Protocols for Real Time Voice Communication on a Packet Local Network

Stephen Ades, Roy Want and Roger Calnan
University of Cambridge Computer Laboratory

Corn Exchange Street,
Cambridge CB2 3QG, England

Phone (0223) 352435
Telex 81240 CAMSPL G

Abstract

There is currently much interest in alternative architectures and networks for the provision of Integrated Services. A project at the University Computer Laboratory has been investigating Integrated Service provision in a local area context using a slotted ring: the scope of this work ranges widely from protocol design for simultaneous transport of voice, data and images to user level Integrated Service facilities, e.g. multimedia editors and image manipulation.

This paper addresses the lowest-level simultaneous-transmission of voice and data on one network. In particular, since our network is packet based whereas traditional voice-carrying networks have been circuit switched, it considers the design of protocols for packet voice to produce delays no greater than those in current circuit switched PABXs - a capability without which packet systems simply are not practical for general use.

Our work on packet-based protocols has been fuelled by the fact that commonly accepted telecommunications approaches to voice/data transport are based on assumptions, from some time ago, that bandwidth is expensive and that its usage must be maximised and optimised. These assumptions are no longer true within current local area network technology.

1. Introduction

In constructing an Integrated Services network, we have placed emphasis on four particular areas:

(1) low level protocols for simultaneous transport of voice, data and images

(2) a modular design of servers which implement the network in an elegant and cost-effective way. We consider our design *elegant* in the way in which simple servers build up (a) functionality equivalent to a modern PABX and (b) a logical basis on which higher level services can be provided to the user.

(3) reliability of the system: much greater availability is expected of a phone network than of a general computer network. If a 'total communications system' is to be built from computer-like objects then these must be made highly reliable, with emphasis upon a graceful degradation down to basic telephony services in the event of failure.

(4) user level integration: we feel an important part of Integrated Services to be the provision of new integrated facilities to the user at his workstation. This extends beyond conventional ISDN philosophy, which stops at the use of a single network to carry a variety of traffic only as far as single service terminals.

The second aspect of our work has been reported in detail [1], showing how our modular design approach facilitates construction of the higher level services. This paper concerns lower level issues, but in order to view these in context, the next section summarises our approach to modular system design.

2. System Partitioning

The basic philosophy of our design was first to remove almost all avoidable complexity from the phone itself. Given a background in conventional distributed computing there would be a tendancy to add intelligence to the phone in order to decrease the dependence on other, fallible, components of the system. However we dislike this approach because

(i) it increases the complexity of the phone, and hence its production and maintenence costs - maybe not by much but this increment must be multiplied by a very large number of phones - *and*

(ii) to place functionality in the phone is to determine there what functions are offered to the user.

When network services are upgraded and improved, this implies that the phone itself will have to be altered - something we wish to avoid. (A PTT would rather modify all its exchanges than all its phones!)

Our phone design achieves simplicity by (a) using a very simple voice transmission and reception protocol and (b) avoiding any knowledge of network configuration or system functionality. The topology and configuration of the network is understood by a controller (replicated for reliability). This controller tells the phones to connect at different times to various servers in the system which provide advanced functionality.

These servers implement simple components of the system function according to the following philosophy: a server which implements a single function will perform it rather better than a large software package designed to implement a number of rather different tasks simultaneously. Its software, being a small logical partition, will be much cheaper to build and maintain than an equivalent part of a rather complex operating system. As with the phones, we avoid the need in the servers for knowledge of other components in the system. By designing a simple functional interface to each server and using the controller again to configure these servers in ways which provide particular services, we avoid the need to modify existing servers when another server is modified/introduced.

Within the voice domain, two examples of servers are

(i) a conference server, the function of which is to take voice streams from various sources, combine them in some manner such that each sender will hear the sounds of all other senders, and then re-emit voice streams to the appropriate destinations

(ii) a translator, designed to receive voice streams (e.g. from phones), package them into long data blocks and send these somewhere else, e.g. to a file server for storage. This is an example of *a single function per server*: phones are kept simple by only handling a very simple stream voice protocol. This, as will be explained, consists of very short data packets and such a stream is not convenient for a file server - handling such a protocol in real time would have serious implications for its performance. We would like our file server to store voice, data and images in a uniform manner for the sake of elegance of higher level Integrated Services. Instead of building separate voice and data file servers for real time performance reasons, we build a general-purpose file server, plus a translator which partitions out the real time stream handling.

