where a facility can be for example automatic call forwarding or a 'wake-up call' mechanism). New facilities are very expensive to introduce, simply due to the time taken for engineers to determine the way they will interact with all the other facilities already in the system. It is trivial to incorporate a single facility into an exchange, but ten facilities will be vastly more difficult, due to the effort needed to prevent them from interacting in subtle and unpleasant ways. It is very easy to invent scenarios in which two badly designed facilities would combine to produce an unwanted effect.

It is not appropriate to try and express possible solutions to this problem in brief notes. The problem is raised so that the reader should not think that the techniques set out in this dissertation are claimed to solve all the problems of building PABXs.

10.3.3 Image and Combined Media Transmission

A variety of possible image-based services has been mentioned in this dissertation, along with features in the framestore-based workstation display designed to facilitate the provision of these services, but little work has been done on image transmission itself. Image transmission currently occurs mostly in a wide area context, where bandwidth is expensive. For this reason a very large amount of work has been done on compression of video. Significant further improvements in the compression of images are likely to come only from 'semantic' interpretation of their content. A simple example of what this means is the identification of all independently moving components of a picture, so that for example moving lips within a moving head can be treated separately.

Video transmission across LANs is not currently regarded as very useful, but the advent of high bandwidth LANs which allow high speed image transfers at very little cost may stimulate LAN-based video services. Potential applications are to be found in surveillance and in robotics and process control. A LAN may be used to gather video information for a centralised plant controller. Instructions to robots in the plant may also be sent back over the LAN. Greater ease in transporting still and sequential images within the local environment may also stimulate new applications for them within document architectures.

There have been various reports of systems allowing shared access to media across a network, for example to enable collective editing of a document. There is scope for development of more systems of this type. Such systems may need simultaneous communications channels for several media. For example, for editing a document, audio, slow or fast scan video, a distributed blackboard and shared text displays may all be useful. Although the PTTs have strong views, there appears to be very little hard experimental evidence as to how closely the various channels need to be synchronised for such applications. Problems are most likely to be encountered on wide area links, where the difference between delays on various channels may be large (e.g. if some channels are carried by satellite and some terrestrially).

10.4 Conclusion

The lessons drawn from this work have been set out in the middle part of this chapter. The final matter to be reviewed is the area of applicability of those lessons. They are principally relevant in a local area context. At present wide area networks do not merit the same approach to integration at the network level, since the vast majority of traffic they carry is voice. (Most of the rest is electronic mail.)

This may not remain the case forever. The term 'metropolitan area network' (MAN) is becoming popular and seems often to be used to denote extension of LAN-style practices at least to town-wide systems. Use of an ISLAND-like system can be envisaged not only to replace large PABXs used by single customers but also to provide for shopping centres, where each shop represents a customer using only a handful of equipment connections. The combined needs of such customers for new facilities might be better served by installation of a local system than by running connections from the town exchange to each piece of equipment. The fact that cables from the local exchange to the customer account for 80% of a PTT's investment has often been held to provide insurmountable inertia against going further than this and rewiring a whole town as a MAN. However the spread of cable television installations suggests that if investors see good revenue potential, rewiring a town can suddenly become remarkably feasible. The announcement in Japan of proposals for carrying local voice and data traffic on cable television networks suggests that ISLAND-like systems could achieve a wide applicability.

References

Adams 84:

Adams and Falconer: "ORWELL: A Protocol For Carrying Integrated Services on a Digital Communications Ring"; Electronics Letters, November 1984

Allen 83:

Allen: "Composition and Editing of Spoken Letters"; International Journal of Man-Machine Studies, Vol 19 No. 2, August 1983

Benito 84:

Benito, Celandroni and Ferro: "FIFO Ordered Demand Assignment"; C.N.U.C.E., Pisa, Italy, 1984

Brady 68:

Brady: "A Statistical Analysis of On-Off Patterns in 16 Conversations"; Bell System Technical Journal, Vol 47, January 1968

BT 86:

Figures from British Telecom internal studies

Bullington 59:

Bullington and Fraser: "Engineering Aspects of TASI"; Bell Systems Technical Journal, March 1959

Burst 86:

