Inter-Office Memorandum

To	File	Date	September 13, 1981
From	S. M. Ornstein, L. Stewart, D. Swinehart	Location	Palo Alto
Subject	Voice Project Proposal	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPOutline.bravo

ABSTRACT

This memo presents what we think is an appropriate voice project for CSL, discusses visions of new functionality that might come into our lives as a result of the endeavor, describes the enabling architecture as we see it, and then discusses a plan for setting out.

Attached are documents describing our present view of the three major system components and several appendices describing alternative architectures, voice transmission issues, and some voice hardware proposed by SDD.

INTRODUCTION

Why voice? We see two domains. First, in the real-time world, our information management skills can give us improved control over voice communications (taming the telephone). Second, when we can integrate voice with our other endeavors (voice as data), we can add an additional dimension to all those activities. By ‘‘voice’’, we do not mean ‘‘audio’’, as in music, and we do not mean ‘‘speech’’, as in speech synthesis or speech recognition. We do mean the integration of telephone service and the recording and retrieval of stored voice.

We have concluded that the most useful way to begin adding voice to our systems is to provide a new and better, Ethernet-based, form of telephone service. It seems likely that we can make genuine improvements in our own lives; and, in the process, we will build those fundamental tools for handling voice from which other extensions will grow naturally.

Our eventual goal is to build fully integrated systems that include voice as naturally as our systems now include text. Within about two years, we plan to provide every member of CSL with an Ethernet telephone, or *Etherphone*.

The Etherphone system will transmit voice and control information over the Ethernet rather than over conventional phone wires. The Etherphones themselves will serve as simple terminals (ethernet peripherals), without much intelligence of their own. Users’ *workstations* will provide enhanced user interfaces. A *Voice File Server* will enable storage and retrieval of voice. An *Etherphone server* will control the collection of Etherphones and generally manage the system. A gateway function will connect the Etherphone system to the public switched telephone network.

In addition to those mentioned above, Susan Owicki, John Ousterhout, and Bill Nowicki have been involved in the preparation of the voice project proposal.

OUTLINE

The remainder of this document consists of eight sections:

Visions: Describing six visions for the future -- ways we think that an integrated environment including the telephone system can improve our lives.

Architecture and Plans: Describing with a broad brush our proposed system architecture and its major components. A rough schedule and plan of action is attached, as are a partial list of unresolved issues.

Etherphone: Describing our present design and implementation ideas for a microprocessor based Ethernet telephone.

Etherphone Server: Describing our present design and implementation ideas for a Cedar based controller for the telephone system.

Voice File Server: Describing our present ideas for an Ethernet file server for voice messages.

Alternatives: Describing some of the alternative system designs and system components we have considered.

Etherphone Protocol: Describing in some detail the issues of Ethernet transmission of interactive voice. Descriptions of the operational Etherphone 0 and Voice File Server 0 are attached.

Voice vs. Data: Some thoughts on the relationship between data communications and real-time communications and a proposal for incorporation of class-of-service into our internet protocols.

Voice Peripheral: Describing a voice and telephone management box for workstations. We propose to copy the design as part of the CSL Etherphone.

Inter-Office Memorandum

To	File	Date	September 14, 1981
From	S. M. Ornstein, L. Stewart, D. Swinehart	Location	Palo Alto
Subject	Voice Project Visions	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPVision.bravo

VISIONS FOR THE FUTURE

We do not necessarily advocate all of the features we can envision. In truth, we do not know exactly what we will find useful and elegant. The provision of plain-old-telephone-service is a complex enterprise; we may be able to do a better or less expensive job of it than has been done, but that alone is not sufficient reason for us to undertake a large scale project. We have marshalled our thoughts and come up with a series of visions of problems as they exist today and of ways in which an Etherphone system could improve our lives. We expect that the design of particular functionality (integrated systems) will be a continuing challenge long after a sufficient number of capabilities have been built.

Most of our proposed improvements in ‘‘telephone’’ service have to do with getting phone participants more smoothly into direct voice communication -- or circumventing (supplementing) the need for such direct communication through voice messages. We argue that the telephone provides reasonable communication once you are talking to the other person. It is in the process of negotiating the establishment of a conversation between the participants that the phone system is most inadequate and annoying. For principals, a personal secretary mediates both incoming and outgoing calls and thereby takes over most of the burden. We hope that we can provide some of this same relief for others. We also believe that voice in messages and documents will find application far beyond simple telephony.

We must be careful, because the conventions of phone use are deeply established. Although people are often annoyed by the present arrangements, they also tend to be quite conservative and to resent changes in the system. Our proposals could cause substantial shifts of burden between callee and caller -- as will become evident below.

The reader must keep in mind that there are problems for which we would like (and could use) solutions but which we do not intend to try to solve. In particular, we do not plan to address issues of artificial intelligence (speech understanding), speech recognition (single words), or even speech synthesis from text. Obviously such capabilities could greatly enhance our systems, but we lack the resources for such work and further, when such capabilities become available, we will be able to fairly easily incorporate them into our systems.

We present here six visions for the future -- ways we think that an integrated environment including the telephone system can improve our lives.

1. BETTER PLACEMENT AND RECEIPT OF CALLS

The Caller's Plight

Often one does not know and cannot find the desired number. The phone book cannot be found or is out of date. Sometimes when looking for a service rather than an individual, one may not even know what to look for. The organization of a directory often does not match one's needs: ''Well, his first name is Tom.''

Even after a number is found, one's call attempts are often stymied. It is simply too much work to dial and re-dial a phone. The number is misdialled, the line is busy or doesn't answer. (A modern day variant is ''please hold on, all of our agents are busy.'') The call may be answered, but the person sought not be available. The call may be answered by an inflexible answering machine.

These problems occur in subtle combination! The call is not answered, but it was the wrong number anyway.

Solutions

Most of the difficulties of finding the right number can be solved by placing calls by name rather than number and by use of appropriate data bases (e.g. *white pages, yellow pages*). These matters have little to do with telephones, but we can smooth the interface of data base systems and telephone systems by automatically placing calls (autodialling) rather than requiring the user to copy the number from the data base terminal to the telephone.

We can also provide ''call placement'' services at a level well above simple number translation and automatic dialling. We could construct a calendar system incorporating a schedule of calls, and note taking and time accounting facilities. A lawyer might be interested in such a system.

The difficulties which arise when the desired party is not reached are both more difficult and more closely associated with telephones. We will have to distinguish between calls placed *within* our new system and those calls placed to points *outside* the system. For internal calls, the calling station will be able to distinguish between busy and no-answer, and perhaps could know who answers. The calling party could then choose to leave a message, to try again later, or to try somewhere else. Handling all these cases for outside calls would require signal processing and speech understanding beyond the scope of our intentions; we plan to let the caller listen in and decide what should happen next.

The callee's plight

People are often busy with tasks more important than handling a particular call. One often wishes to know who is calling, what the call is about, and how long will it take. We live in a social environment in which answering the telephone has unduly high priority. In addition, one is often out ''for just a moment'' and misses calls. Answering machines seem too impersonal and the receptionist too remote. In essence, we would like to provide everyone with the same services provided to those who have personal secretaries.

Solutions

Let your telephone help field calls. For arriving *inside* calls, one's telephone can know who is calling. We will develop ways of letting the caller indicate the relative importance of calls and the general subject area. We will provide a *call filter*, instructions to one's telephone of the flavor: ''No calls for half an hour, except if it's Bob or if Fred calls about the budget.'' If the caller *really* wants to get through, we can provide an override button. This negotiation process can even

proceed if you are already using the phone for something else.

These kinds of functions will be much harder to provide for calls arriving from outside the system. An expert calling in from outside will be able to use pushbutton phone signals to engage in negotiations. For other callers, the filter will operate as for internal calls, but will not directly support negotiations. Instead, callers will be forwarded to an attendant who can act as their agent.

It is clear that new burdens will fall upon the caller, if only to judge the importance of calls. One is used to barriers when one makes a call to or receives one from a person with a secretary, but otherwise, as a caller, one is used to getting through and may resent buffering by a non-human mechanism. Although we can try to soften the resentment by making the mechanisms as polite as possible, there seems no way to get around the fact that we are redressing an age-old imbalance -- the caller has traditionally had the ability to interrupt the callee.

The reader may refer to the attached figure detailing a possible call negotiation scenario for some feeling of the complexity of these matters!

2. BETTER MANAGEMENT OF FANCY FEATURES

Advanced ‘‘features’’ of traditional telephone systems are too hard to use. Such features fall into two classes: those used separately from use of the phone, and those used during calls. Forwarding is a good example of a facility used while *not* making a call. It is too easy to forward one’s phone and then forget about it. One often receives forwarded calls intended for someone else. One often forgets to forward one’s phone before leaving the office. Features of the second class, those used during a conversation, include setting up a conference call (including a third person in an existing conversation), and call-waiting (receiving another call while you are already engaged in conversation):

> ‘‘I have one person on hold, want to form a conference, but the second callee doesn’t answer. Let’s see now . . . flash, pause, . . . now what do I do next . . .’’

> ‘‘How exciting to be part of the first interstellar phone call! [beep] Uh oh, I have another call, could you wait a moment, your excellency? [flash] Hello? A year’s supply of what? No, I’m not interested! [slam] oops. . . ’’

Solutions

We will use our workstations to control the telephone, bringing all of our user interface expertise to bear. We will clearly indicate the *state* of the system: filters, forwarding, calls on hold. We will provide reasonable user interfaces for *controlling* features. It will be possible to enter control commands, under password control, from anywhere in the system, not just from one’s office. Such capabilities will allow much better control over as seemingly simple a matter as call forwarding.

While we do not neccessarily advocate any particular feature, a computer controlled telephone system could also provide such features as paging, intercom, personalized dial-tones, personalized ringing, and so on. Although we don’t know exactly how to implement it, suppose everyone wore a tiny device (Locator) which locally broadcast his/her identity -- to the nearest telephone. Such a mechanism would allow the system to recognize when you were near your phone. Thus it could desist from trying to set up a call for you after you left your office. Furthermore, as recipient of your calls, your phone, using its Locator, could change its answering strategy when you left your office. It could begin to answer calls with ‘‘He’s out; please leave a message’’. Alternatively, with a more sophisticated system, the phone nearest whereever you were could ring with a ‘‘signature tune’’ -- assuming your profile indicates that’s what you wanted.

There is an implication that telephones not associated with nearby workstations will need better user interfaces. The telephone, extended in the ways we envision, is too difficult to control with only twelve buttons and the switchhook. A larger keypad or keyboard and a one line display seem appropriate.

3. BETTER HUMAN-ASSISTED CALL MANAGEMENT

This section discusses possible solutions for the telephone problems associated with attendants and receptionists. The kinds of problems that arise are usually due to insufficient information presented to the attendant, too great a workload, and poor communications between the attendant and potential callees.

The caller's plight

This situation is not too bad for a person calling into a *good* manual attendant system. A light may indicate exactly who is being called. The attendant may know the callee is out and answer immediately. In more ''advanced'' systems -- which are usually bought for reasons of cost-reduction rather than improved convenience -- the attendant doesn't know who is being called, the phone rings for a long time before anyone answers, and one is always put on hold. (These problems are particularly acute when a call is forwarded several times before the guard at the door finally answers -- and he is not equipped to deal with phone calls at all!) The situation is better in some systems: there may be a distinguished console which can display the called number, but usually there can only be *one* such console per installation. In CSL, we have a key system for the laboratory behind the PARC Centrex telephone system.

The attendant's plight

Many calls can be in different states, resulting in lost or mishandled calls. Since the call director or receptionist's console is a special hardwired device, the attendant load cannot be either split or transferred. Usually messages are transcribed and filed on paper.

The callee's plight

Messages don't always get through. If they do, they are usually late. Because one has to explicitly check for messages, it is hard to know when a message has arrived. Messages are too short: ''Please call back Fred.'' By when? About what?

Solutions

We will implement attendant features as extensions to the standard telephone/workstation functions. We will identify the original callee, by name as well as number. We will use a well designed user interface to indicate the status of multiple calls. Potential callees will be able to leave general or individual messages for potential callers: ''If Fred calls, tell him I'm at home.''

Because attendant facilities can be implemented with a standard telephone/workstation setup, the attendant's duties can be easily transferred (including calls in progress!) or shared. Messages for callees will be *voice messages* instead of handwritten notes.

We should add that we have not neccessarily invented many *new* features. Modern PABX systems can handle these kinds of functions, but often the facilities are only available at the main console. Subgroups within an organization do not have good access to the facilities and in any event, the

functions are not well integrated with the office environment.

4. VOICE MESSAGES

Voice messages are a quite general facility. It might be better called ‘‘storage, retrieval, and editing of segments of voice.’’

The Curse of the Dictating Machine

> . . . . .wonder if *this* will make the goddamn thing work . . . testing . . . testing . . one, two, three Dear Mr. Brown colon paragraph I wonder if you have any idea how much trouble your blasted firm no leave out the blasted has caused our no better make that we have had a good deal of trouble difficulty with the by the way send a copy of this to Peter too with the bobble sockets you have been making for our ouija boards they ouch (click) easy honey tend to fall out . . . . .

The Curse of the Answering Machine

Answering machines, which today are relatively inexpensive and quite powerful, *ought* to solve many of the problems that we have identified. Using answering machines, it is now possible to convey and exchange information even when both parties are not available at the same time. One can also use them as remote dictating machines, as crude call-filtering mechanisms, and so on. Use of an answering device can help redress the callee/caller imbalance.

But in practice, especially in an internal office situation, anyone who has encountered an answering machine instead of the intended callee knows how unsuitable a conversant such a device is; it is one of the more intimidating of modern inventions. Its non-interactive nature essentially demands a well-constructed, composed message rather than an informal communication, and few of us are able to produce such a composition in real time.

Solutions

Digital recording, editing, filing, dissemination, and playback of segments of voice permit us both to construct more reasonable versions of existing facilities and to construct entirely new facilities.

For users within the system the caller, not the callee, will make the decision to leave a recorded message -- or to take some other action -- when an attempted call fails. This removes much of the intimidation. The caller will also be able to thoughtfully compose a message by editing and review before the message is delivered. Finally, our existing message systems can be used to identify and organize voice messages for the callee’s benefit.

For outside callers, we will not rely solely on automatic facilities, for the reasons given above. (Callers who understand our automatic systems may be able to log in using tone signalling, then to behave as if they were internal users.) Rather, we envision providing facilities that will allow human attendants to handle incoming calls better and more efficiently than do current systems. Callers from outside the system, after they are routed to the attendant, will be able to leave quite complex messages without the problems of paper transcription. Inside callers will also have the option of speaking to an attendant when problems arise.

As an example, consider the following hypothetical fragment of conversation between an outside caller and an attendant (this scenario is a liberally revised version extracted from a 1975 memo by

Ed McCreight):

. . . .
Receptionist: ‘‘*Ms. Foozle is not in the office. May I give her a message?*’’

Caller: ‘‘*Yes,*

[Receptionist buttons Telephone Message. A Laurel message form appears, with a text header addressed to Foozle; the following interchange is recorded and associated with that form]

would you tell her that George Geargrinder called, and ask her to call me back at area code 408, 555-8024?’’

Receptionist: ‘‘*All right, Mr. Geargrinder, I’ll make sure she gets the message.*’’

[Receptionist (optionally enters a From: field and a subject field, then) buttons Deliver.]