Our workstation conforms to the 'single function' philosophy in a perhaps surprising manner. Workstations do not in fact handle voice at all - integration of voice into the workstation is performed not at the physical level but as a software binding between a complex workstation and a very simple phone. This has two motivations: first to make the complex and unreliable workstation a part of the phone network is to compromise the reliability of the phone network; secondly a workstation is designed to handle high level software packages in an efficient manner: this is incompatible with real time handling of voice streams.

3. Protocol Requirements

The main purpose of this paper is to describe a protocol layer for Integrated Services traffic upon a slotted ring. Use of slotted rings for data transmission is well understood and widely reported [2] as are intelligent data-oriented interfaces to such a network: therefore the matters of interest here are

(i) transport of voice in a packetised form,

(ii) how voice and data may be carried in the same network *and*

(iii) a 'voice provider' service to give a convenient interface from packet voice streams to processes which consume or generate voice.

Historically, in real systems voice has been conveyed by circuit switching: this will remain universally true until packet techniques can be shown to produce acceptably small delays. Our approach to voice tranport is an attempt to provide real time performance comparable with circuit switched systems. A number of packet networks have been designed to achieve this by separating voice transport at a hardware level (e.g. using separate voice and data slots in the same slotted ring [3] or by the use of different classes of block [4]). Whilst this is an improvement on two separate networks, it is attractive to consider a system in which slots for voice and data are not distinguished at the hardware level or indeed at all - otherwise we veer towards the ISDN approach, where different services have different network terminations. The slotted ring used for this work is such that there need be no separation of slots for voice and data.

Not only do we avoid slot separation, but also it becomes possible to transmit voice and data each in their 'natural' modes; using a stream-like approach with guaranteed bandwidth for voice and a data-block approach with as much bandwidth as happens to be available at a given instant for data. This is very different to for example the ISDN, which was essentially designed for and is best suited to voice traffic. In the ISDN, voice works in its 'natural mode' and data is something of a second class citizen. The fact that we can simultaneously treat voice and data as first class citizens is a function of the properties of our ring, which we now describe.

4. The Cambridge Fast Ring

Our slotted ring, the Cambridge Fast Ring [5], is a derivative of the better known Cambridge Ring. The current microcoded implementation of the Fast Ring achieves a bandwidth of 50Mbs^{-1}: a VLSI realisation to give 80-100Mbs^{-1} is well advanced. Each slot in the Fast Ring has a data field 256 bits long. The properties of the Ring which are most relevant to combined voice/data transport have been described before in the literature [6] but are sufficiently pertinent to this paper to be summarised here:

(i) *Equal bandwidth sharing amongst equal requesters*: if N users simultaneously request maximum bandwidth, they will each receive an equal share of the total. This is not unusual in local networks, but note that the total usable bandwidth does not fall off as the offered load increases (c.f. simple CSMA systems).

(ii) *Fine granularity of bandwidth sharing*: in for example a CSMA or token ring system, if the packet size is small the available bandwidth drops off sharply. This is not the case for a slotted ring: bandwidth sharing may be done at the slot level without loss of efficiency and it is perfectly normal for different users to be using dramatically different packet lengths.

(iii) *Bandwidth sharing amongst unequal requesters*: where P small users and Q large users of bandwidth make requests simultaneously, the P users' needs are satisfied extremely quickly and the Q users take equal shares of what then remains.

These three properties make a near ideal climate for voice and data sharing a single network. For data transmission (e.g. a file transfer or a screen refresh) we desire as much bandwidth as available at the instant of demand. In a typical data network there will be many logical connections at one time, the majority of which are idle at any instant. When a connection is not idle (it might say be loading an executable file across the network) it is desirable that the transfer happen as fast as possible.