The AT&T prototype packet switched PABX, 'Burst', which has been on field trial providing telephone services for a series of AT&T company offices in San Francisco, California, since June 1986

Bux 83:

Bux et al.: "Architecture and Design of a Reliable Token-Ring Network"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 5, November 1983

Campbell-Grant 86:

Campbell-Grant: "Piloting the Office Document Architecture"; Proc. IEEE ICC Conference, Toronto 1986

Carmody 69:

Carmody et al.: "A Hypertext Editing System for the /360"; Pertinent Concepts in Computer Graphics, Ed. Faiman and Nievergelt, University of Illinois Press, 1969

CCIR 76:

CCIR Document 4/75-E: "Effects of Bit Errors on Transmission of Speech Through PCM Systems"; CCIR, U.K., February 17, 1976

CCITT 74:

CCITT Planning of Digital Systems: Special Study Group D; Contribution 103, June 1974

CCITT 80:

CCITT recommendation X.25: "Interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for terminals operating in the packet mode on public data networks"; CCITT Yellow Book, Vol VIII, Fasicle VIII.2, 1980

Centerpoint 86:

The Centerpoint system, produced by Santa Barbara Laboratories Inc. of California, available since 1986

All comments about and descriptions of Centerpoint are based on observations by the author of the system in use.

Chapanis 72:

Chapanis et al.: "The Effects of Four Communication Modes on the Behavior of Teams During Cooperative Problem-Solving"; Journal of the Human Factors Society of America, Vol 14 No. 6, December 1972

Cheriton 86:

David Cheriton, quoted talking on "VMTP" at the International Workshop on Protocols for Distributed Systems, University College London, September 1986

Chiariglione 82:

Chiariglione, Nicol and Schaefer: "The Development of the European Videoteleconference CODEC"; Proc. IEEE Globecom Conference, Miami, November 1982

Clark 85a:

Clark: "The Structuring of Systems Using Upcalls"; Proc. ACM Symposium on Operating Systems Principles, Washington D.C., 1985

Clark 85b:

Clark: "NETBLT: A Bulk Data Transfer Protocol"; ARPA Network Working Group RFC Note No. 969, December 1985

The BLAST protocol corresponds to the RESEND strategy of NETBLT

Clark 86:

David Clark, quoted speaking on "Why Network Protocols Don't Go Fast" at the NASA AMES Research Center, California, March 1986

Craft 85:

Craft: "Resource Management in a Distributed Computer System"; PhD Thesis, University of Cambridge Computer Laboratory, March 1985

Dallas 80:

Dallas: "A Cambridge Ring Local Area Network Realisation of a Transport Service"; Proc IFIP WG6.4—Workshop on Local Area Networks, Zurich 1980

Delisle 86:

Delisle and Schwartz: "Neptune: a hypertext system for CAD"; Proc. ACM SIGMOD 86 Conference, Washington DC., May 1986

DeTreville 83:

DeTreville and Sincoskie: "A Distributed Experimental Communications System"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 5, 1983

Dion 80:

Dion: "The Cambridge File Server"; SIGOPS Review, New York, October 1980

Donahue 86:

Donahue and Widom: "Whiteboards: a graphical database tool"; ACM Trans. Office Information Systems, Vol 14 No. 1, January 1986

Engelbart 82:

Engelbart: "Towards High-Performance Knowledge Workers"; Office Automation Conference Digest, AFIPS Press, Arlington, Virginia, April 1982

Falconer 85:

Falconer, Adams and Walley: "A simulation study of the Cambridge Ring with voice traffic"; British Telecom Technical Journal, Vol 3 No. 2, April 1985

FERMA 83:

The telephone pulse detection system developed by Francaise d'Electronique, Recherche et Mathematiques, Paris, described in French patent no. 83.20.14, 1983

Forgie 75:

Forgie: "Speech Transmission in Packet Store and Forward Networks"; Proc. NCC, 1975

Forgie 80:

Forgie: "Voice Conferencing in Packet Networks"; Proc. IEEE ICC Conference, Seattle, 1980

Forsdick 84:

Forsdick et al.: "Initial Experience with Multimedia Documents in Diamond," Proc. IFIP 6.5 Working Conference, May 1984

Fry 86:

Fry: "Integrated Voice and Data in the Automated Office"; Technical Report 86.2, School of Computing Sciences, New South Wales Institute of Technology, 1986

Gan 86:

Gan and Davidson: "Speech Compression for Storage and Transmission Using Silence Detection"; Proc. IEEE ICC Conference, Toronto, 86

the method described uses dynamically altering thresholds, a third parameter—zero crossing rate—to avoid suppressing some of the weak fricatives and a fourth which specifies the minimum duration of an energy burst which can be classified as speech

Garcia-Luna 84:

Garcia-Luna, Poggio and Elliot: "Research into Multimedia Message System Architecture"; Report on SRI Project 5363, February 1984.