5. ANNOTATED DOCUMENTS

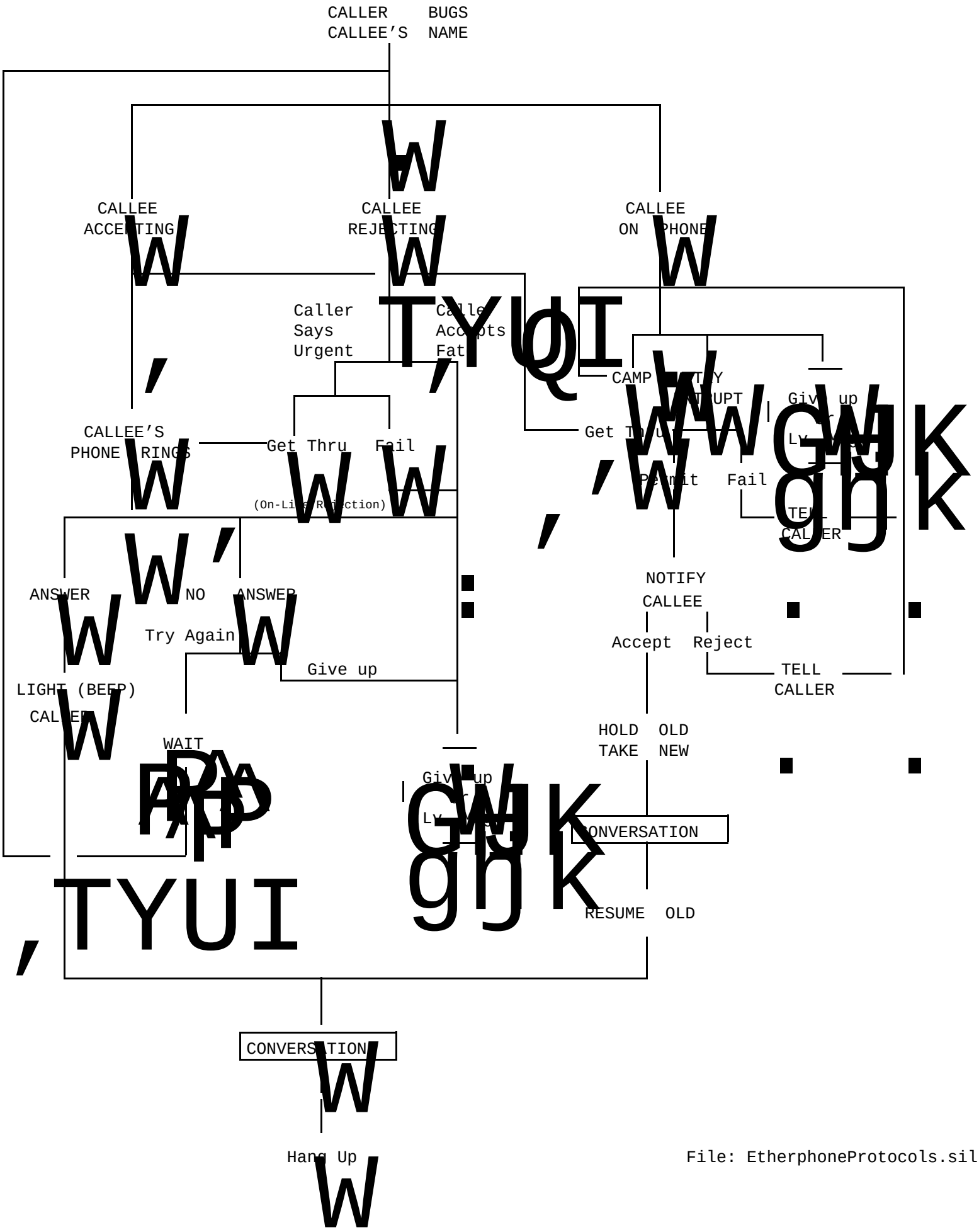
By annotated document, we mean a more or less ordinary text document with voice messages *attached* to particular words, sentences, or paragraphs. Envision an executive making comments on a proposal, or a playwright expressing the voice tone of a particular line. We want to make voice a full partner with text and with graphics.

Begin with an ‘‘ordinary’’ document, for example a Tioga document including text, graphics, and other structure. Select a point of interest and record some comments. A distinctive icon marks the location of a comment. Selecting that icon causes the playback of the associated voice.

It will be possible to obtain a new window permitting one to edit or amend a comment. We might be able to provide sufficient digital signal processing to play back a comment at a higher or lower speed.

6. TELE-TIOGA

Fully integrated systems, mixing text, voice, and imaginal material on an equal footing, will present both unforseen opportunities and unforseen problems. One possibility is ‘‘document-level telephony,’’ wherein two or more participants use their workstations and their telephones to interactively collaborate on the form and content of a document. Both participants see the same display and discuss their work over a conference call. This area is not strictly a voice application, since no special voice capabilities are required. It is just one of the many possibilities for systems combining the capabilities of computers, large displays, and digital and voice communications.



File: EtherphoneProtocols.sil

Inter-Office Memorandum

To	File	Date	September 13, 1981
From	L. Stewart	Location	Palo Alto
Subject	Voice Proposal: Architecture	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPArchitecture.bravo

SYSTEM CAPABILITIES

While the voice project must have some means for shipping voice around, there are additional capabilities required of any system providing the kinds of functionality described in our visions. Without too much predisposition towards a particular architecture, we have managed to group the basic capabilities into a number of areas. Some of the areas described below imply matching collections of hardware, but some describe functional requirements which are quite abstract.

Availability

Availability is less a capability than a requirement. A telephone system should always work. In the large, this means that the system must continue to function even after some of its components break. Essential components, of course, must be designed with high availability in mind. Availability in the small is more subtle: one's phone must still work even if one's workstation is in the debugger.

Terminal Equipment

In the same sense that our workstations usually have a display and a keyboard, the voice system requires machinery for voice input and output and enough of a "digital" user interface to control it. This might take the form of an ordinary desk telephone set with its earpiece *receiver* and mouthpiece *transmitter*, plus a 12-button keypad and hookswitch. It might take the form of a fancier telephone, perhaps with a full typewriter-like keyboard and a one-line display. It might simply be a speaker and microphone attached to a regular workstation display and keyboard. (In any case, it will be possible to connect a variety of *transducers* to the voice terminal, such as a speakerphone or headset.)

Transmission and switching

We require some means of getting voice from place to place. Some possibilities are the existing telephone switching and transmission system, a private switching system such as a PABX (Private Automatic Branch Exchange) using traditional wiring, and the Ethernet. Good quality telephony places lower limits on the performance of any method we might choose. The most crucial performance requirements are related to high bandwidth and low delay. Most telephone industry digital voice uses 64,000 bits per second, although some new PABXs usr more in order to obtain higher quality. While data compression schemes exist which might reduce this rate to as low as 8000 bits per second, they are quite expensive. Delays are generated by speed-of-light problems or by buffering in packet-switched systems. Acceptable delays depend on such factors as background noise and echo level. The transmission system we use must supply enough bandwidth for an adequate number of users without excessive delays.

Control

We must have rapid and versatile control over the transmission and switching system. We must be able to manage connections among people, between people and servers, and between machines. Because we must communicate with the outside world, we cannot entirely rule out generating tones to control the traditional phone system, but the quality of the voice system we construct will depend a great deal on the speed and accuracy with which system control is handled. Some of our visions depend on the flexibility of control arrangements; for example, forwarding requires the ability to change the association between numbers and terminals.

Voice Filing

In order to handle voice messages, annotated documents, answering machine facilities, and the like, we must have a voice filing system with the necessary real-time capability. Although voice messages could be implemented with analog tape recorders attached to terminals, voice segments which might be accessed by many people seem to require digital storage using high performance disks.

Voice composition and editing

Once we begin constructing documents incorporating voice, we will need systems for composing and editing voice. Such systems might range from simple "record" and "pause" buttons through some graphical representation of a voice passage. The voice filing machinery must be sufficiently versatile to handle complex restructuring of a passage (e.g., by something like a piece table).

Data-bases

Although we do not expect to construct data base systems ourselves, we expect to make heavy use of such systems in the implementation of white pages, yellow pages, and perhaps for organizing voice messages.

Encryption

People expect their telephone calls to be private. There are legal sanctions against wiretapping, but wiretapping on a broadcast medium where a wiretap might be just a program would be very difficult to detect. High speed single chip DES encryption chips are now inexpensive and key distribution using a trusted server is straightforward.

ARCHITECTURAL PRINCIPLES

The basic premise of the Etherphone approach is that actual transmission of the voice data, as well as all control information, is done in digital form over the Ethernet. Connections to the outside telephone are done by servers with trunks to the phone company. This scenario has the advantage of complete control over the telephone transmission system. We benefit by the natural multiplexing of the ether and by direct access to voice-as-data. Control of the system is distributed; negotiation for a call takes place between the source and destination Etherphones and other parties.

We plan to guide our implementation by four architectural principles:

Use the Ethernet for both voice and control information. This is the basic premise. We plan both to control the telephone/voice system by digital communications through the internet and to transmit

voice on the Ethernet both for conversations and for storage.

High availability and reliability. We plan to construct a system with as close to the availability and reliability of the existing telephone system as we can.

Keep the Etherphones simple. We plan to treat the Etherphones themselves as simple terminals without much intelligence. The complexities of software and system control will reside in a more powerful server.

Use workstation when available for wonderful user interface. We plan to take considerable advantage of the workstations in most of our offices. Only they can provide the large displays and versatile user input capabilities we will need to provide advanced yet friendly functions. In locations without workstations we will provide telephones with somewhat fancier user interfaces than the usual twelve buttons.

SYSTEM COMPONENTS

We turn to a more detailed discussion of the Etherphone architecture, shown in Figure EPSystem.sil.

Etherphone -- We view the Etherphone primarily as an Ethernet peripheral. It's job is A/D and D/A conversion of voice and transmitting and receiving voice over the Ethernet. These activities are carried out under the close control of the Etherphone Server. The digital part of the user interface, the buttons and lights, will be controlled by the server, but the Etherphone must transmit and receive event notifications.

Etherphone Server -- The Etherphone Server is the system controller. It is in charge of monitoring the state of the system, keeping track of the state of each Etherphone and what conversations are in progress, and of controlling the system. It is responsible for setting up all connections. In order to achieve high reliability we plan eventually to use redundant Etherphone Servers.

Voice File Server -- The voice file server is a general purpose computer with high capacity disks. It performs more or less standard file server functions, but is specialized for the real-time needs of telephony. The voice file server must reliably handle several simultaneous file stores and retrieves at the telephone data rate of 64 Kilobits per second.

POTS gateway -- "Plain Old Telephone Service" gateway refers to a server machine that provides access from the Etherphone world to the public switched telephone network. Calls arriving from the outside arrive at the gateway and are routed under control of the Etherphone Server to the appropriate Etherphone. Calls originating on the Ethernet but bound elsewhere use the gateway as a path to the outside world.

The next two system components are slightly different in character; we need them to provide a complete system, but in some sense we do not have to do all the work ourselves. These are not *new* components.

Workstation -- We will depend in large measure on workstations for good user interfaces.

Database -- We plan to use existing and planned standard file servers and data base services for storage of white pages and yellow pages information and for storage of users' call filters and other information. We may use data base services for storage of the voice file server *directory*, although

not for the voice files themselves.

COMPATIBILITY

We have no immediate plans to build the POTS server. Instead, we plan to retain the present system of individual phone lines. Rather than connecting them to standard telephone sets, we will connect tham as a "back-door" to the individual Etherphones. Calls for a particular station might arrive either over the Ethernet or over the back-door standard phone line. If a user dials an outside number, the Etherphone Server will direct the Etherphone to use the back-door line rather than the Ethernet.

We are taking this approach (which may be considered a *distributed POTS gateway*) largely for compatibility with the existing PARC phone system. If we removed all the existing lines and instead aquired direct-inward -dialling trunks for connection to a POTS server, we could no long be part of our Centrex phone system. Those without Etherphones would have to call us as "outside calls".

In addition, this organization offers additional protection against system failures. We will provide a deadman timer to automatically reconnect the outside line in the event of Etherphone or system failure.

COMPONENT OVERVIEW

We now turn to overall descriptions of the various system components. More detailed descriptions will be found in the various design documents, as they appear.

Etherphones

We plan to produce three or four versions of the Etherphone.

Etherphone 0

The Etherphone 0 exists now. It consists of an Alto I together with an Auburn audio board and a Danray telephone set. The program is written in BCPL and incorporates a first Ethernet voice transmission protocol together with a simple connection mechanism. One can "dial" the destination station’s Ethernet address and the program will ring the destination phone or return a busy signal. The program includes silence detection and a number of facilities for performance evaluation.

Etherphone I

The Etherphone I will use essentially the same hardware as the Etherphone 0, with the addition of a "back-door" interface to the office phone line. The Etherphone I program will also be BCPL, but should be simpler (more refined) than the existing program. We plan to let the Etherphone Server control the collection of Etherphone I’s.

Etherphone II A and II B

The Etherphone II series will be the first *real* Etherphones. Etherphone II will be a microcomputer system with its power supply in a shoebox on the floor and its telephone set on the desk. The "B" model would include a keyboard and a small display for telephone applications without a nearby

workstation. The Etherphone II will be built using off the shelf LSI components. It will be programmed in assembly language or in a higher level language such as C or Pascal. The program should be a near transliteration of the Etherphone I program.

Etherphone III

After we gain sufficient operational experience with the Etherphone II, and as available LSI parts improve, we may build newer, smaller Etherphones. We hope the Etherphone III will include a Dragon and be programmed in Mesa.

Etherphone Server

The Etherphone Server will be responsible for management and control of the entire voice system. The individual Etherphones will act as peripherals of the server; a users' actions of pushing buttons on an Etherphone will be transmitted to the Etherphone Server for interpretation. However, after the server directs two Etherphones to establish a connection, the actual two-way transmission of voice will proceed without further intervention by the server. We are taking this centralized approach because of the relatively low powered Etherphones and because we see a medium term need for some centralized management of the system. In the longer term, ther server functions should be assignable to one or two Etherphones in small systems.

Since the Etherphone Server will be responsible for interpreting users' actions, it is the logical place for the software controlling many system functions such as forwarding, call filtering, and control of the Voice File Server.

We will use a Dolphin running Pilot/Cedar for the Etherphone Server. We have tried to avoid time critical tasks for the Etherphone Server, so it should be possible to take advantage of some of Cedar's facilities. The Etherphone Server program (Thrush) design is further described in a later section.

Voice File Server

The Voice File Server will be controlled by the Etherphone Server. Storage of voice in real time is a sufficiently specialized activity that we feel no existing file server can fill our needs. The voice file server will have to speak the same protocol as do the Etherphones, and it will have to play and record several simultaneous voice streams at 64 Kilobits per second each. No special voice hardware is needed, because the voice will have already been digitized on its way to the Ethernet. Large capacity disks, however, will be important.

We will use a Dolphin running Pilot for the VFS. Use of Cedar facilities might compromise the real-time requirements, but we intend to take advantage of the improving programming environment.

REMAINING ISSUES

Ethernet Transmission

While we do not fully understand the details of Ethernet behavior in a very high load regime, we are very confident that the network behaves very well (low delay) up to sufficient load to build a usable Etherphone system. We intend to incorporate load management into the system to insure that the Ether does not become overloaded. We think that a 10 MBit Ethernet would handle

around 400 telephones, that a 3 MBit Ethernet would handle around 150 telephones, and that a 1.5 MBit Ethernet would handle about 80 telephones.

More measurements and experiments are probably required to pin these matters down precisely. There is interest in SDD (Bob Printis & Yogen Dalal), in GSL (Bernard Huberman), in the Science Center (Shoch) and in CSL (Stewart, Baskett, Swinehart, Nowicki, Gonsalves) in various experimental and theoretical efforts.

In the appendix there is a document "Voice Transmission Protocol Issues" by Nowicki and Stewart which includes some discussion of these matters.

Telephone Systems

All telephones are never in use at the same time. One very useful figure is that only about 22% of telephones are busy during busiest hour of the day. (This number via Lynch, we should locate supporting data. Can we find out how many calls are inside vs. outside?)

Long delay telephone circuits sometimes have trouble with echos. Echos can be caused, for example, by acoustic echo from the far end of a call or by an imperfectly balanced hybrid at a junction between 2 and 4 wire circuits. The Etherphone system, since it has "high delay" connections by industry standards, may require echo handling circuits on some outside calls. Single chip echo cancellers are slowly becoming available. We have a collection of literature and we may have to invest in experiment and hardware at some point.