In contrast the bandwidth requirements of voice are modest but the bandwidth must be available when required and should be formed of small packets (for reasons of delay - discussed further below). In a data system we do not wish to limit the number of simultaneous logical connections - if all connections become active at the same time it is expected that all will proceed slowly. In contrast for voice we are willing to limit the total number of calls but require that bandwidth be available for those calls on demand. Property (ii) above ensures that the voice can be sent in suitably small packets. The other two properties enable voice and data to be merged conveniently: the voice is clearly the P users and the data the Q users. In a typical Fast Ring the phone can expect to acquire a slot within roughly a tenth of a millisecond of requesting it.

5. Properties Required for Voice Connections

In order to justify the design of our voice protocol we need to consider two issues - acceptable network delay and acceptable error rate. In a packet network, delay will come from three sources: packetisation delay, minimum transport delay and statistical queueing jitter. The minimum transport delay can be neglected within a local area context, being mainly a function of the speed of light. The statistical queueing jitter occurs due to the fact that in a packet switched system bandwidth is not actually prereserved so that at any instant demand may exceed supply. For this reason some packet network builders (as mentioned above) have proposed slots specifically reserved for voice. We contend that this is unnecessary, since in our network if the phone only wishes to obtain a single slot at a time the maximum waiting time is very small*.

*This is not true under all possible loading conditions, but given a 50Mbs^{-1} network and 64kbs^{-1} voice streams, it would take a very large number of stations on a single ring to make it false: the Cambridge Fast Ring design is of small rings linked by slot-level bridges and that number should not ever even remotely be approached.

The most serious source of delay is packetisation, for a packet containing time t's worth of samples implies a delay of t due to packetisation. For this reason we send very short packets of voice, something to which our network is well suited. Our normal packet length is 2ms - i.e. 128 bits of standard PCM-encoded voice. This length was chosen for three reasons: it fits into a single ring slot and hence the transmission scheme in a phone can be very simple. The delay induced is of the right order for speech in a local link. Equally important, this packet length is the unit of 'not actually caring whether a packet is delivered' - as explained below.

Error rates acceptable for voice have been investigated many times: an oft quoted figure is 0.1% [7]; more recent work has produced a more conservative requirement of 10^{-5} under low sound level conditions [8]. Which figure is appropriate depends on the types of error on the link. Simple random errors are irrelevant in the context of our network: the error rate of the Cambridge Ring is typically better than 10^{-10} [9]. At that level, it is better to play back packets known to contain errors than to reject them on checksum. Our concern 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). 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.

In packet voice systems we are therefore interested in the ear's perception of burst errors rather than poisson-distributed single bit errors. Studies have shown [10] that the ear is not affected by bursts occuring up to 1% of the time provided each burst lasts less than 4ms. Thus the 2ms packet really is a unit of 'not actually caring whether a packet is delivered'. It is important to realise that in the context of a packet based stream, it is very difficult to build a service with complete guarantee of fast, first time delivery. We have adopted the alternative approach of designing a service such that it does not need to make any such demands.

The target in our voice protocol work is to achieve phone-to-phone delays of a few milliseconds: our present protocols work in the range 2-5ms. Telecommunications engineers may object that the total delay currently allowed for PABX equipment is a maximum of 2ms. The reply to this is that we are interested in showing that packet-based systems can work in the right ball-park, i.e. a few rather than a few tens or hundreds of milliseconds. Our protocol could be made to work to CCITT standards by reducing the number of bits sent in a slot below 128, although the utilisation efficiency of the ring would eventually begin to suffer.

6 The Voice Protocol

So far we have expressed a desire to send voice as a stream of 2ms packets. Since random errors on the ring are not frequent enough to concern us, we need not 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 place for flow control, since both the network and any consumer of voice must guarantee to handle 64kbs^{-1} per stream. Acknowledgements cannot be of any use: in data applications if a block is delivered containing errors or is not delivered at all, the transmitter will time out, the acknowledgement and retransmit. In a voice application the retransmitted data would arrive too late to be replayed, 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 consistant 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 [11], but this is imcompatible with our approach - we are talking about constructing a network in which the jitter has a predictable and small upper bound rather than the approach of saying that packet networks will have unpredictable and large jitter, with which we cope using adaptive buffering. In practice it is hard to make such adaptive schemes work consistently well [6] and besides we claim that in a practical local area scheme the delay due to such buffering is inadmissable.