Garnett 83:

Garnett: "Intelligent Network Interfaces"; PhD Thesis, University of Cambridge Computer Laboratory, May 1983

Gopalakrishnan 84:

Gopalakrishnan and Patnaik: "Integrating Voice and Data on SALAN, an experimental LAN"; University of Bangalore, 1984

Gould 82:

Gould and Boies: "Speech Filing—an Office System for Principals"; IBM Research Report RC-9769, December 1982

Gould 83:

Gould and Boies: "Human Factors Challenges in Creating a Principal Support Office System—the Speech Filing System Approach"; ACM Trans. Office Information Systems, Vol 14 No. 1, Oct 1983

Gray 78:

Gray: "Notes on Data Base Operating Systems", in Bayer, Graham and Seegmueller (editors): "Operating Systems: An Advanced Course"; Springer-Verlag, 1978

Gruber 83a:

Gruber and Strawczynski: "Judging Speech in Dynamically Managed Voice Systems"; Telesis 1983 Two (Bell Northern Research), pp 30-34, 1983

Gruber 83b:

Gruber and Le: "Performance Requirements for Integrated Voice/Data Networks"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 6, December 1983

Hagirahim 86:

Hagirahim and Hawkins: "Performance Evaluations of a Slotted-Frame, Packet-Switched Integrated Voice and Data Network"; IEE Colloquium on Packet Switching of Digital Speech and Data in Radio Systems, October 1986

Halasz 87:

Halasz, Moran and Trigg: "Notecards in a Nutshell"; submitted for ACM SIGCHI+GI, April 1987

Hills 79:

Hills and Evans: "Transmission Systems" (2nd Edition); Allen and Unwin, 1979

Hopper 86:

Hopper and Needham: "The Cambridge Fast Ring Networking System"; Technical Report No. 90, University of Cambridge Computer Laboratory, 1986

Horak 85:

Horak: "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization"; IEEE Computer, October 1985

Jankowski 76:

Jankowski: "A New Digital Voice-Activated Switch"; COMSAT Technical Review, Vol 6, Spring 1976

JNT 82:

Joint Network Team: "Cambridge Ring 82 protocol specifications"; U.K. Computer Board for Universities and Research Councils, 1982

Joyce 83:

Joyce et al.: "Buffering Speech Signals in a TASI System"; (Northern Telecom, Ottowa) U.S. Patent No. 4,382,296 issued May 1983

Karhan 86:

Karhan et al.: "Text-to-Speech Synthesis for Pronouncing Names and Addresses in a Telecommunications Service: Designing the User Interface"; Proc. American Voice I/O Society Conference, Alexandria, Virginia, September 1986

Knuth 84a:

Knuth: "The Complexity of Songs"; Comm. ACM, Vol 27 No. 4, April 1984

Knuth 84b:

Knuth: "Literate Programming"; Computer Journal, Vol 27 No. 2, May 1984

Lamel 81:

Lamel et al.: "An Improved Endpoint Detector for Isolated Word Recognition"; IEEE Trans. Acoustics, Speech and Signal Processing, Vol 29, June 1981

As well as two energy thresholds, this algorithm also uses hysteresis and energy gradient around the thresholds. See also the bibliography in [Gan 86].