Compression

While compression does not seem particularly attractive for transmission, it may be attractive for storage of voice. We have no plans to use compression. 64,000 bits/second is 8,000 bytes per second, or roughly 4 IFS pages per second. A single T-300 disk drive can accomodate 9.5 *hours* of speech. We think that will hold us for a while. If voice messages really take off, however, 9.5 hours is still only 11 minutes for each of 50 CSL members.

Conference Calls

We don’t really know the right way to handle conference calls. If we use multicasting, then when more than one person is speaking the involved Etherphones must mix the audio or decide who should be allowed to talk. Another approach is that used by most PBXs. We could build a few "conference bridges" and mix the audio and send it back out on the net.

Traffic management

In the appendix there is a memo "Voice Vs. Data" by Stewart which generally raises the spectre of class-of-service in datagram networks. If we are ever to use internet voice (imagine a collection of 10 MBit Ethernets hooked together by 1.5 MBit point to point links), then the gateways must allocate guaranteed bandwidth to voice users. Similar problems apply to combined voice and data on a single net or to use of an Ethernet as a transit network.

System reliability

To a degree, we have not signed up to build something as reliable as the exisitng phone system. We are using commercial power and each Etherphone retains an outside phone line for use when the system or Etherphone breaks. We are centralizing control in the Etherphone Server. What if it breaks? (In fact we will probably have to allow multiple Etherphone servers anyway, in order to allow internet Etherphone calls between areas served by different Etherphone servers.

Alternative Architectures

There is some material in the appendices about alternatives to the single line Etherphone. We think that the advanced functions could be built on a variety of voice transmission systems, but we think that the single line Etherphone is the most appropriate *for us*.

INITIAL PLANS AND SCHEDULE

By this Fall we plan to have the Alto I Etherphone I prototypes (2 to 5 of them) able to talk to each other and to standard phone lines with the aid of a first Etherphone Server. We plan to have essentially frozen the Ethernet voice transmission protocol and to have collected information about (and perhaps measured) Ethernet performance for voice traffic. We are already collecting information we need to start the Etherphone II design.

By this Winter we will have more of the basic Etherphone Server operational as well as a basic Voice File Server. We expect some preliminary applications work to start in parallel with these activities.

By next Summer we expect to have a number of Etherphone II prototypes and some solid applications.

Shortly thereafter we intend to build enough Etherphone IIs to supply all of CSL.

As was mentioned, we have no particular plans at this time to build the POTS gateway.

PERSONNEL, PLANS, FACILITIES, AND BUDGET

Dan Swinehart, Larry Stewart, and Severo Ornstein are the primary participants. They are spending roughly 70% of their time on the Voice project. In addition John Ousterhout and Susan Owicki are working on the project - John, one day a week, and Susan, one day a week for now and more after mid-October.

Specific roles are as follows:

Dan:	Define protocols; work on Etherphone Server program
Larry:	Work on Etherphone hardware and software
Severo:	Manage things and work on Etherphone hardware with Larry
John:	Work on Voice File Server
Susan:	Work on Etherphone Server design with Dan and File Server design with John

We are presently designing and will soon begin implementing both hardware and software.

Dan and Susan will develop the Etherphone Server code on Dan's Dolphin. If we can't find another machine by the time we're ready to start incorporating the Etherphone Server with the Etherphones, we'll use Dan's Dolphin as the Server - and get him an Alto II.

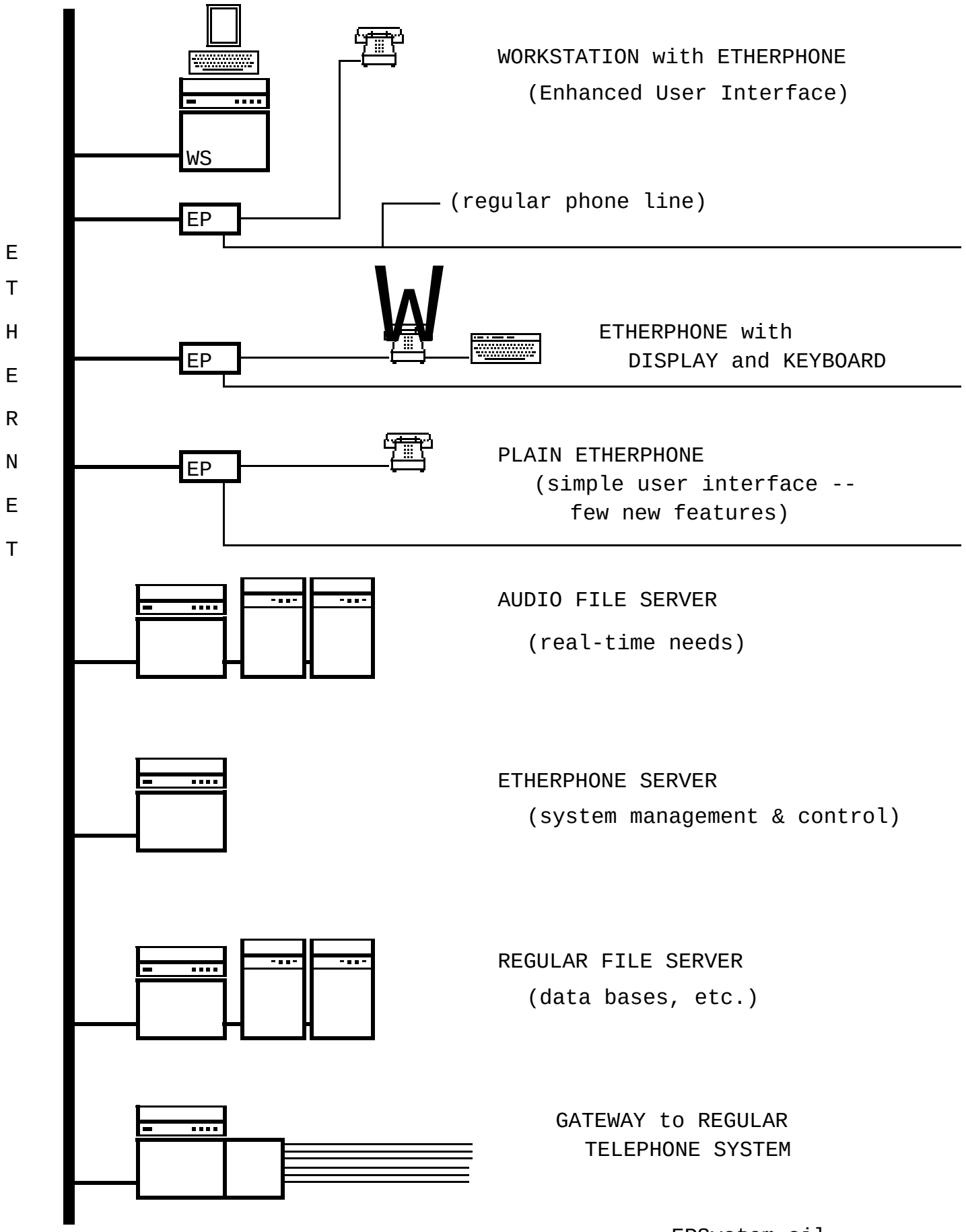
John will develop the Voice File Server code on a Dorado. (He has special sign-up dispensation for his one day a week here). Larry Stewart has a Dolphin with the understanding that if, by the time we are ready to start incorporating the Voice File Server with the other machines, we can't find another one for the server, we will use his machine - and replace it with an Alto II.

We have ordered a number of parts for the Etherphone II and are beginning to try out some small circuits to become familiar with how the parts work. We hope to have a working breadboard by about the end of the year. Most of our hardware work will consist of plugging together small numbers of reasonably high level standard parts on prototype boards. A lab bench or table will do for this. Also we'll need some power supplies and scopes and things but we don't forsee need for *substantial* lab space. On the other hand, during development, we will want to assemble all of the elements of the distributed system in one room, to make debugging feasible. When fullblown this could include the following items of equipment:

1 or 2 Alto based Etherphone I's
1 or 2 prototype Etherphone II's
1 Dolphin Voice File Server
1 T-80 for Voice File Server
1 Dolphin Etherphone Server
1 control ("Midas")* Alto for Etherphone II's
1 3MB to 1.5 MB Ethernet Alto Gateway (could be elsewhere)
Scopes, tables, etc.

* Not really Midas, but serves some of the same basic functions

Not a trivial pile of stuff. We plan to use the Nursery. It's about the right size and near the lab. (Our use fits the mold of the Nursery's original purpose). Over the next few weeks it will be cleared of other occupants and we will start to move in - at first with the Alto I Etherphone 0's (presently in Dan's office and in the Nursery) together with our hardware breadboarding. Gradually the other stuff will assemble there. We hope to have a rudimentary combined system working next spring. By next summer we expect to have the Etherphone II design solidified enough to order enough units to outfit everyone in CSL with an Etherphone. We have allocated $50K in the 1982 CSL budget for this purpose (not including server machines and disk drives). By the end of 1982 we hope to have a working backbone system in place in CSL that will handle telephone calls in standard ways, permit simple storage and retrieval of voice messages, and generally put us in a position to start experimenting with more esoteric uses of the voice handling facilities - e.g. integration with Laurel (or derivative).



EPSystem.sil

L. Stewart 9/13/81

Inter-Office Memorandum

To	File	Date	September 13, 1981
From	S. M. Ornstein, L. Stewart	Location	Palo Alto
Subject	Voice Proposal: Etherphone	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPEtherphone.bravo

ETHERPHONE II

The Etherphone II consists of two (perhaps three) physical pieces:

1. A telephone unit

2. A microprocessor unit

Note 1. As with a traditional phone, there may also be an external speaker and microphone; for simplicity we will ignore these here.

Note 2. We sometimes refer to the microprocessor unit itself as the Etherphone.

The telephone sits on the desk; the microprocessor unit fits in a shoebox underneath the desk.

The microprocessor unit can be logically separated into two parts:

Part 1 interfaces directly to the phone and to the regular phone line and has to do with audio switching (handset - speaker - mike), analog/digital conversion, reading the keypad, etc.

Part 2 consists of all the digital parts - interface to the Ethernet, the microprocessor itself, packetizing and de-packetizing the digitized voice, encryption, etc.

Bob Belleville is currently building hardware that corresponds to Part 1 and we are hoping simply to use that. His device will sit directly underneath the telephone and interface to it. (It is the possible third physical piece mentioned above.) If we copy his hardware directly, packaging and all, we will end up with some unnecessary things (most notably a tiny tape recorder). However it would save us considerable work. If we can't use his device directly (not ready in time or not suitable), we will at least be able to copy relevant parts of his design into our microprocessor unit. For the present we are planning to use his device. [See the Appendix.]

HARDWARE

Figure EPBlock.sil shows a block diagram of the Etherphone II. In order to get under way, we decided not to wait for either Alan Bell's or Intel's or AMD/Mostek's proposed 10 Mbit Ethernet controllers. Instead we decided to use the 1.5 Mbit SLC chip designed by the MEC for use in the Lotus copier program. We understand that this chip can run faster than it's 1.5 Mbit advertised speed. We understand that the limiting factor in the design has been fixed, and that 3 Mbit chips will be available soon. We have in hand a number of the 1.5 Mbit chips and are planning to put up a Gateway between the Parc (3 MB) Ethernet and a special 1.5 Mbit local Ethernet. (Apparently a single jumper on the Alto backpanel can halve the speed of one of its Ethernet controllers). If the 3 Mbit chips become a reality before we're ready to go, we will forget the Gateway. A diagram may be found as Figure Etherphone3.sil.

Our choice of the SLC biased our choice of microprocessor. The SLC was designed to talk to an Intel 8085 bus. We might, therefore, simply have chosen to use an 8085, an 8 bit machine. But there is a faster, more modern machine, the 8088 which internally is a (16 bit) 8086 but externally (its bus) looks (approximately) like the 8 bit 8085. We might have used an 8086 but did not feel that the additional potential bus bandwidth (16 bit bus vs. 8 bit bus means twice as fast) was worth the additional complexity in bus hardware. The 5 MHz 8088 completes a bus cycle in 4 clock cycles, a byte every 800 ns, which gives 10 Mbit/s bus bandwidth. That is insufficient for the 10 Mbit Ethernet, but by the time there are 10 Mbit Ethernet controllers, there will be faster versions of the 8088. There is, in fact, a planned 8 MHz version of the 8088 although most of the peripheral chips won't run that fast. Alternatively, we could go to the 8086 16 bit bus and still run the same software.

The bandwidth argument goes as follows:

> Digital voice will be at 64 Kilobits (8 Kilobytes) per second. A typical sample (from the person speaking) traverses the bus four times:
>
> 1. coming from the phone A/D into memory
> 2. from memory to encrypter
> 3. from encrypter back to memory
> 4. from memory to the outgoing ethernet
>
> In the worst case, both members of a phone call may talk at once. So the bus must service 8 x 64 Kilobits = 256 Kilobits per second. Of course this is *average* speed and we haven't included the processor's use of the bus to run the program or any other Ethernet traffic (control packets, conferencing . . .). However, even for a 5 MHz machine, 256 KBits seems like a small part of its bus bandwidth.

The SLC includes its own quite nice channel logic. It will use a DMA channel only to multiplex its bus access requests with those of the other devices; other than that it doesn't need any DMA facilities since it handles the bus itself. The SLC has independent DMA controllers for transmit and receive. Each controller interprets a chain of control blocks in memory.

We are planning to use the Advanced Micro Devices Z8068 encrypt/decrypt chip. This chip can run very fast, so we will have to slow it down to prevent it from hogging the bus. It will operate through use of two DMA channels, one for input and one for output.

We plan to use the Intel 8237 DMA part (which has a convenient "autoinitialize" feature that restarts data transfer at the beginning of a buffer when the buffer fills/empties). Each DMA part offers 4 independent channels and the parts can be cascaded using one of the channels. Two chips thus provide 7 channels.

Digitized voice arrives from (and/or goes to) the codec at one 8-bit sample every 125 ms. We will use DMA channels to get it into the memory. In fact we are thinking of using *two* channels for each direction. We'll assign two buffers to two DMA channels for (say) codec input. When the first buffer fills from the codec, the hardware will swap channels and the codec will start filling the second buffer. The first will be autoinitialized in preparation for refilling from its beginning, but will remain inactive (until after the second has finished filling). Meanwhile an interrupt causes the program to deal with the first buffer (passing it through the much faster encryption chip before the codec channel swap again needs the first buffer).

So the 8 DMA channels (two chips) we expect to use are:

> Codec In A
> Codec In B
> Codec Out A
> Codec Out B
> To Encrypter

From Encrypter
SLC (DMA cascade for bus access request only)
DMA cascade (for second DMA chip)

In addition to a small amount of bootstrap ROM we expect to utilize about 4 K bytes of ROM (RAM during development) for program and 4 K bytes of RAM for buffers. Our present intentions are to use mostly RAM and to include sufficient ROM for an Etherboot loader and perhaps teleswat functions. We are planning to use only static memories.

We intend to have two RS-232 interfaces (there is a convenient dual channel compatible chip). One channel will be for controlling the audio-handling hardware (Belleville's hardware). The other is for communicating with a mother ("Midas") Alto at first, and later to communicate with the keyboard/display of the Etherphone II B. We plan to download and interact with the Etherphone via the Ethernet after the kernel is working.

We will use the Intel 8259 interrupt controller which includes priority logic and interrupt vectoring.

We have in hand an SDK-86 (borrowed from Belleville) which is a small Intel 8086 prototyping kit including a monitor in ROM. We are able to speak to the 8086 from an Alto via DLS and an RS-232 interface included on the board. The ROM monitor on the board knows how to display and modify memory and registers, how to set breakpoints (hexadecimal), and how to load a program in Intel hexadecimal absolute loader format.