We have already accepted that occasionally packets either fail to arrive or arrive with excessive jitter. It is important to realise that our protocols are very lightweight simply because the Ring is not a hostile environment. We feel that the protocol should be made as lightweight as the environment will allow; that a packet protocol attempting to work across both local and wide area connections is a mistake because it will either

(a) be built for the local area and hence not withstand more hostile wide area conditions or

(b) be heavyweight enough for the wide area and thus unnecessarily heavyweight for local use - not only making the phones unnecessarily complex but also giving unacceptably large delays.

Our voice protocol is therefore not adaptive - it can be termed 'synchronous' in that both transmitter and receiver have a notion of time linearly increasing. However the two ends neither hold the same absolute time nor know the time difference between oneanother. This can be explained by describing how an individual voice stream starts 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. When it receives the first packet from x, y delays it for time j before starting a regular playback of samples. This allows for the case of a first packet with zero jitter, and hence 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 of a first packet with j's worth of jitter and ensure that there is sufficient buffering in the phone to accommodate packets with no jitter*. All will be well now iif packets from x arrive with jitters in the range $0..j$: there will never be insufficient room in the buffer to store them, nor a lack of samples to be played back upon demand. The 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, consumed in receiver buffering.

We now consider a packet arriving with a greater jitter than j, or not arriving at all. We have already established that such occurances, unless very frequent**, will not upset the ear, but must ensure that they do not upset the protocol. Suppose that a packet is heavily delayed. The transmitter generates packets at regular intervals. The receiver maintains two pointers: 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 the first pointer overtakes the second, the condition under discussion has occurred. The CODEC's sample-fetch routine sets a flag indicating that it will continue to increment its 'next sample' pointer (to retain synchronism) but will play back a D.C. level instead of samples picked up from the buffer. When the delayed packet arrives, the protocol observes that the CODEC's 'playing back silence' flag 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 was 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.

Suppose a packet fails to arrive at all: we need a method to maintain synchronism. 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 next sequence number to arrive. Suppose that the sequence number indicates one missing packet. The 'next packet' pointer is therefore moved up the length of one packet before inserting the packet. Since in our system j should be less than d, when the packet arrives the flag 'playing back silence' will almost certainly be *true*. Therefore, after moving 'next packet' to denote the missing packet, the space between 'next

*Clearly if the playback duration of the packet d is greater than j then the buffering must be at least $(d+j)$.

**or regular - the ear is very sensitive to noise which forms a coherent pattern but not to a fairly high level of block loss if truely random.

sample' and 'next packet' is filled with D.C. level samples. Then after the new packet has been added to the buffer, 'playing back silence' can be set to *false*.

This completes the account of the voice protocol under 'normal conditions'. A few further issues remain to be tackled: first how the phones maintain synchronous clocks, secondly some practical details as to how the voice stream service is implemented for servers as different as phones on one hand and conference servers/translators on the other. The reader may also have noticed some additional error cases which need appropriate handling: these will be discussed. Finally the place of silence detection and suppression in this scheme is considered.

7. Distribution of an 8kHz Clock

In the above algorithm, whilst two phones need not know the time difference between their two clocks, the scheme only works 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 and also we desire a system where failure of the clock distribution service does not stop phone-to-phone interactions. Our phones contain a crystal trimmed by a VCO. Thus the sampling frequency is always close to 8kHz (as required by the filters in our CODECs for comprehendible 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 make the two equal (adjusting with a suitably long time constant and in small increments).

Thus two phones initially set at different frequencies will adjust to the mean of the two. Suppose that a phone is receiving voice from either a wide area network gateway or from for example a translator. The wide area interface will contain a stable clock and we also arrange that our translator receives a similarly accurate clock. Neither of these will adjust to the phone, so that the phone will adjust exactly to them. However a problem arises if a phone at the wrong frequency is sending packets to a translator (a simplex stream, unlike the interaction of two phones), since our 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 absolute clock through their receiver algorithms. We have an interest in ensuring that the phones normally run at a canonical frequency.

Our novel approach to this is a service called the 'speaking clock'. When a phone's handset is down, instead of closing its voice input channel it listens to a stream from the speaking clock. This stream is a perfectly ordinary voice stream, but emitted by a server with an accurate time reference. The phone's clock thus adjusts to this reference. It was explained above that the phone's reception algorithm can cope with missing packets. In the stream from the speaking clock nearly all packets are missing - it only sends a packet to each idle phone roughly once a minute. Thus both the network bandwidth overhead and the complexity of the speaking clock itself are trivial, but the use of sequence numbers in the voice stream enables the phone's clock to be accurately maintained.