Lampson 81:

Lampson et al.: "The Dorado: A High Performance Personal Computer: Three Papers"; Xerox PARC Technical Report CSL-81-1, January 1981

Lazar 85:

Lazar et al.: "MAGNET: Columbia's Integrated Network Testbed"; IEEE Journal on Selected Areas in Communications, Vol 3 No.6, November 1985

Leblang 85:

Leblang, Chase and Mclean: "The DOMAIN Software Engineering Environment for Large Scale Software Development Efforts"; IEEE 1st International Conference on Computer Workstations, San Jose CA, November 1985

Leslie 81:

Leslie, Bannerjee and Love: "Organisation of Voice Communications on the Cambridge Ring"; Local Networks and Distributed Office Systems Conference, Online Publications Ltd., London, May 1981

Limb 82:

Limb and Flores: "Description of Fasnet—A Unidirectional Local Area Communications Network"; Bell System Technical Journal 61 No. 7, September 1982

Limb 83:

Limb and Flamm: "A Distributed Local Area Network Packet Protocol for Combined Voice and Data Transmission"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 5, November 1983

Logica 81:

"Interface Unit Manual: Multibus Intelligent Interface Unit - VMI-1"; Logica VTS, 1981

McCracken 84:

McCracken and Akscyn: "Experience with the ZOG Human-Computer Interface System," International Journal of Man-Machine Studies, Vol 21 No. 4, October 1984

Markel 76:

Markel and Gray: "Linear Prediction in Speech"; Springer-Verlag, 1976

Maxemchuk 80a:

Maxemchuk: "An Experimental Speech Storage and Editing Facility," Bell System Technical Journal, Vol 59 No. 8, October 1980

Maxemchuk 80b:

Maxemchuk and Wilder: "Experiments in Merging Text and Stored Speech"; Proc. IEEE National Telecommunications Conference, Houston, Texas, 1980

Maxemchuk 82:

Maxemchuk: "A Variation on CSMA/CD That Yields Moveable TDM Slots in Integrated Voice/Data Local Networks"; Bell System Technical Journal, Vol 61, September 1982

Metcalfe 76:

Metcalfe and Boggs: "Ethernet: Distributed Packet Switching for Local computer Networks"; Comm. ACM, Vol 19 No. 7, July 1976

Mirrer 82:

Mirrer: "An interactive, graphical, touch orientated speech editor," Master's Dissertation, Laboratory for Computer Science, MIT, 1982.

Mockapetris 77:

Mockapetris, Lyle and Farber: "On the Design of Local Network Interfaces"; IFIP Information Processing '77, Toronto, August 1977

Montgomery 83:

Montgomery: "Techniques for Packet Voice Synchronization", IEEE Journal on Selected Areas in Communications, Vol 1 No. 6, December 1983

Naylor 82:

Naylor and Kleinrock, "Stream Traffic Communication in Packet Switched Networks: Destination Buffering Consideration"; IEEE Trans. Communications, December 1982

Needham 82:

Needham and Herbert: "The Cambridge Distributed Computing System"; Addison-Wesley Publishers, 1982

Newman 86:

Newman et al.: "QPSX, A Queued Packet and Synchronous Exchange as a Metropolitan Area Network"; Department of Electrical and Electronic Engineering, University of Western Australia, 1986

Nicholson 83:

Nicholson: "Integrating Voice in the Office World"; BYTE Magazine, pp177–184, December 1983.

Nicholson 85:

Nicholson: "Usage Patterns in an Integrated Voice and Data Communications System"; ACM Trans. Office Information Systems, Vol 3 No. 3, July 1985

Oftel 86:

"Code of Practice for the Design of Private Telecommunications Networks": U.K. Office of Telecommunications, 1986

this report has only advisory status

Poggio 85:

Poggio et al.: "CCWS: A Computer-Based, Multimedia Information System"; IEEE Computer, October 1985

Reynolds 85:

Reynolds et al.: "The DARPA Experimental Multimedia Mail System"; IEEE Computer, October 1985

Richards 79:

Richards et al.: "TRIPOS—A Portable Operating System for Minicomputers"; Software—Practice and Experience, June 1979

Rudin 78:

Rudin: "Studies on the Integration of Circuit and Packet Switching"; Proc. IEEE International Conference on Communications, Toronto, June 1978

Ruiz 85:

Ruiz: "Voice and Telephony Applications for the Office Workstation," Proc. 1st International Conference on Computer Workstations, San Jose, CA, November 1985.