We plan to experiment with the more complex peripheral chips by wiring them up on a wire-wrap board and connecting them to the 8086 with ribbon cable to a wire-wrap Augat board. (The SDK-86 includes bus drivers for an external bus.) Our plan is to try to replace the 8086 with an 8088 while retaining the bootstrap/monitor ROMS. Once we understand the operation of the DMA parts, the SLC, and perhaps the Encryption chip, we will build a complete 8088 system on a Dorado stitchweld board. Mike Overton has completed a chassis holding a single Dorado board. The reason for the two level prototyping structure is to avoid too many reweldings of the stichweld board.

We have not selected a keyboard/display for the Etherphone IIB. We may try to obtain or borrow a Sunrise for the task.

SOFTWARE

We plan to use the 8086 assembler, linker/loader, and C compiler developed by Bill Duvall for Bob Belleville's Cub project. The assembler is free, but the C compiler is not owned by Xerox, although several groups (Belleville, Webster) are using it. A license will be $1000, including some degree of support. The compiler is not quite Unix version 7 compatible, it lacks things like bit fields in structures.

Since our debugging will have to be at the level of assembler listings, hexadecimal breakpoints, and load maps, we will try to keep the program in the Etherphone very simple.

Some portions of the grant universities' C Pup package may be useful for the Etherphone software. Bcpl to C transliteration is straightforward and we already have a working bcpl Etherphone 0 program on the Alto.

SUPPORT

We have a friendly Intel salesman, who is providing samples for most of the stuff we need and supplying us with some very useful application notes and gotcha sheets on the various parts.

Bob Belleville has been very helpful in lending us the SDK-86, supplying us with all the useful parts of his Cub software, and to some extent, in incorporating some of our needs into his thinking

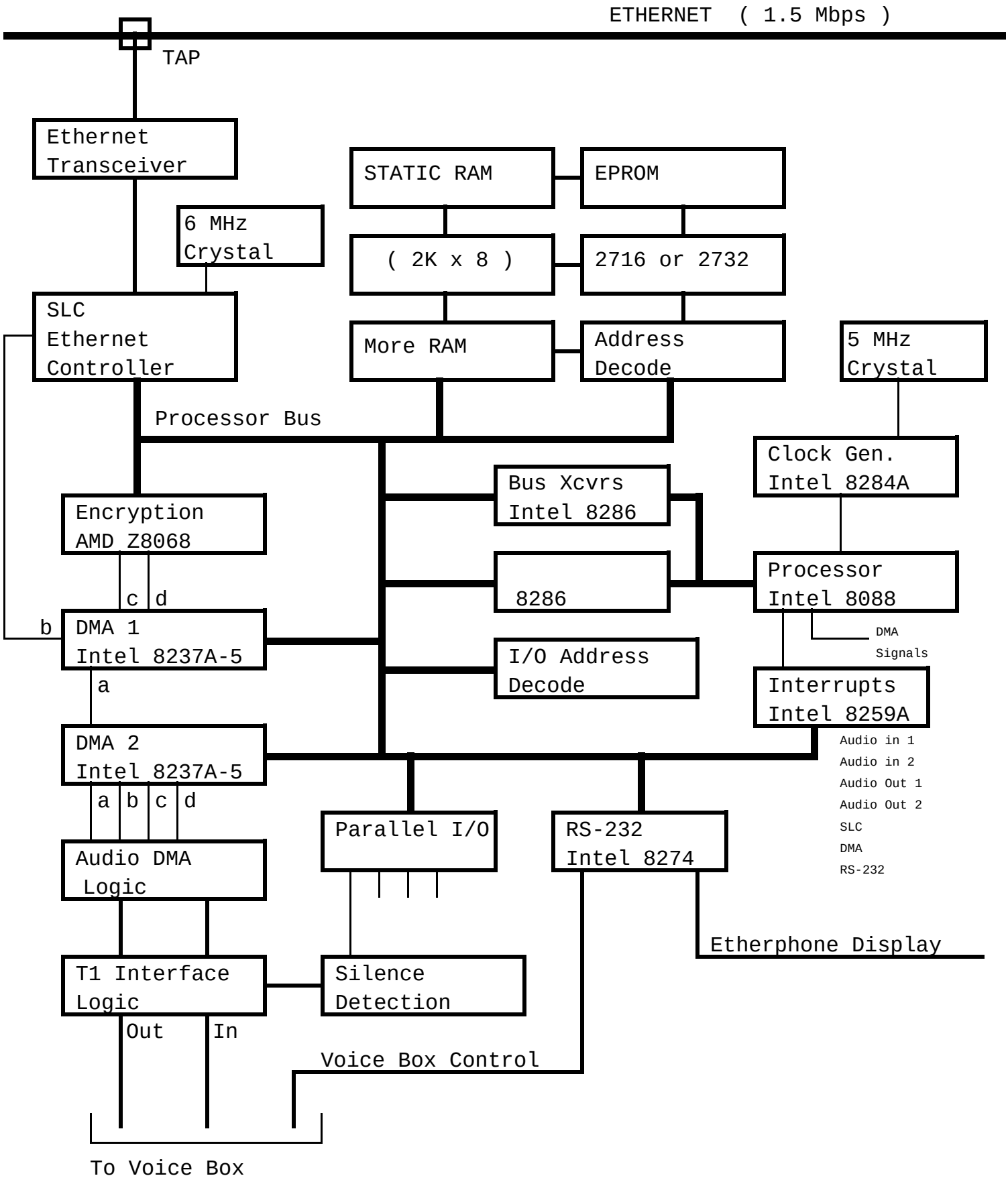
for the "voice box."

We have had good conversations with Simon Lau, an engineer working for Belleville on the voice box.

We (finally) located the right people at MEC and have learned enough about the SLC to be confident that we can make it work. In particular we now understand the assumptions that the SLC makes about the design of the rest of the microprocessor system.

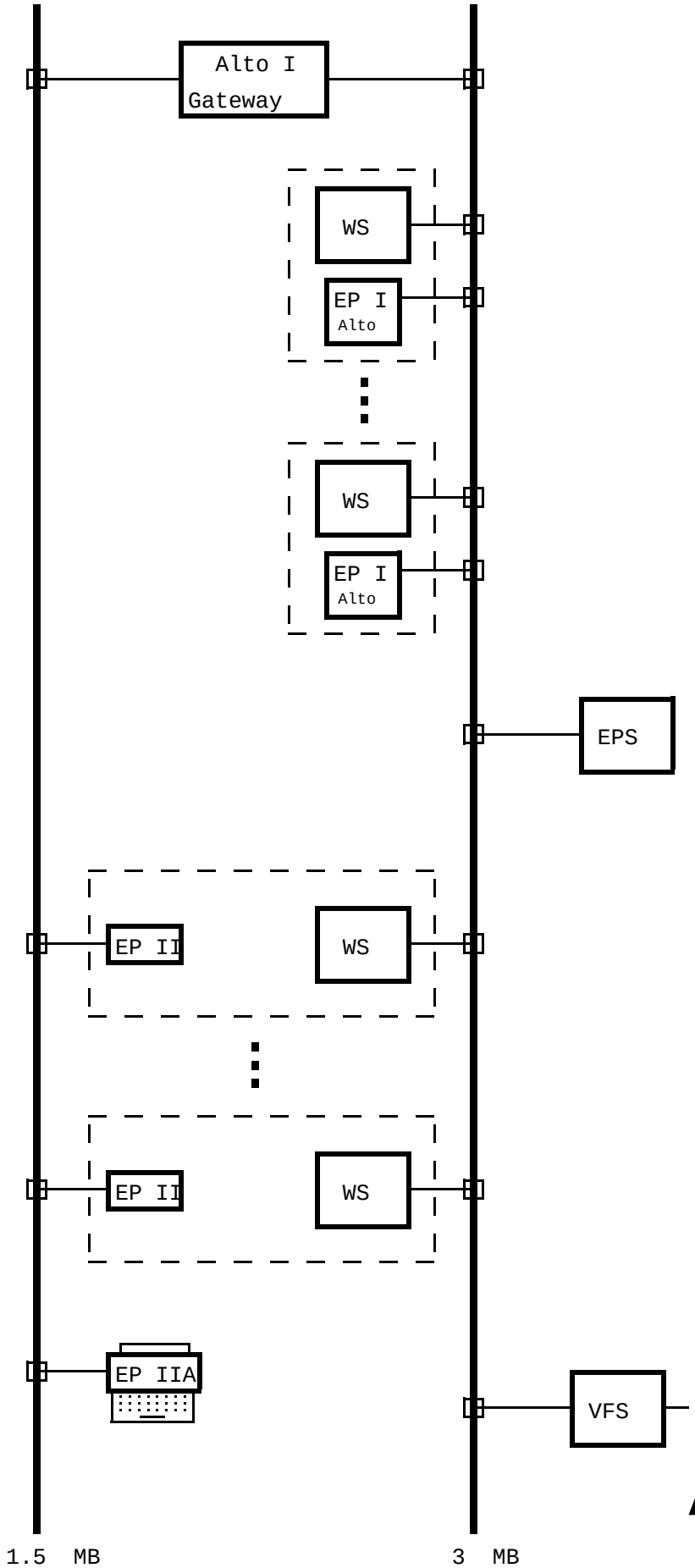
Bill Duvall has an office in building 96 and will be close enough to help with development tool problems, at least for a while.

Dave Boggs has agreed to help in setting up a 3 Mbit to 1.5 MBit Ethernet gateway, should we need one.



General Idea for Etherphone II
Omitting various glue chips

EPBlock2.sil
L. Stewart 9/13/81



ASDF

asdf

Etherphone3.sil

Inter-Office Memorandum

To	Voice Project	Date	August 31, 1981
From	Dan Swinehart	Location	Palo Alto
Subject	Control Protocols for the Etherphone System	Organization	CSL

XEROX

Filed on: [Ivy]<Audio>Docs>Thrush.memo

Introduction

This document represents the protocol design as of mid-August, 1981. Subsequent developments will appear in later versions.

This section describes the conceptual structure of the Etherphone system (the architecture as viewed by the software.) It begins by describing the basic entities in the system and the nature of their interaction. As much as possible, this design is independent of the assignment of these entities to hardware components and the actual communication paths that are employed; we envision a redistribution of responsibilities once we have gained some experience.

Given the structure, we will identify a number of important interfaces, and describe the functionality of the system in terms of the operations that these interfaces supply. Finally, there is a discussion of the intended assignment of functions to machines and of the low-level communications mechanisms that will be used to implement the interfaces.

Goals

A review of some of the goals of this project will help put the design in perspective. An overriding goal is to produce a telephone (etc.) system that we can have full control over, and in the manner to which we have become accustomed. This means that it must be possible for a programmer to write an application that will run on a workstation in our internetwork, with as much access to and control over our voice capabilities as it needs. Of course, we should attempt to present these capabilities to such a programmer in as uncluttered a form as we can manage.

On the other hand, we intend to replace existing telephones with our gadgets. This places availability and reliability constraints on the design that can only be met of if the phone system can operate independently from individual workstations. We have described in other sections the hardware ramifications of this requirement. Both those hardware decisions and the realities that led to them also have an effect on the software architecture.

In summary, then, our system will have *clients* as well as *users*, and we have to deal with that.

Basic Entities

We have found it convenient to think and talk about the software architecture of this system using the object style typified by <Smalltalk, etc.>. In this case, there is a small number of basic kinds of objects, or classes, instances of which communicate with other objects of their own and different classes. There may be but one instance of some classes, but in general there are many. Some of the classes are further specialized into subclasses which deal with variations of whatever basic themes their classes represent. For most purposes, it is reasonable to think of objects as Mesa

program instances (global frames) that communicate with other objects by calling procedures in the interfaces supplied by those program instances.

To motivate the choice of object classes in the current design, consider the following, relatively disjoint, kinds of activities and responsibilities:

> 1. The basic purpose of the system is to provide a variety of telephone and other voice communications facilities. There must be components of the system which know about telephony:
>
> > Name to number to hardware-specific mappings;
> >
> > Interpretations of addresses in terms of the network connectivities required to establish conversations;
> >
> > The nature of the call placement, connection, and maintenance activity;
> >
> > The meaning and implementation of features like call forwarding, conferencing, call recording, retry, etc.;
> >
> > The novel negotiation and filtering techniques that we intend to experiment with;
> >
> > Traffic and load management and instrumentation.
>
> These are the kinds of capabilities one would expect to find in any computer model of a telephone network; they can be described, and in some sense implemented, without detailed knowledge of the specific hardware and interfaces that the user sees. These capabilities are also representative of the kinds of facility that the CSL voice project is interested in implementing.
>
> 2. The clients of this general telephony knowledge are the specific implementations of the applications and user interfaces (digital and analog) that make up the voice communications system: workstation applications (for individuals and attendants), stand-alone telephone behavior, voice file server implementations, etc. These applications must be provided with an interface to the telephony "model" on the one side, and with the specific hardware on the other. They will determine the external appearance of the system. We will implement some of these facilities as part of the voice project; other people will produce additional applications at this level.
>
> 3. We have decided to minimize the size and complexity of programs in the Etherphone processor, assigning to other server processor(s) the responsibility for interpreting its control inputs and deciding its sequence of control outputs. It is therefore necessary to define the interface that will be used by this remote intelligence to communicate with the primitive Etherphone capabilities. Eventually, there may be a number of implementation-dependent interfaces.

Based on this taxonomy and the reasons behind it, we have produced the conceptual system architecture depicted in Figure 1. The labelled items represent objects whose classes are suggested by their outline shapes. The labelled lines, of varying thickness, represent the interfaces that these objects present to their various clients.

Fones

The circular objects are instances of the class *Fone*. Each object represents an individual or other entity that is or could be a party in a voice conversation. The abbreviation below the horizontal bar in each Fone identifies that Fone's subclass. Thus there is a Fone[IND] for every individual (identified by Grapevine RName) within the system who has an active Etherphone. In addition, there is a Fone[TRK] for each available conventional telephone trunk connecting the Etherphone system to the public switched network, other PBX lines, etc. The figure depicts a third kind of Fone, Fone[REC], representing the voice file system's involvement in recording a particular conversation.

Net

The triangular object represents a collection of objects and activities that will be further elaborated as the design progresses. It is supposed to represent the capabilities for registering and creating new Fones and other more numerous objects, for managing various name/address databases, for observing and controlling overall network traffic, and in general for whatever other truly shared or centralized concepts need to be represented. In an internet with multiple servers, each server machine will need its own Net object, so even here we must allow for the various kinds of distribution and replication that will result.

Smarts

The square boxes depict objects that implement the actual applications, or "smarts", in the system. This is the trickiest object class to motivate or describe; a substantial portion of the rest of this memo is devoted to doing that. For now it suffices to state that there is a Smarts object, residing somewhere in the system, for every distinct application or user interface that provides components of or interacts with the voice services. Thus, both because there are some voice functions (e.g., audible ringing and providing actual voice converations, stand-alone call placement and reception) that only the Etherphone processor itself can do, there is a Smarts object (Smarts[EP_x] for various x) providing or supporting these functions. But because of our critical goal of allowing workstation participation, there is also a Smarts object (Smarts[WS_y]) to represent the workstation implementation. Similarly, other Smarts objects provide the "intelligence" for such facilities or functions as outside telephone lines (trunks) and voice recording facilities.

Etherphones

The rectangles named EP_1, EP[TRK_1], etc. in the picture, merely represent the actual implementations of the simple programs, residing on the Etherphone processors, that provide the basic hardware control and voice transmission. These EP objects by themselves do not interpret user actions or understand how to participate in telephone activity.

Protocols (Interfaces)

It is hard to ascribe any value to this assignment of functions to objects without an understanding of the interfaces between the objects. In fact, a realization of the various kinds of interfaces that would be suitable for this system preceded this particular choice of object classes. The Fone, Smarts, Net, and EP objects were originally designed to provide reasonable places to put these interfaces. A small number of refinements later, the objects themselves began to make a good deal of sense as a way to factor and structure the system.

The most important requirement was that workstation-based client programs could play an active role in the user’s telephone dealings by, in effect, programming in a very high level telephony language. Short of actually implementing a new language, it is of course simpler to provide a collection of procedures, operating on data types that capture the appropriate level of abstraction. In the Etherphone system this high-level client to phone system interface operates exclusively between Smarts objects, representing the client applications, and Fone objects, representing the telephony model. In the figure, these interfaces are labelled "C".