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

8. Provision of a Voice Service

The next issue to consider is how a voice delivery and transmission service should be provided to a user in (a) a phone (b) a server handling several streams simultaneously. The answers turn out remarkably similar and are contained in a service which we have defined and implemented under the title of a 'voice provider'.

Our approach to this service, like our whole approach to voice protocols, comes from the observation that voice streams behave very well on our 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 like 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 above. When a client calls the voice provider to set up a voice reception channel, it specifies locations of the pointers and flags needed in the voice reception protocol. As it receives packets, the voice provider then updates the 'next block' 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 block', setting 'playing back silence' to *true* at appropriate moments*.

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

(a) a circular buffer of sufficient size *and*

(b) code in the provider to handle a few error conditions that can occur.

The minimum size of buffer required is in fact not $2j$** but rather $4j$ (or $4d$). This is because the tests for abnormal conditions are done on the basis of whether pointers are 'in the correct half of the buffer' (otherwise terms such as 'behind' or 'in front of' have no meaning), implying that normally no more than half the buffer can be in use at a time.

The error conditions which may arise concern cases where the reception buffer is either habitually overfull or 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 this can happen. One is that the first packet, against which synchronism is measured, is unusually delayed so that the buffer is then constantly overfull. We have observed the other case 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. When the frequencies do equalise, the buffer may be left persistently too full or empty

The provider therefore checks for continual occurences of buffer overfull or empty (not single occurences as our protocol is designed in expectation of these) and in such a case simply reinitialises the reception pointers - effectively when an error condition is detected the provider causes the next received packet to be delayed for time j before playback, just as at startup. It will also take suitable actions upon detecting abnormal leaps in the sequence count. It discards a single block 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.

The way in which the value of j is determined is noteworthy: in our system it is given to a phone (or other server) by the system controller as part of the 'listen to y' command. For phone-to-phone links j must be very small, since real time delay on local links must be minimised. This is not true of all sources - consider a translator talking to a phone. This is clearly not a real time link;

*If the reader is becoming baffled by detail at this point, (s)he should simply note that this is really only the standard procedure for manipulating a circular buffer.

**Or $2d$, whichever is the larger: it is convenient 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.

furthermore the translator has a more complex interface than the phone, so that it may have more difficulty than a phone in launching a rock steady stream of packets. This can be accommodated by increasing j in the receiving phone a little (within limits set by the fact that the phone must have $4j$ of buffer available) such that the task of a translator becomes much easier. This technique can be applied further: suppose a wide-to-local area bridge has similar trouble in transmitting. Then it could increase j, but note that

(a) this is a real time application *and*

(b) the extra delay induced must be accounted in the overall delay specification of the wide area link.

All clients of the voice provider use circular buffering, with explicit contact only to start up or close down a stream, but different clients have slightly different needs: the voice provider therefore provides a number of different options. For a translator we do not want space to be left in the circular buffer to denote missing blocks, but want the occurance of missing blocks be flagged in the buffer. In the phone and the conference server we wish spaces to 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. These are however slight differences and the outline behaviour of the provider is the same in each case.

9. Silence Suppression

In papers on packetised voice the writer will generally argue somewhere that packetised voice can be much more efficient than circuit switching, if use is made of silence detection and the TASI advantage. We on the other hand do not suppress silence transmission from phone to phone: we are building a packet network not to economise on bandwidth (which is cheap and plentiful) but to provide an environment for integration of different types of traffic in a natural manner. Our reason for not wishing to suppress silence from phone to phone is that when listening to a speaker who is in a noisy room, there is a very noticable and irritating difference between the background noise during talk spurts and complete silence during suppression. There is no need for this annoyance, since bandwidth is not an issue. However when transmitting our phones do detect silence periods (using very simple hardware) and set a flag in each transmitted packet appropriately: packets marked as silence can then be suppressed when for example storing voice on a file server or sending the stream over a satellite link or similar wide area connection where the cost of bandwidth is important. Simple silence detection gives an appreciable saving in bits stored or transmitted*. When the voice is recreated for playback in another phone, either it will contain the irritating gaps or alternatively processing power can be used to synthesise the missing background noise, this being an acceptable cost for the reduced bit rate. However we would be loathe to place such processing power in the phone, because we want to keep the phone simple.