Samet 84:

Samet: "The Quadtree and Related Hierarchical Data Structures"; ACM Computing Surveys, Vol 16 No. 2, June 1984

Sarin 85:

Sarin and Greif: "Computer-Based Real-Time Conferencing Systems"; IEEE Computer, October 1985

Satya 86:

M. Satyanarayanan, quoted speaking on Carnegie Mellon University's 'Andrew' project, at Xerox Palo Alto Research Center, April 1986

Schiller 86:

Schiller: "Field Craft Technician Communication with a Host Computer using Synthesized Voice"; Proc. American Voice I/O Society Conference, Alexandria, Virginia, September 1986

Schmandt 81:

Schmandt: "The Intelligent Ear: A Graphical Interface to Digital Audio," Proc. IEEE Conference on Cybernetics and Society, October 1981.

Schmandt 84:

Schmandt and Arons: "Phone Slave: A Graphical Telecommunications Interface"; Proc. International Symposium of the Society for Information Display, 1984

Shiu 84:

The figures quoted are the observations of Stephen Shiu of Rank Xerox, Uxbridge, U.K., made in 1984

These figures are not the result of a formal study, but are derived from experience in supporting about 100 middle management users within Xerox. Given more sophisticated workstations offering more facilities, a manager will spend more time using the workstation but its cost will be correspondingly higher and the same conclusion will still hold.

Sincoskie 83:

Results of a study by Sincoskie and others at Bell Communications Research, Murray Hill, NJ, on LAN and conventional PABX traffic in a business premises, 1983

Smith 82:

Smith et al.: "Designing the Star User Interface"; Byte Magazine, pp242-282, April 1982

SSI 86:

The Phonetic Engine™, a product of Speech Systems Inc. of Tarzana, California, launched in 1986

Steele 85:

Steele and Fortune: "An Adaptive Packetization Strategy for A-law PCM Speech"; Proc. IEEE ICC Conference, Chicago, 1985

Swart 86:

Swart: "ISDN Business Services"; Proc. IEEE ICC Conference, 1986

Swindon 83:

Cambridge Ring Station Chip—Specification; Swindon Silicon Systems Ltd, Swindon, U.K., 1983

Swinehart 83:

Swinehart, Stewart and Ornstein: "Adding Voice to an Office Computer Network"; Proc. IEEE GlobeCom Conference 1983

also available as Xerox PARC Technical Report CSL-83-8

Swinehart 86:

Swinehart, Zellweger, Beach and Hagmann: "A Structural View of the Cedar Programming Environment"; ACM Trans. Programming Languages and Systems, Vol 8 No. 4, October 1986

also available as Xerox PARC Technical Report CSL-86-1

Tada 86:

Tada, Taka and Honma: "16kbs^{-1} APC-AB Coder using a Single Chip Digital Signal Processor"; Proc. IEEE ICC Conference, Toronto, June 1986

Takeuchi 86:

Takeuchi et al.: "Synchronous Composite Packet Switching, a Switching Architecture for Broadband ISDN"; submitted for publication in IEEE Journal on Selected Areas in Communications

Temple 84:

Temple: "The Design of a Ring Communication Network"; University of Cambridge Computer Laboratory Technical Report No. 52, 1984

Terry 87:

Terry and Swinehart: "Managing Stored Voice in the Etherphone System"; in preparation (Xerox Palo Alto Research Center, Palo Alto, California)

Tevanian 86:

Tevanian: "MACH: A Basis for Future UNIX Development"; European UNIX Users' Group, Autumn 1986 Conference

MACH is not the only system with the aims set out in chapter 6: very similar ideas have been pursued at DEC Systems Research Center in California, for example

Thomas 84:

Thomas, Coudreuse and Servel: "Asynchronous Time-Division Techniques: An Experimental Packet Network Integrating Videocommunication"; International Switching Symposium, Florence, May 1984

Thomas 85:

Thomas et al.: "Diamond: A Multimedia Message System built upon a Distributed Architecture"; IEEE Computer, October 1985.