Similarly, another interface, distinct from the high level is needed to convey user actions from Etherphone processors to their remote intelligence, and vice versa. These communications paths are denoted by the label "P" in the figure.

Communications between the Fone objects and between Fones and the Net are required to set up, take down, and monitor the progress of telephone calls, dictation sessions, etc. The diagram indicates by connecting lines the objects that would have to interact in order for the user of EPhone1 to place and record a call to the user of EPhone2. These interfaces, labelled "F", are private to the Fone/Net implementations; Fone clients do not need to know what they are.

Wherever network communications are required these interfaces must be expressed in terms of protocols that deal properly with the communications problems while expressing the right semantics. It is our current intent to employ the Remote Procedure Call (RPC) methods being developed for Cedar. Thus all of the interfaces can be expressed as procedural interfaces, whether or not they span machine boundaries. This approach will require careful attention to the process structures of the machines that comprise the system. That work remains to be done.

One ! Many Mappings

A number of activities require the participation of both the workstation and an associated Etherphone. Examples include:

> Placing calls -- the workstation initiates the activity, but the Etherphone must perform the actual voice transmission. It may also be called upon to generate call progress tones, etc. In addition, it has to keep track of the user's switchhook.
>
> Receiving calls -- the workstation Smarts may choose to be involved in the filtering and information flow that accompanies an incoming call. In particular, for calls forwarded to a central position, the attendant's workstation will perform a crucial role. We will probably discover many other possibilities.
>
> Messages, transcription applications -- the workstation will of course be in control.

This multiple participation is indicated in the figure by the appearance of connections between a given Fone objects and a number of Smarts objects. Rather than anticipate all the interactions that might be desirable, we've settled on a simple scheme for managing this multiplicity. It will be very nice if it works.

In the Smarts ! Fone direction there's no problem: a Fone will accept or deny a request based on the current state of the system. Going the other direction, there is an ambiguity about which of the Smarts should handle each activity. The proposed scheme is to register each of the Smarts for a Fone in a list maintained by the Fone, and in a precedence order. Each entry but the last is permitted to handle or pass on each request; the last must be willing to handle all requests (at least by firmly rejecting them.) A Smarts that passes on a given request may still choose to take some application-dependent action based on the request (posting the caller's name on the screen, for instance.)

If two Smarts have the same priority, the requests will be issued to each simultaneously, and the earlier respondent will win. This allows for the appearance of an individual's telephone line in more than one location. Such an individual has one Fone, connected to a Smarts representing each instrument bearing his line (e.g., in his office and in his laboratory).

Possibly there will have to be a different priority ordering for different groups of Fone ! Smarts requests; preferably not.

The lab phone situation also introduces the need for a multiplicity of Fone connections from a single Smarts object. Such a telephone may represent a number of different individuals. All of this resembles what happens at more complex attendant locations, and still needs to be worked out.

Mini-Scenario

A later section (to be written) includes detailed scenarios with sample uses of most of the protocol design. Here is a simple, high-level description of how a simple "stand-alone" call might be placed, maintained, and terminated. Assume that the system of Figure 1 has been initialized and contains the objects pictured there:

Cohen, intending to call *Jones*, lifts the handset of her telephone. EP1 detects the action and

forwards the offhook indication to Smarts[EP_1]. The Smarts instructs EP1 to issue a dial tone. When *Cohen* pushes "4", EP1 forwards the "4" to its Smarts, which responds by ordering EP1 to silence. After "4977" has been entered and forwarded, EP1's Smarts asks Fone[IND_1] to place the call. The Fone consults the Net to obtain the RName "Jones.PA" matching the phone number "4977", and a handle for Fone[IND_2], the representative for Jones.PA. Negotiations between the Fones determine that Jones is in to callers, and they agree to set up the call. Fone[1], via Smarts[1], instructs EP1 to provide a "ringing" tone to Cohen. Fone[2], via Smarts[2], instructs EP2 to ring its telephone. When *Jones* lifts the receiver, Smarts[2], Fone[2], Fone[1], and Smarts[1] find out about it, in various ways. A set of socket numbers identifying the conversation is distributed to the EP's, and they converse. The Fones register the conversation with the Net, which uses the information to monitor Ethernet traffic.

As the conversation progresses, the Fone objects monitor each others' status. Each will terminate the conversation if it detects any uncorrectable anomaly in the other (e.g., no response to the query.) Normally, though, the conversation will be explicitly terminated when one party hangs up and the change in switchhook state progresses through the system.

Intial Architecture: Thrush and Etherphone 1

It seemed important (at the time) to describe this architecture in the absence of specific assignments of functions to machines. In fact, the architecture should survive a number of reassignments that we contemplate making over time. But we have of course chosen an initial system configuration; Figure 2 is an augmented version of Figure 1, depicting the proposed setup.

The large central box is the Etherphone server (its program is named *Thrush*). The Thrush server provides the entire implementation for Fone and Net objects -- the network model. In addition, it is the current site for the "Smarts" for the Etherphones. These Smarts provide stand-alone Etherphone functions as well as the ultimate interpretation of workstation requests that must be satisfied by the Etherphones.

As we have said, the initial system will not include a separate server connecting to outside telephone lines, or trunks; instead, each Etherphone will have a "back-door" connection to the existing Centrex telephone line for that office. However, to indicate that this connection is in principle entirely separate from the the local telephone instrument and the Ethernet connection, and will in fact eventually be concentrated in separate servers, we have explicitly separated them in the design, by providing independent Smarts and Fone objects for the back door connections. The Etherphone will also deal with them independently.

The Etherphone processors, will implement the EP objects, using RPC communications (in both directions) to obtain the wisdom of their Thrush-based Smarts. We will write Etherphone programs of this kind for both the initial Alto I Etherphones and for the later microprocessor-controlled systems.

The workstation-based smarts, reside, naturally enough, in the workstations. They comprise the realization of customized calling and answering capabilities, powerful attendant features for outside, unanswered calls, voice document annotation systems, and the like.

Provisions for multiple Thrush servers

If we are to have any confidence of achieving our reliability requirements (basic telephone service always available), we will need more than one Thrush server. What sounds best at present is a model similar to the Grapevine server model: there is a Thrush server at each campus (or perhaps on each Ethernet in a large campus), which serves as the primary server for Etherphones in its locale. Another server can provide service to an Etherphone when its primary server is broken (rejecting or not responding.) Handling conversations that take place between Etherphones with different primary servers will require the participation of both, to an extent and in a manner yet to be determined (agent Fones representing the other end in each server? Net to Net communication?

direct Fone to Fone communication via cleverly arranged Remote Procedure Call?)

When a server goes down, we will attempt to avoid terminating the conversations that it was managing. Instead, the Etherphones will frequently reassure themselves that their (Thrush-based) Smarts are still functioning, searching frantically for new ones if they are not. They will provide enough information in the process to allow the new (or resurrected) server to rebuild some sort of model of the ongoing conversation.

Specifics of Protocols

These aren't done. They aren't even right any more. They are representative of the kinds of things going on at each level.

Notation

We have decided to base our control protocols on the Remote Procedure Call (RPC) methods being developed for Cedar. The idea is that of a simple packet exchange, simulating a procedure call and its return. The packet exchange will comprise a pair of packets of the following sort:

```
PUP[code, ID (sequence #), sourceSocket, destSocket, data]
        RESPONSEPUP[matchingCode, same ID, reversedSockets, responseData]
```

In what follows, we will abbreviate this as a procedure call qualified by an indication of the source and destination objects, or by an interface name that implies the source and destination. We will use this notation whether or not network communications are involved. In the case of network communications, the source and destination values should be thought of as socket identifiers that will locate the objects to which the packages are addressed. Sometimes this socket interpretation will be explicitly indicated (e.g., as [net#host#socket#].)

Abbreviation:

```
<Source!Dest>.Code[data] RETURNS [responseData];

        or

Interface.Code[data] RETURNS[responseData]; -- familiar?
```

Smarts$*Fone Protocols (Client interface)*

<Smarts ! Fone> interface abbreviated as ToFone; <Fone ! Smarts> interface abbreviated as ToSmarts

ToFone.GetStatus[] RETURNS [status: {callInProgress, outOfService, idle, TBD}, filterInfo: TBD}];

ToFone.CallByRName[self: Smarts, name: RName, priority: {TBD, includes "normal"}]
RETURNS [{callInProgress, priorityTooLow, rejected,
busy (and not rejected), noAnswer}];

Smarts[EP]$*EP Protocols ("Hardware" interface)*

<Smarts ! EP> interface abbreviated as ToEP; <EP ! Smarts> interface abbreviated as FromEP (I know, I know)

FromEP.StillHere[!timeout, rejection=>-- time to reregister];

To be issued at intervals by EP, or when there's no response to some other query.

FromEP.RecordEvent[Event];

Event is an enumerated type containing {0, ..., 9, #, *, A, B, C, ..., onHook, offHook, ...};

ToEP.Reset[severity parameters?];

Cancel any tone sequence in progress. Clear the display. Forget about any conversation in progress. Hang up any automatically switched audio devices (Speakerphone, etc.) Forget the name of your Smarts, except when received as the first command after a GetSmarts (?????) Alternative: severity parameters indicate how much to forget. This is a catchall for various kinds of reset or abort functions.

ToEP.Tones[f1, f2, modulation: Hertz, on, off: Milliseconds,
repetitions: CARDINAL, mode: {ring, ringback, transmit}];

f1=f2 implies silence. modulation=0 implies that f1 and f2 specify sine waves to be added. Otherwise tones of the two frequencies alternate at the modulation rate. Tones occur in bursts whose duty cycle is deterined by on and off. Any Tones in effect are cancelled by a Reset[] or Converse[] request. Tones returns immediately. **TBD**: how to achieve a timed sequence of tone behavior (aside from Feep); require EP to queue requests, abortable only by Reset?.
mode:
ring -- tone to office speaker only -- annunciation
ringBack -- tone to office handset speaker only (or speakerphone/headset equiv.) -- call progress
transmit -- tone to office handset and transmission line (both parties) -- signalling

ToEP.Feep[on, off: Milliseconds, mode: {ring, ringback, transmit},
length: CARDINAL, number: PACKED ARRAY [0..0) OF Event];

Equivalent to a complex sequence of Tones requests resulting in the generation of a DTMF sequence, usually in transmit mode. Make/break intervals (on, off) parameterized, since one can usually push the TelCo specs so experimentation will be useful.

ToEP.Display[length: CARDINAL, number: PACKED ARRAY [0..0) OF Event];

EP is assumed to have a one line character display. This specifies the string to be presented there.

ToEP.ConverseWith[yourParty: Socket, otherParty: Socket, protocolType: {interactive,
recording}];

The protocolType field will allow us to behave differently towards the file server than towards other conversants. Smarts or its Fone will have to communicate further with the file server to obtain "time and charges" -- information about the call's duration and nature.

ToEP.GetStatus[yourParty: Socket, otherParty: Socket, protocolType: {interactive,
recording}];

The protocolType field will allow us to behave differently towards the file server than towards other conversants. Smarts or its Fone will have to communicate further with the file server to obtain "time and charges" -- information about the call's duration and nature.

Fone$*Fone, Fone*$*Net Protocols (Internal interfaces)*

(mostly TBD during Etherphone Server design and implementation -- none of these interfaces are visible to the Client (Smarts) or to the Etherphone implementations.)

<$Fone_i$! $Fone_j$> abbreviated as InterFone; <$Fone_i$! Net> abbreviated as ToNet

InterFone.CallRequest[callDescriptor: TBD (includes $Fone_i$ ident), priority: {TBD, includes
"normal"}]
RETURNS [{canBeDone, priorityTooLow, rejected, busy (and not
rejected)}];

InterFone.ConnectRequest[callDescriptor: TBD] RETURNS [{callInProgress, rejected, timedOut}];

ToNet.RegisterCall[callDescriptor: TBD, Fone$_j$: Fone] RETURNS [<two conversation socket values>];

"Sneak Paths" (Bootstrapping interfaces)

```
<x!Broadcast>.RoutingInfoRqst RETURNS [routingTable];
<x!Broadcast>.NameLookup[serviceName: STRING]
        RETURNS [list of [net#host#rtpSocket] tuples];
```

(interpretation of rtpSocket (rondezvous/termination protocol): a socket allowing a brief sneak path from the caller to the Net object in the Thrush server to generate some smarts)

There are more sophisticated schemes for linking up to services, etc., in the works as part of the RPC effort; we will watch them with abiding interest.

<EP!Net>.GetSmarts RETURNS [Sx: SmartsSocket];

Net is [ThrushHost#rtpSocket] obtained from NameLookup. Sx will be used in the sequel to describe the socket over which the EP and its Smarts communicate.

<Sx!Net>.GetFone[self: Smarts, rName: RName] RETURNS [Fone: FoneHandle];

This is just a procedure call within the server.

<Sx!Net>.GetRName[soc: Socket] RETURNS [rName: STRING];

Finds an RName associated with that host, in local data base. TBD: a Smarts!Fone!Net function to update this and other data bases.

Scenarios

When sufficient pieces of the protocol have been designed, it will be possible to render a detailed scenario of call placement and receipt under a variety of circumstances. We will use these scenarios to convince ourselves that we have enough bases covered to begin programming.