10. Image Transmission

This paper has concentrated on the integration of voice and data. To produce a general integrated system, images (from single frame transmission to full frame-rate streams) must be incorporated into the transport framework. Work is in hand in this area. Little will be said here other than that our schemes for compression of images are based on the same policy as for voice streams, i.e. the observations that

(a) bandwidth saving is not the single crucial issue *and*

(b) it is much more difficult to produce a highly reliable stream delivery service in a packet network than to design a protocol tolerant to bursts of data loss.

Therefore we are working on compression algorithms which trade efficiency against a combination of coding simplicity and redundancy specifically aimed at accommodating error bursts.

11. Conclusion

The design of the packet voice protocol set out in this paper is primarily motivated by

(i) the desire to use a packet network for Integrated Services provision because it produces a better basis on which to provide higher level integration in a natural, cost effective manner

(ii) the need to produce a voice stream service giving delays comparable to those currently required of the conventional PABX

(iii) the requirement that a packet-based phone be a very simple entity

Integrated transport of voice and data in the same network is made very straight-forward by the low level behaviour of our slotted ring: there is no need for example to use slots designated for voice stream traffic.

Our voice transmission protocol is very lightweight because

(i) conditions for stream transmission in the local network are favourable, so that no heavyweight protocol is required

(ii) a heavyweight protocol, as normally used for data, is of no use because of the tight delay requirements for voice streams.

This lightweight approach appears to be satisfsactory, in that our phone system does indeed give good sound quality at the same time as very short delays.

All handling of voice streams in our system is performed by a service known as the 'Voice Provider' which reflects the style of our protocol: since the handling of streams in the network requires little supervisory effort, producers and consumers of voice do not need to interact in a complex manner. They use a circular buffer into which the producer places digitised voice samples and from which the consumer takes them. The circular buffer is made sufficiently long that such a simple approach will produce acceptable sound quality. Consumer and producer communicate *implicitly*, i.e. by sharing pointers and flags, rather than *explicitly*, i.e. using a conventional procedure call and return mechanism. This gives a very efficient service, which is of great benefit in both reducing the processing power required in a phone and increasing the capacity of servers which handle several streams in parallel.

*Better compressions can be obtained from for example LPC [12] techniques, but this is much more complex than silence suppression and also causes appreciable delay.

References

[1] Needham and Ades: "Integrated Services using a Baseband Local Network" (to be published)

[2] Needham and Herbert: "The Cambridge Distributed Computing System"; Adison-Wesley Publishers, ISBN 0-201-14092-6 (1982)

[3] Lazar et al.: "MAGNET: Columbia's Integrated Network Test Bed"; ICC, Chicago 1985

[4] Bux et al.: "A Local Area Communication Network based on a Reliable Token Ring System" Proc IFIP - TC6 Symposium on Local Area Networks, Florence 1982

[5] Temple: "The design of the Cambridge Fast Ring" Proc IFIP WG6.4, - Workshop on Ring Technology based Local Area Networks, Kent 1983

[6] Adams and Ades: "Voice Experiments in the UNIVERSE Network"; ICC, Chicago 1985

[7] Bullington and Fraser: "Engineering Aspects of TASI"; Bell Systems Technical Journal p353, March 1959

[8] CCITT Planning of Digital Systems: Special Study Group D; Contribution 103, June 1974

[9] Dallas: "A Cambridge Ring Local Area Network Realisation of a Transport Service"; Proc IFIP WG6.4 - Workshop on Local Area Networks, Zurich 1980

[10] Gruber and Strawczynski: "Judging speech in dynamically managed voice systems"; Telesis 1983 Two, pp 30-34

[11] Kleinrock and Naylor: "Stream Traffic Communication in Packet Switched Networks"; IEEE Trans. Comms., Dec 1982

[12] Markel and Gray: "Linear Prediction in Speech"; Springer Verlag (1976)