Tichy 83:

Tichy: "Design, Implementation and Evaluation of a Reference Control System"; IEEE 6th International Conference on Software Engineering, February 1983

Tobagi 83:

Tobagi, Borgonovo and Fratta: "Expressnet: A High-Performance Integrated-Services Local Area Network"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 5, November 1983

Ulug 81:

Ulug, White and Adams: "Bidirectional Token Flow System"; Proc. 7th Data Communications Symposium, Mexico City, October 1981

VME 82:

VME Bus Specification Manual, Revision B; VME Manufacturers' Group, 1982 (available from Motorola Corporation and others)

Wang 82:

Wang Laboratories, Inc: "Audio Workstation Author Guide"; 1st Edition, Part Number 700-6989, 1982.

Want 84:

Want: "PhD Thesis Proposal"; University of Cambridge Computer Laboratory (internal note), September 1984

Waters 84:

Waters et al.: "The use of broadcast techniques on the Universe Network"; ACM Symposium on Communications Architectures and Protocols, June 1984

Weinstein 78:

Weinstein: "Fractional Speech Loss and Talker Activity Model for TASI and Packet-switched Speech"; IEEE Trans. Communications, Vol 26 No.8, 1978

Weinstein 79:

Weinstein et al.: "The Tradeoff between Delay and TASI Advantage in a Packetised Speech Multiplexor"; IEEE Trans. Communications, Vol 27 No.11, 1979

Weinstein 83:

Weinstein and Forgie: "Experience with speech communication in packet networks"; IEEE Journal on Selected Areas in Communications, Vol 1 No. 6, December 1983

Westall 85:

Westall and Lee: "Simultaneous Transmission of 7kHz Bandwidth Speech and Data in 64kbs^{-1}"; British Telecom Technical Journal, Vol 3 No. 2, April 1985

Wilbur 83:

Wilbur and Hall: "UNIVERSE Boot Server Protocol Implementation Specification"; INDRA Note No. 1295, University College London, January 1983

Wilder 82:

Wilder and Maxemchuk: "Virtual Editing II: the User Interface," Proceedings of the SIGOA Conference on Office Automation Systems, Philadelphia 1982.

Wilkes 79:

Wilkes and Wheeler: "The Cambridge Digital Communication Ring"; Local Area Communications Network Symposium, sponsored by Mitre Corporation, Boston, MA, May 1979

Wirth 82:

Wirth: "Programming in Modula-2"; Springer-Verlag, 1982

Xerox 85:

Mesa Processor Principles of Operation; Version 4.0, Office Systems Division, Xerox, May 1985

Yankelovich 85:

Yankelovich, Meyrowitz and van Dam: "Reading and Writing the Electronic Book"; IEEE Computer, October 1985

Yudkin 83:

Yudkin: "Resource Management in a Distributed System"; PhD Thesis, University of Cambridge Computer Laboratory, September 1983

Zellweger 87:

Zellweger: "Scripted Documents"; in preparation (Xerox Palo Alto Research Center, Palo Alto, California)

Appendix

Published Papers Containing Material in Common with this Dissertation

Material presented in this dissertation also appears in the following publications:

Chapter 2 and part of chapter 3:

Adams and Ades: "Project UNIVERSE Report No. 11: Voice"; Science and Engineering Research Council, U.K., November 1984

Adams and Ades: "Voice Experiments in the UNIVERSE Network"; Proc. IEEE ICC 85 Conference, Chicago, 1985

Chapter 4:

Needham and Ades: "Integrated Services using a Baseband Local Network"; to appear in Proc. IEE INCUT 87 Conference, London, June 1987

Chapter 5:

Ades, Want and Calnan: "Protocols for Real Time Voice Communication on a Packet Local Network"; Proc. IEEE ICC 86 Conference, Toronto, June 1986

Chapter 7:

Ades: "A High Speed Network Interface for Integrated Services"; to appear in Proc. IEEE INFOCOM 87 Conference, San Francisco, April 1987

Chapter 8:

Ades and Swinehart: "Voice Annotation and Editing in a Workstation Environment"; Proc. AVIOS 86 Conference, Alexandria, Virginia, September 1986

also published as XEROX PARC technical report CSL-86-3. This material is the copyright of the Xerox Corporation and is included with permission.

Fuller treatment of the material in sections 9.2.2 to 9.2.4:

Ades: "A Workstation Terminal for Multi-Media Office Systems"; Proc. Seescan Conference on Imaging and its Applications, Cambridge, October 1986