Figure 1. EPhone 1 calls EPhone 2

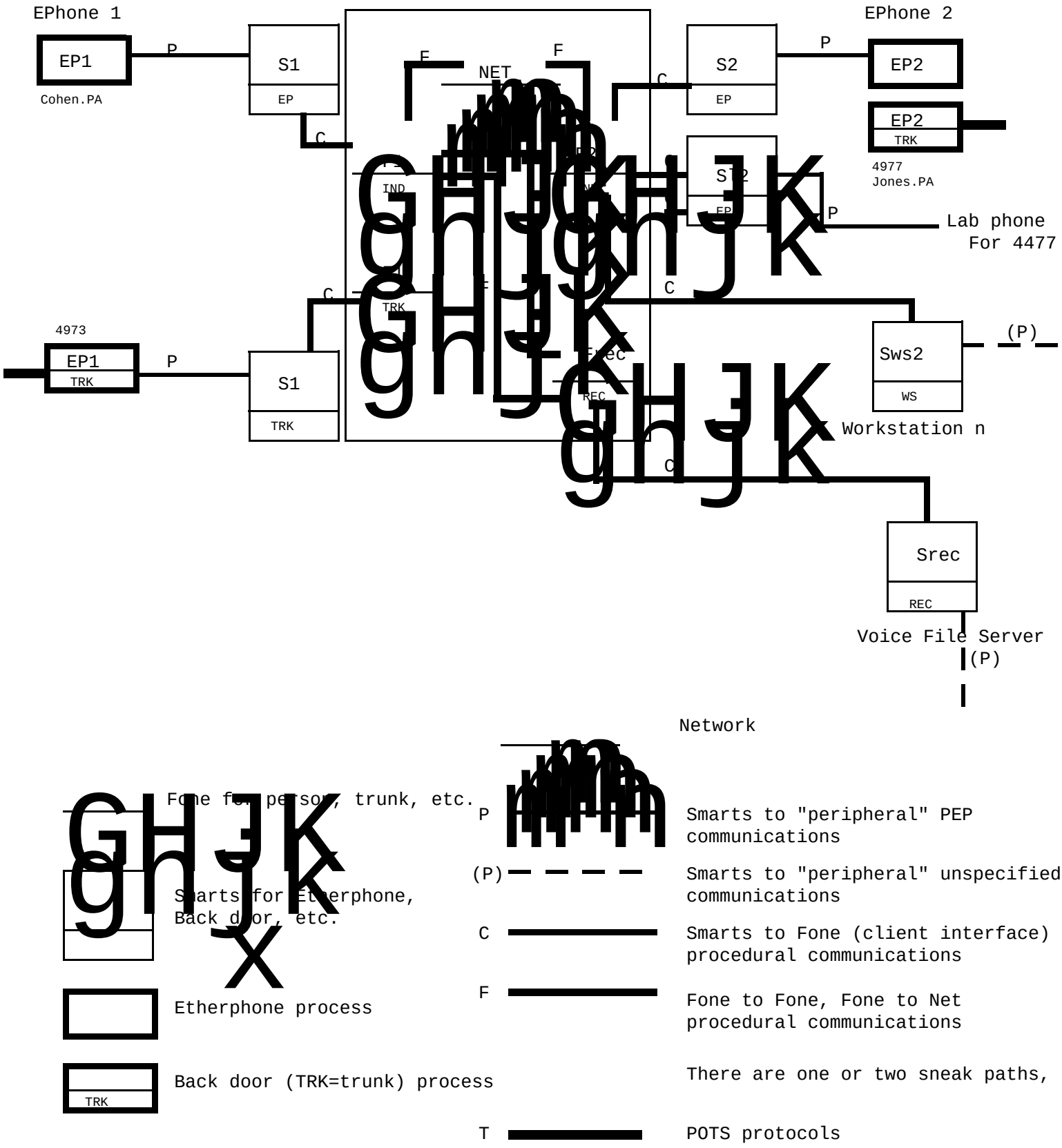
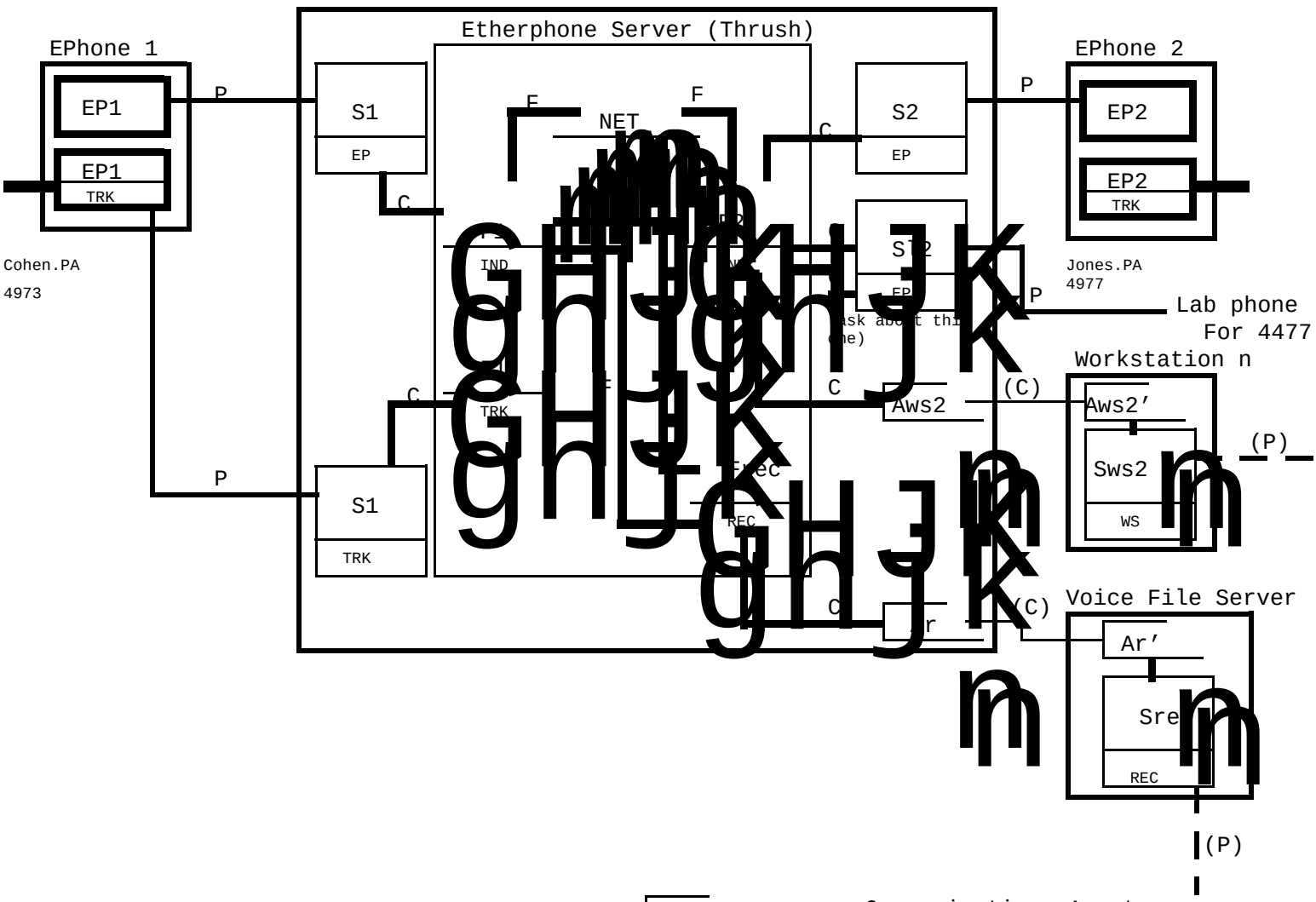


Figure 1. EPhone 1 calls EPhone 2

Figure 1. EPhone 1 calls EPhone 2



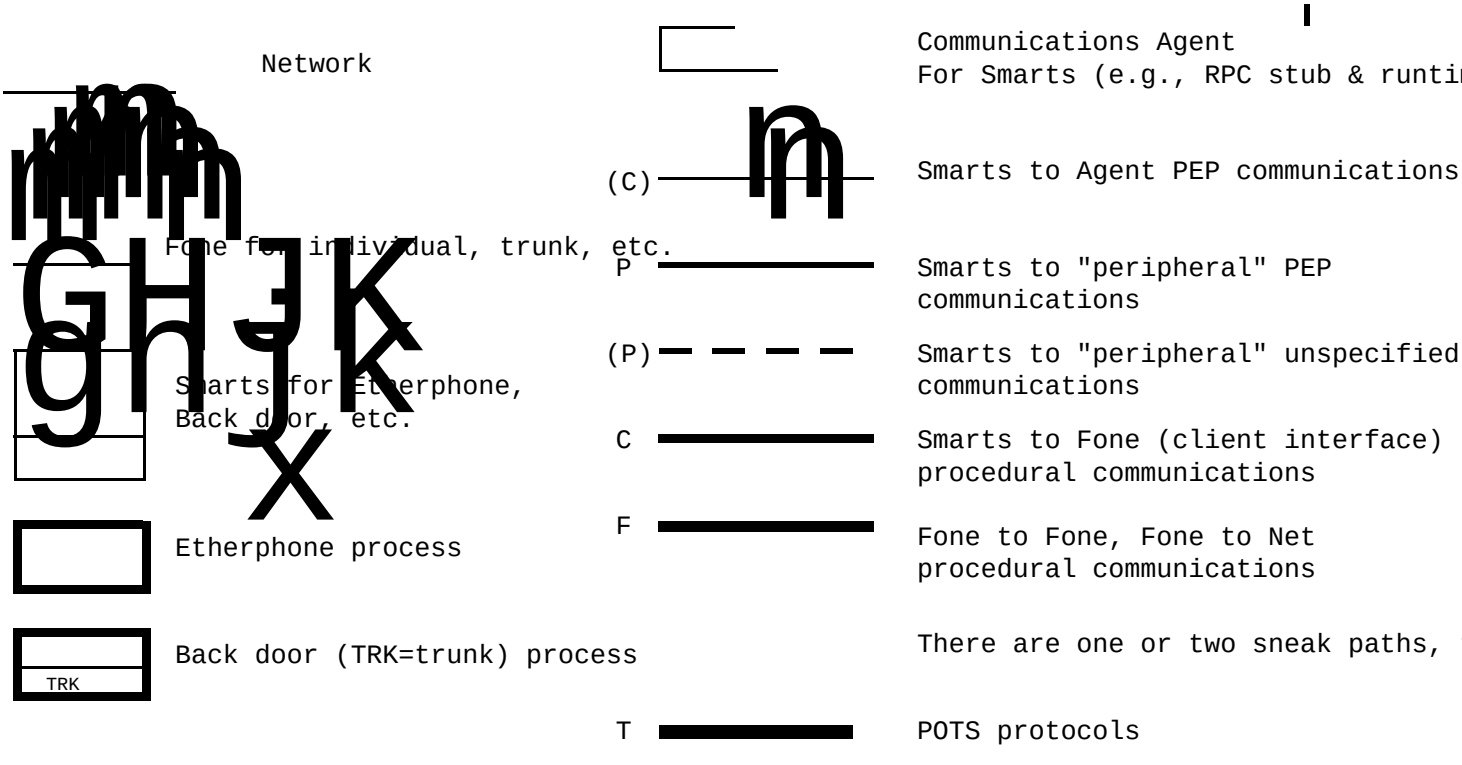


Figure 2. Assignment of functions to machines

Inter-Office Memorandum

To	File	Date	September 15, 1981
From	J. Ousterhout, L. Stewart	Location	Palo Alto
Subject	Voice File Server	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPVFS.bravo

Storage and retrieval of voice from a network file server will serve as the connection between the "real-time" voice world of Etherphones and the non-interactive world of voice messages.

FEASIBILITY

In raw horsepower, our existing collection of hardware provides adequate performance for a multi-user voice storage system. Nevertheless, we feel that it will be a challenge to provide for multiple simultaneous file stores and retrieves.

Precedents

A single Alto with model 31 disks and running the standard bcpl FTP can support a real-time voice file retrieval at 128 Kbps, using 8000 bytes of buffering at the receiver. The same Alto is incapable of *extending* a file at 16 Kbps, although writing into an existing file works well.

An unloaded IFS can be used for file storage at 64 Kbps and retrieval at 128 Kbps. Juniper can both store and retrieve at 64 Kbps although the initial delays are long.

A "voice File Server 0" program, by W. Nowicki, supports a single file store or retrieve on a Dorado using the file stream interface of Pilot.

PERFORMANCE REQUIREMENTS

Bandwidth

64 Kbps voice corresponds to 8000 bytes per second, or 16 Alto disk pages per second or 4 IFS pages per second. Providing two one second disk buffers per voice connection to a file server would permit (for contiguous files) use of only one disk seek per second per connection while using only 16,000 bytes of memory. 64 Kbps is not very fast.

Capacity

A single T-80 disk with Pilot formatting provides 113,680 512 byte pages for a total capacity of 58.2 Mbytes, corresponding to just over two hours of voice. A T-300 with 2048 byte sectors provides 253.7 Mbytes, corresponding to 8.8 hours of voice. While we have not tried to guess how voice messages may be used, such amounts seem adequate for the next year or so.

Timing

The voice file server must speak to the Etherphones. Because file storage and retrieval is less interactive than conversation, we feel that the voice transmission protocol used to communicate with the file server may reasonably involve higher delays in order to reduce the numbers of packets per second. (There may be a problem since we need to support the recording of conversations.)

When multiple actions are proceding at once, however, a very large number (several hundred) of packets per second may be involved.

During playback, the voice file server must maintain a sufficiently accurate clock to accurately meter outbound data to an Etherphone. The Etherphone does not have enough memory to handle uneven flow. One possibility would be for the Etherphone to occasionally (once a second) transmit the time to the server. Such a scheme would only require the file server to have a clock accurate to 50 milliseconds or so over the course of a second.

ENVIRONMENT

We believe that Pilot running on a Dolphin with Trident disks is the appropriate environment for the voice file server.

The reasons in favor of using Pilot are:

1. Alto is on the way out, Pilot is on the way in, we should go with the wave of the future rather than get undertowed by the past.

2. Pilot (Cascade) provides a better programming environment then Alto Mesa, so the VFS will come up faster.

3. The Pilot interface to the disk is better than the Alto one and provides good performance for large transfers. Hence we may avoid writing new disk management routines.

4. D machines provide much better performance than Altos -- performance will be important.

5. There is at least some chance that Alpine will provide suffient performance for a VFS. If not, we may still be able to borrow some components.

Reasons against using Pilot are:

1. There is not yet a Trident disk controller for Dolphins, so we will have to use Dorados until the Trident controller becomes available for Dolphins. Another alternative is to use the main disk of a Dolphin.

The consensus is that I should start working in Pilot/Dorado land, move with Cedar as it becomes available, and migrate to Dolphins when the controller is born.

IDEAS

The voice file server will provide services mediated by the Etherphone Server. The fundamental file server requirement is speed. We will put most of the control machinery elsewhere.

Piece Tables

The file server function must provide piece table like functions in order to permit editing of voice messages without copying of data.

Arguments for implementing piece tables directly:

- All real-time stuff should be in the VFS, no real-time stuff should be outside of it. If piece tables are outside, then they have to satisfy real-time constraints.

- If piece tables are outside, then storage management of the VFS must be at least partially outside, since pieces may have multiple references and cannot be recycled without knowing the (external) state of those references.

Arguments for implementing piece tables elsewhere:

- Inclusion of the piece tables in the VFS will complicate it, thereby delaying its operability and jeopardizing its ability to run in real time.

- If piece tables are managed externally, then we need not implement them at all for starters. When an implementation occurs, we have much more flexibility to play with alternatives.

Letting the real-time constraints of piece table management get outside the VFS is really no big deal: the bandwidth of this information will be 2-6 orders of magnitude less than the bandwidth of the actual audio, so external implementations should have no trouble keeping up with the VFS. In most cases, a single message can give all the pieces for an entire playback. The general consensus is that the 2nd alternative (no piece table in the VFS) is the way to go.

What about problems with too many too small pieces? If pieces get too fragmented then the VFS won't be able to play them back in real time. Furthermore, what constitutes "too fragmented" is dependent on the particular VFS implementation, and so should be decided by the VFS rather than some external party. Our proposal is that the VFS should provide a command CheckPiecesOk that will indicate if a given set of pieces can be played back in real time. Ostensibly this command would be used by editors: if the answer is "no way" then the editor can combine pieces to achieve real-time ability.

What is the minimum grain of pieces? Should pieces be specified as whole disk pages (substantial fractions of a second at a time), individual samples, or at some intermediate level? For encryption, units on the order of 64 bits at a time or larger have to be dealt with together. We propose that the start and stop times for pieces should be specified in 8-sample chunks. This turns out to be one millisecond's worth, and that has a nice ring to it. What kind of time resolution do real audio wizards need? If the VFS were used to edit the soundtrack for a film would 1 millisecond resolution be adequate? One possibility is that clients must require enough data to place pieces on reasonable boundaries (if they need the resolution).

We suspect that 10 to 50 millisecond resolution would be adequate.

High level interface

The high-level command interface for the VFS (i.e. that used between it and the Etherphone server) should provide something like the following calls:

- CreateFile: does the obvious thing, sets file's ref count to 1.

- ExpungeFile: may not be necessary, see reference count stuff below.

- Append: audio is supplied from some internet source and is appended to the end of the file. This operation and file deletion are the only ones that modify the contents of voice files. In particular, there are no insert or replace operations.

- Playback: a piecelist containing (file, extent) pairs is specified; the indicated audio is shipped to a sink somewhere in the internet.

- StopTransfer: causes an Append or Playback operation to terminate.

- Copy (?): simply appends a piece of one file onto the end of another file. This command could be simulated with an Append+Playback combination where both source and sink are the VFS. This may not be necessary, especially for starters.

- IncRefCount: used for storage management. Causes the VFS to increment the reference count for a given file.

- DecRefCount: Causes the VFS to decrement the reference count for a given file. When a reference count becomes zero, the VFS should feel free to reallocate its space. See note 5 below about alternate mechanism for reference counting.

- ChangeStopTime: see note 4 below.

- CheckPiecesOk: see note 6 below.

Additional maintenance-level commands may be needed, such as:

- Enumerate directory: get a list of file id's.

- Store and retireve via FTP (or some other "data" protocol). Clients other than Etherphones may want to store or retrieve voice files in a non-interactive way.

File Storage Model

It is assumed that the VFS will use some mechanism to avoid storing lots of zeros for silence. One technique: use a "magic" block number in file descriptors to indicate "all zeroes so no storage was allocated." The technique used in the Voice File Server 0 program is to store essentially images of the voice packets received, so that silence is not stored and every piece of the file carries sequence number information.

One possibility implementation of voice files is to store the Ether packets exactly as they arrive. This makes recording and playback easy, but makes it somewhat harder to find a particular time in a file (even binary search will take time: the effect will be to increase the minimum piece size)..

Reliability

What if the Etherphone server dies?? In order to avoid dangling recording sessions that eat up all available disk space, all Append and Playback operations should have timeouts. One possible scheme:

- On issuing each Append/Playback command, the EPS specifies an "autostop" time. If this time is reached, then the VFS will automatically terminate the Append/Playback. For Playback, the piece list implies a stop time anyway.

- To continue an Append, the EPS periodically issues ChangeStopTime messages to advance the autostop time farther into the future. In messages containing several pieces, the ChangeStopTime command can be combined with additional piece specifications.

Note that this also gives the EPS a modicum of control over storage allocation in the VFS, since it can interrogate the VFS to find out how much space is left and, if space is getting tight, determine who gets to use how much.

Page Level Reference Counts

An alternate scheme for reference counting is to place reference counts on segments of files rather than on whole files. If piece tables start getting used a lot then there will be many files sitting around with only one small piece in use. If reference counts are on a file basis, then the whole file will have to sit around wasting space. An alternative is to do reference counting on a piece basis, with the IncRefCount and DecRefCount commands referring to pieces rather than files. The VFS could then mark individual blocks. This doesn't appear to involve substantially more work for either party, but would allow blocks to be recycled individually (e.g. turn unused blocks to silence). This may be left out of the initial implementation since few piece tables will span multiple files. However, even if a piece table only spans part of one file, 90% of the file may still be unused.

We are still divided on this issue.

Inter-Office Memorandum

To	File	Date	September 13, 1981
From	L. Stewart	Location	Palo Alto
Subject	Voice Project: Alternatives	Organization	CSL

XEROX

Filed on: <Audio>Doc>VPAlternatives.bravo

ALTERNATIVE ARCHITECTURES

We have identified alternative system approaches to voice. These differ (in various ways) in the area of *transmission and switching* and in the area of *control.* We believe that very nearly the same collection of functions can be *provided* through any of these approaches, but that the choice greatly affects the performance and elegance of those functions. We should not forget that eventually a difference in degree becomes a difference in kind: a system that takes 1/2 second to place a call is very different from a system that takes 10 seconds to place a call.

One can separate three sub-areas of voice work: telephony, filing, and system integration. *Telephony* has to do with the transmission of voice data, with the elementary control functions of placing calls, and with terminal equipment (hardware). *It is in telephony that the architectures described below differ. Filing* has to do with storage and retrieval of voice messages and with composition and editting capabilities. *Integration* has to do with advanced control facilities such as use of a data base to store telephone numbers, and with the manipulation of voice data in cooperation with our other activities. It is important to note that filing and some kinds of voice work could proceed with a voice system that is *not* combined with the telephone system. We could concentrate on voice editting and annotation and ignore telephony, but we think that there is a lot to be gained by the integration of telephony and filing.

1. Use the existing telephone system

The present CSL phone system provides a single direct dial number for each office and has a few value-added features such as call-forwarding and an attendant console. (One can forward calls to another number and one's phone transfers to the attendent automatically after three rings.) The distinguishing features of this system are that control information is ''in-band'', consisting of beeps and clicks on the voice channel, and that both voice and control information passes through conventional phone wires and through a central switch (exchange). These remarks also apply to most PABXs.

To use the transmission and switching facilities of the existing phone system and to provide control over its operation, we would build a device which would connect to a workstation and permit the machine to pick up and dial one's office phone. In addition, we would need some number of voice I/O interfaces on server machines. The voice I/O device, in small numbers, would provide a way of getting voice on and off the Ethernet, where our programs can work with it, file it, and so on. The A/D and D/A conversions are provided by a *server*, rather than by one's own voice terminal.

This system provides the capability for a number of really impressive systems: voice messages, voice annotation of documents, semi-automatic call placement, and so on. There are also some crippling disadvantages. *Any* voice recording or playback would require placing a call to one of the servers!

Calls are placed by electrically picking up the office telephone and electronically generating beeps. This can be made to work, but is a slow and somewhat uncertain process. The progress of an attempted call is determined by the return of various noises over the voice path: ringing, dial-tone, busy, fast busy, etc. It is quite difficult for a machine to sort out these noises; they cannot be ignored because calls do not always get through. In addition, the placing of a call requires several seconds: one or two to dial the call, perhaps one for the system to connect a local call, and as many as six seconds for ringing to be detected at the destination. This means that for applications such as annotation of a document, one's office phone is effectively tied up since we could not afford the overhead of setting up a call for playback of a 10 second segment.

Many of the call placement and call receipt functions, for calls within the system, could be handled by internet communication, one's workstation could check if the callee's phone was in use before placing a call. There would still be a small "window" through which other calls could creep. Functions like complex forwarding would be quite difficult, since we could use only the switching capabilities built into the standard phone system.

The essential difficulty with use of a standard telephone system is the lack of sufficiently fast and versatile control of the switching machinery.

Belleville is essentially following this strategy. His voice box is an audio interface for a workstation. His device includes a tape recorder so that low power workstations can *control* the operation of the voice peripheral without handling voice data directly. Higher power workstations also have access to the digitized voice.

2. Control of a PABX

We could replace our present telephone system with a commercial PABX and use one of our own computers to control it. D/A and A/D conversion functions for manipulation of voice-as-data would be done by servers, and call placement would be be done either by manually dialing one's telephone or by having one's workstation instruct the control computer to set up a call.

In this system, the voice data still travels through more or less conventional wires and switches, but the control information is entirely digital, on the Ethernet, and under our control. Calls inside the PABX would be very fast and we would have easy access to the state of the system. It would be possible to connect to a server for just a few seconds to record a voice annotation, while still remaining available for incoming calls. Complex switching operations would be possible because the phone system would be entirely under our control. We would still require an adequate number of servers to meet our D/A and A/D needs.

The key disadvantages of this system are that we do not have such a controllable PABX and, before installing such a system, we would have to negotiate control of the switch with the vendor.

3. Build our own

If we build our own transmission and switching, we will clearly have sufficient control over it to supply the facilities we need. The possible approaches lie along the spectrum from constructing a more or less conventional telephone switch to a fully distributed architecture of telephones connected to a local network. In our environment, the local network of choice is the Ethernet -- we call this approach Etherphone. Somewhere in the middle of the spectrum are systems with multiple line controllers connecting 8 to 32 telephones each to the Ethernet (or interconnected with high speed lines).

One might call the three approaches to contruction of transmission and switching facilities the *centralized,* the *hierarchical,* and the *distributed*. Construction of a standard telephone switch lies largely outside our expertise. The hierarchical approach of shared Etherphones is described more fully later, but briefly it might be a bit less expensive in trade for a larger and more difficult design. For these reasons, we are concentrating on Etherphone.

Discussion

We feel that the first option -- control of the existing phone system -- is unacceptable because it does not offer sufficient reliable functionality and performance. The PABX route -- control of a commercial telephone switch -- is impracticable for us because we do not have one. The Etherphone approach is the only alternative which provides sufficiently versatile transmission, switching, and control capabilities and which is available to us. At a later time, we may wish to integrate both hierarchical and "centralized" components into the transmission and switching system. If we do a good enough job with the control protocols for the system, such integration may not be too difficult.

SHARED VS. SINGLE LINE ETHERPHONES

This is a series of notes by Stewart on this general subject. Dan Swinehart's comments marked (DCS).

The shared Etherphone is thought of as a Dandelion class machine without a display or disk, but with a highly capable Ethernet controller and a capable T1 style TDM interface. Belleville's rough estimates place such a design at about $200 to $300 per line. Thacker has suggested that a shared controller might be more reasonable than single line micro based Etherphones.

1. Etherphones do not need much memory. I estimate 2K - 4K program (ROM) and 2K - 4K RAM. Of these estimates, the program memory is the weakest. We expect to put most of the control machinery in the Etherphone server and not in the Etherphone but . . . As for RAM, we are building a system with 30 - 40 millisecond delays, which translates to 400 bytes of audio buffering in each direction. Double that for double buffering and add some extra packet buffers for control and static and stack space. An Etherphone cannot use very much memory unless we were to hang on a lot of extraneous functionality -- 64K is way too much.

(DCS) In a "third generation" system, I'd expect most of the functions of the Etherphone server to have drifted back into the Etherphone -- or at least into one Etherphone on each cable -- so that cheap systems could be made and sold. The memory, esp. program memory, will then have to be somewhat larger. I believe your memory sizes within a factor of two or so, for the Etherphone II (one for everyone in CSL, but still not a product).

2. The shared Etherphone and the POTS gateway have considerable commonality. Both have an Ethernet controller on one side and a T1 line on the other. There are a couple of differences. With the shared Etherphone, we also need a way to get the button and light I/O done. We might use T1 time slots for this or we might run a different line.

(DCS) To build the Etherphone II B, one needs a reasonably powerful keyboard and some kind of alphanumeric display. That is easier if there's a processor nearby. The T1 time slot scheduler for all this stuff could get pretty hairy.

However, if we go with the single line Etherphone, we (CSL) might never build a POTS gateway because we may not be able to put it to much use without divorcing ourselves from the Xerox Palo

Alto phone system. (If we eventually go for replacing that system we would have more resources to use to build the POTS server.)

3. Ethernet transceivers should not be an issue. Not only will they get cheaper, but we can share them across several devices. The blue book calls for 50 meter transceiver cables. There will typically be a lot of phones and workstations inside a given 100 m circle. Even if we decide to run a separate 10 MHz net for phones and for workstations, adjacent offices can share transceivers.

a) Don't forget, though, that coax is much cheaper than transceiver cable. We could afford to let the coax lie in loops in order to get the tap points where we want them.

4. The single line Etherphone design is less work for us. (This requires accepting the idea that we won't design the POTS gateway for a while, if at all.)

[Hardware] It seems much easier to design a microcomputer out of standard parts on a single board than to (say) take apart a Dandelion and replace the display controller with some kind of T1/button/lights interface. (If we build a Dolphin board, the economic justification for building the shared device vanishes, yes?)

[Software] The shared controller must handle over 400 packets per second for 8 simultaneous users. There are laws of large numbers for concentration of telephones but 8-24 phones doesn't seem like enough to get much advantage. It is harder to write the program.

Part of the same picture is that the shared controller is more complicated. We don't know just how many conversations a given machine could handle, and the economics (see below) are critically dependent on this number.

5. The economic justification is not that clear.

a) The codecs and analog electronics come out about even.

b) You need a certain amount of ram per conversation, plus additional ram for the more complicated program. The shared machine can use larger cheaper dynamic memory chips though.

c) The program memory scales less than linear for the shared Etherphone, but its size does increase. (DCS) The shared Etherphone memory would have to be much faster stuff for the faster processor.

d) In trade for multiple Ethernet controllers and transceiver cable interfaces, the shared Etherphone needs multiple cable drivers for the T1 loop.

e) The power supply has economies of scale, as does the box (but don't forget that all the phones have boxes too, and that eventually the single line Etherphone could fit inside.) (DCS) The phone box will probably need more power than can be sent down the loop cable anyhow. A speakerphone does. So do most keyboards, displays, etc.

f) The T1 interface stuff is not that small, especially since the button and light information needs to be handled.

g) You need a certain number of processor cycles per conversation, plus a few to run the more complicated program (scales more than linear!).

h) You might use a common encryption unit instead of distributed, but this requires doubling memory bandwidth, plus setup time etc.

(DCS) If we were convinced that a non-shared EP could never be made economically feasible (favorable or neutral), I think we'd have to look at some other architecture. We are not convinced of that. It is more important to explore a design whose architecture looks superior when evaluated by any standard other than cost. I think, for us, that's the non-shared EP.

6. Shared Etherphones might improve the Ethernet utilization. Suppose that those conversations between two particular shared units are sent as a single packet stream. (This requires substantially more cycles in the shared Etherphone to unscramble the conversations.) However, we get 500 phones per 10 MHz Ethernet without shared controllers, so who cares?

(DCS) Samples from all conversations from a particular shared EP could be sent in a single packet, using multicast techniques. This would cut contention and overhead even more, at a cost of some hairy descrambling in all receivers. Silence detection, over short intervals anyhow, becomes useless in such a system. It's all very hairy.

7. Our forward vision to a product Etherphone is not crystal clear, but:
a) Single chip micros are getting cheaper.
b) Memories are getting bigger and more reliable.
c) Ethernet controllers will be like jelly beans.
d) Transceivers will be cheaper in the expected large quantities. (If telephones aren't a big market, what is?)
e) The random logic can be built as a custom part.
f) Wafer scale?

8. Larger computers are getting cheaper too. There is a product in the works with an I/O bus for server use that we might (in a few years) plug our T1 controller into.

9. It is harder to upgrade the voice quality of the shared controller than of single line Etherphones. Suppose we decide to go to 12 bits at 10 KHz, T1 and codecs no longer work. The single line Etherphone can make easier use of parallel single chip converters. (But we might run analog phones to the shared controller and have a multiplexed A/D D/A.)

(DCS) Consider even different phones with different, or variable, voice/sound quality, depending on the application and need.

10. The shared controller is an uninteresting part of the design space. Clearly there is a full spectrum of telephone system configurations, from single line Etherphones all the way to centralized PABXs. The shared controller lies in the middle of the space, with the problems of both ends and some of the advantages of each. The small simple distributed Etherphone is more appealing.

11. It will be a lot easier to experiment and play around with somewhat overpowered single line Etherphones than with a highly tuned, shared program.

Discussion

Given that we choose one approach to start, nothing prevents us from either changing our minds later or even mixing single line and shared controllers in the same system. (The POTS server is essentially this.) The single line Etherphone we can start now, we would have to use Dolphins (probably) to start the shared machine now.

We will be done sooner and at work on the bigger part of the project (applications) if we do the single line Etherphone. The hardware is the least important part of the thing anyway. The economics are close enough that we stand a good chance of being wrong whatever we decide.

WORKSTATION AUDIO HARDWARE

One obvious way to avoid the expensive separate Etherphone is to place audio hardware in our workstations. This approach is probably fine for annotation of documents, but our workstations are not designed for 100% availability (You can't get calls while you are in the debugger.) and they are not designed for real-time performance (Your call to your friend breaks up because the collector starts running.) Basically, if we want to use the system *while a special program is running* then workstation audio hardware is fine, but we can't build a telephone system that way -- it has to work *all* the time.

STANDARD MICROCOMPUTER BASED ETHERPHONES

One way to avoid building a microprocessor system from scratch is to construct stand alone Etherphones out of commercial 16 bit microcomputers. At the present time, both the processor and Ethernet would be full boards, the audio hardware would be a few extra chips, and a fairly bulky power supply and cabinet would be needed. This approach is interesting because we would use a standard single board computer that someone else has debugged but has the disadvantage that any such computer would be more general purpose than we need.

Requirement for ethernet communications very nearly requires the use of the Stanford (SUN terminal) multibus ethernet controller. There are other possibilities, the Intel chip (1982 at the earliest), the VLSI systems area (many months at least), the Intel board level product ($4000), and the SLC.

There are 8086 multibus single board computers and Z8000 SBC available now for around $1000. The Stanford 68000 board will probably be available in several months. The 68000 systems are likely to be at least $2000. Given the performance capabilities of these machines (don't forget the NSC16000 and TI9900), there is no particular reason to choose one over another.

The 8086 has always seemed more popular around Xerox, probably because it was available first. There are several projects around which use it. There is a C compiler and there is an assembler. Webster intends to build ethernet printers using the Stanford multibus ethernet controller and an 8086 SBC.

The Z8000 probably has a C compiler available for Unix, but we don't know for sure.

The 68000 has a Unix based C compiler and there is a C pup-package for it at Stanford. The 68000 board at Stanford is great overkill for us. It includes virtual memory.

No matter what we do, we would need encryption and audio hardware for such a system. We will need it anyway for the eventual Etherphone of course, but that will be a fully integrated design probably with no bus, while this device would need to speak multibus protocol (probably). One disadvantage of microcomputer audio compared with the Alto is that since there is no micro-machine multiplexing (TASK) going on, the microcomputer audio interface would need to be buffered.

The best approach looks like a multibus based system using the Stanford ethernet controller and either an 8086 or a 68000, but either way, it looks like >$3000 per box.

Inter-Office Memorandum

To	VoiceProject^	Date	September 14, 1981
From	W. Nowicki, L. Stewart	Location	Palo Alto
Subject	Voice Transmission Protocol Issues	Organization	PARC/CSL

XEROX

Filed on: [Ivy]<Audio>Doc>EpProtocol.Press

Abstract

A frequent criticism of the Ethernet local area network is that it is not suitable for real-time applications. Transmission of the human voice in the form of telephony is one application with severe real-time constraints. This memo describes some characteristics of voice transmission and the Ethernet. We have designed and implemented a real-time voice transmission protocol based on Pups. We also describe prototype implementations of the Etherphone and a Voice File Server.

Facts about Voice and Telephony

Medium-bandwidth

Telephone quality voice can be achieved with transmission rates down to 8000 bits per second, but the required compression techniques are, at present, very computationally expensive. Intermediate bit rates are a possibility, but 64,000 bits per second is the present telephone industry standard. For this reason, we restrict our attention to 64 Kbps telephone industry compatible speech. Such digital voice signals consist of sampling the analog waveform 8000 times per second and representing each sample as an 8 bit digital encoding of the amplitude of the signal. The standard encoding is called m-255, a form of segmented logarithmic companding [AT&T 80].

Real-time

Voice communication from human to human (telephony) is a real time communications problem. The perceived delay must be fairly small and constant. Tolerable delays are generally below 100 milliseconds. [AT&T 80].

Voice filing, transmission of voice between a human and a storage device, is a half-duplex function. As such, it can tolerate higher delays provided that the *initial* delay, when a connection is set up, is not too long (on the order of a second).

Echo

Echoes are responsible for much of the perceived annoyance caused by delay in current long-distance telephone calls. There is a tradeoff betwen allowable delay and the loudness of echo.

Generally speaking, with more *return-loss* (less echo), longer delays can be tolerated. There are many sources of echo. Two important classes of echos are *acoustic echo*, which occurs when acoustic energy from the receiver (speaker) enters the transmitter (microphone), and hybrid echo, which is an electrical effect caused by reflections from hybrid circuits or impedance discontinuities in 2-wire voice paths [AT&T 80, Section 7.2]. The major concern is for "Talker Echo," which is generated on the receiver side (the person listening), but perceived on the transmitter side (the person talking).

Conversation statistics

Although a conversation is potentially full duplex (both people can talk at once), usually only one participant at a time is speaking. In addition, when a person is speaking, there are often gaps between words and sentences. On the other hand, both participants do occasionally speak at once. Over most conversations, about 47% of the full-duplex channel capacity is used [Bullington & Fraser 59].

The laws of large numbers apply to these statistics. Useful data points come from the telephone industry use of *Time Assigned Speech Interpolation* (TASI), in which a certain number of trunk circuits (such as transoceanic cables) are overcommited. If 24 full duplex trunks are available, usually 36 conversations can be supported, for a ratio of 1.5. If 150 circuits are available, 300 conversations can usually be supported, for a ratio of 2.0. [AT&T 80] These statistical effects are usually referred to as the *TASI advantage*.

It is also fairly well known that only a small fraction of phones will be in use at the busiest hour. Most of the time, almost all of the phones will be unused, but of course the system must be designed for worst case behavior. However, normal bussines phone usage statistics will probably not be valid in an integrated voice and data network.

Error-tolerance

To a certain extent, the human ear is tolerant of brief distortions in speech. For digital speech this means that small, transient errors in the digital representation of speech can be ignored. Dropouts of up to several milliseconds will be perceived as "pops" and "clicks," and will be tolerated as long as they are kept sufficiently infrequent.

Virtual Circuit Service

Once set up, a voice connection should maintain an adequate quality. In the presence of network overloading it would probably be better to reject connection attempts altogether than to offer poor quality. Thus, it is often better to block new calls than to degrade old ones.

Consequences for Ethernet Transmission

The above characteristics of interactive voice have several consequences for the design of a datagram based voice transmission protocol. The two most critical requirements are that the voice transmission protocol and the end devices have sufficient throughput to support the voice data rate in a steady state and that the end-to-end delay be sufficiently small.

It is not sufficient for the system just to support the *average* data rate. The system must support the average data rate with sufficiently low variance to maintain a constant low delay. Some variation in the data rate can be compensated by increased initial delay. Voice data is buffered at the *receiving*

end so that the buffer runs dry with very low probability.

Bandwidth Considerations

The voice protocol must transmit enough packets per second to achieve a small delay. However, the number of packets per second must be low enough to be handled by software in the producer and consumer. Experiments have shown this limit to be about 100 packets per second. Inexpensive microprocessors will probably not do as well. For this reason speech compression techniques offer little help. They reduce the bandwidth, but the per-packet overhead is usually the limiting factor. The packet rate is related directly to the delay as:

$$D = 1/P + T_{Total}$$

Where D is the delay, P is the number of packets per second, and T_{Total} is the total transmission delay, described later in more detail.

A simple sequence of arithmetic can tell us that the three Megabit per second Experimental Ethernet currently available within PARC can support about 40 simultaneous transmissions at 64 Kbps plus overhead. This means about 150-200 telephones could be supported on a single network, assuming 20% of the phones are busy during peak periods. On a 10 Mbps Ethernet, about 400 telephones could be supported, and about 100 phones on a 1.5 Mbps network. The number of telephones does not scale linearly with bandwidth because of the greater overheads necessary at higher speeds. The Ethernet slot time consumes more bits at higher data rates.

Delay

Most current digital voice transmission systems use some form of time-division multiplexing (TDMA). The advantage of TDMA is fixed delay, while the delay on CSMA channels is potentially unbounded. The total end to end delay is:

$$D = T_I + T_P + T_{S1} + T_D + T_X + T_{S2}$$

Where T_I is the initial delay, T_P is the packetization delay ($1/P$ above), T_{S1} is the delay of software and process scheduling at the transmitter, T_D is the defer time (waiting until the channel is not busy, or backing off after a collision), T_X is the transmission time (time for the signal to propagate down the cable), and T_{S2} is the delay of software and process scheduling at the receiver. These times are all probabilistic, so an accurate model must consider both their means and their deviations.

Although longer packets would improve efficiency, the total round-trip delay ($2D$) must be kept well below 100 milliseconds. Most of this delay is absorbed by the very process of packetization. The first sample of a packet cannot be sent to the receiver until the last sample of the packet has been digitized. The remaining delay has a certain minimum value corresponding to the minimum transmission delay between the two stations, but is actually made longer (by T_I) in order to smooth over *jitter* or variations in transmission delay. Note that the component of jitter associated with access to the Ethernet is typically much smaller than the jitter associated with the scheduling of processes in the sending and receiving stations. Several experimental studies at Lincoln Laboratories [Johnson & O'Leary 81], and PARC [Gonsalves 81] have shown that the defering delay is essentially zero up to 80% load. Even at loads of close to 100%, packet loss rates are only a few percent. Experiments with the prototype voice transmission system under heavy loads indicate higher loses, a topic which requires further study.

Error Detection and Retransmission

To the extent that the types and number of errors in the system are tolerable to the end users, a voice protocol does not need acknowlegements or retransmissions. Experiments have determined packet loss rates to be about 0.1% through all layers of software with no retransmission. This rate will be worse for heavier network loads, but could be better with improved Ethernet interface hardware. The Alto Ethernet interface has a brief window after receipt of a packet during which it cannot receive another packet. The result of this is loss of up to 1% of the packets when both sides of an interactive conversation are speaking at once. To a lesser extent, broadcast packets cause similar problems. These problems should be solved by an Ethernet interface in the Etherphone II which is capable of receiving back-to-back packets.

Silence Detection and Echo Supression

In order to benefit from the TASI advantage, a voice protocol must detect periods of silence and utilize reduced bandwidth while silence is present. This factor of two reduction in *average* bandwidth guarantees efficient utilization of the channel capacity [Shoch and Hupp], even when there are many simultaneous conversations. In fact, some studies have concluded [Weinstein 80] that the CSMA strategy of Ethernet is *more* efficient than traditional TDMA schemes.

Since digital voice transmission provides independent transmission and receive paths, equivalent to a "4-wire" telephone, it need not be directly concerned with echo. However, acoustic echo may exist at one or both ends, and a voice protocol connection might be connected in tandem with a 2-wire transmisssion path, leading to hybrid echo. The silence threshold must be high enough to avoid detecting this echo as speech.

Etherphone 0 Transmission Protocol

The Etherphone 0 is an Alto I using an Auburn audio board. The VFS 0 is a Dorado running Pilot. The Etherphone 0 protocol permits real-time voice conversations between two Etherphone 0s, or an Etherphone and the Voice File Server.

Voice is digitized at 8000 samples per second. The samples are encoded in 8 bits using industry standard m-255 companding. When speech is detected, the Alto transmits 50 packets per second, each with 160 voice samples plus a sequence number indicating the sample number of the first sample of the packet. Since the samples are generated at a fixed 8000 Hz rate, this sequence number is equivalent to a timestamp.

Instrumentation

Since the voice transmission protocol does not retransmit lost packets, we are trying to engineer the system in such a way that it does not lose too many. Each packet has not only a sample number, but also a second sequence number indicating the *number* of packets which have been transmitted -- allowing the receiving station to accurately count lost packets.

Silence Detection

The Etherphone 0 audio microcode computes the sum of the upper 8 bits of the absolute value of the 12 bit linear encoded samples produced by the Auburn A/D converter before it converts them to the m-255 code. If this value, summed over a given 160 sample block, falls below a certain

threshold, the input is deemed to be *silent*. After a certain number of consecutive silent blocks, the originating machine stops transmitting packets. By this means, typically half the required communications bandwidth is saved. Care must be taken, however, to set the number of consecutive packets before shutdown high enough to avoid the anoying effect of shutting down between very short pauses, such as those that occurr between words.

During a silence interval, the receiving station plays silence to the listener. After a silence interval, packets again begin to arrive at the receiving station. In order to account for jitter in the arrival of future packets, the very first packet is delayed by 10 milliseconds before it is played.

Delay

We have chosen to allow about 30 milliseconds between the time a particular sample is digitized at the originating station and the time at which it appears at the D/A converter of the receiving station. Two thirds of this delay, 20 milliseconds (or 160 sample times) is due to the packetization process. The component of jitter associated with access to the Ethernet is essentially zero, while the jitter associated with the scheduling of processes in the Alto I can be a few milliseconds.

The implementation of the jitter reduction delay is as follows. An assumption is made that the *first* packet to arrive does so with a *typical* transmission delay. A 10 millisecond silence is placed on the D/A queue in front of the first packet. Assuming that the clocks of the sending and receiving station are running at the same frequency, the delay between A/D at the originating station and D/A at the receiving station becomes fixed at 20 msec for packetization, plus the transmission delay of the first packet, plus 10 msec smoothing delay inserted by the receiving station. This process is repeated at the end of each silence interval.

In fact, the resynchronization is done whenever the sequence number of an arriving packet does not match the expected sequence number, so a lost packet is treated exactly like a silence interval.

Protocol enhancements: Etherphone I protocol

The Etherphone I system also uses Alto I/Auburn as the *Etherphone*. The system includes an *Etherphone server* and uses an enhanced transmission protocol. The Etherphone Server keeps track of the state of the entire Etherphone system and is responsible for setting up calls. The protocols for communication with the Etherphone server are described elsewhere [Swinehart 81].

For Internetworking and System Control

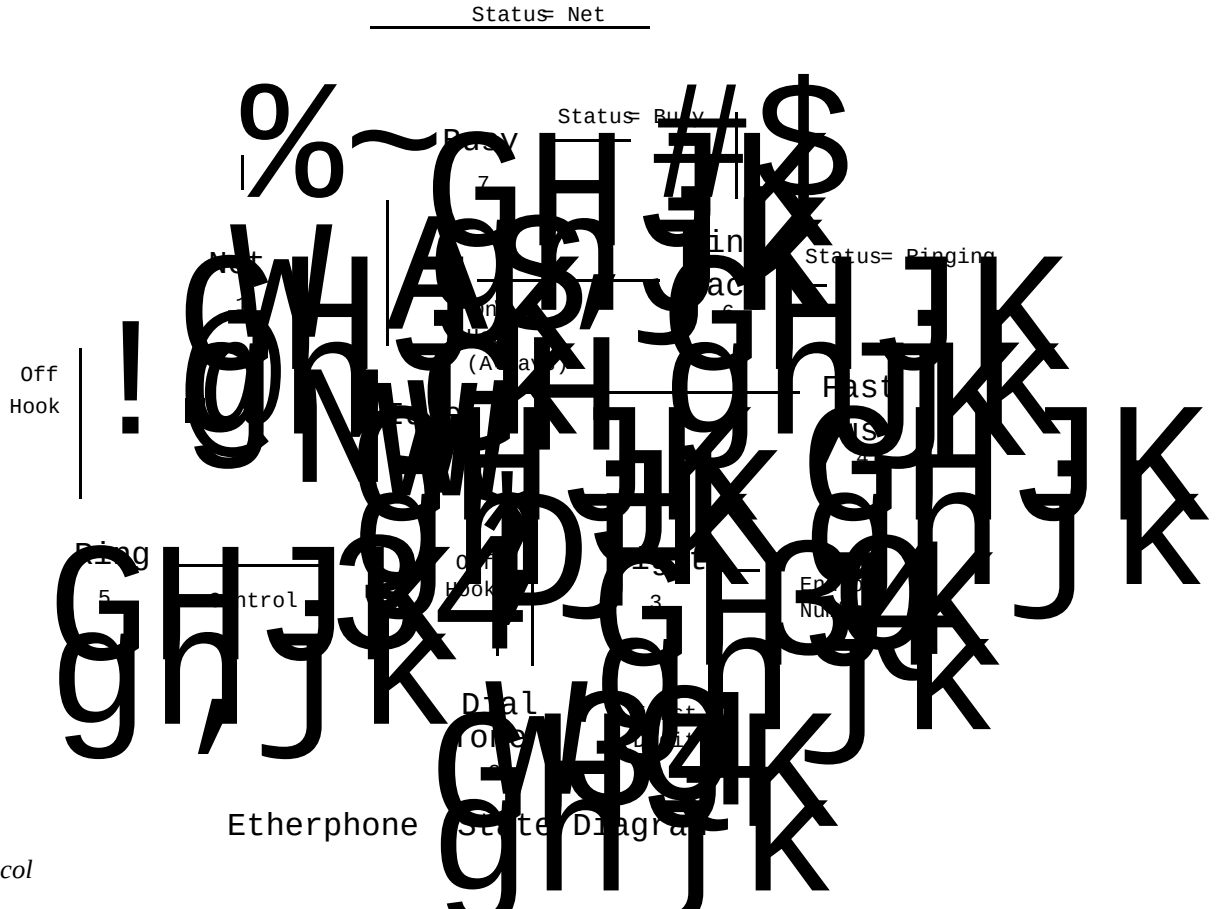
As a general principle, we are trying to avoid maintaining distributed state, and the associated synchronization problems. For example, two Etherphones must be able to tell whether a connection between them should be closed. In addition an existing voice connection should not be disturbed by other traffic. It is better to prevent a call from starting than to allow it to be disturbed once set up. For these reasons, Etherphones transmit packets even during silence intervals, which are shorter than packets carrying voice data and transmitted at a lower rate (two per second).

Appendix: Implementation Details

This section briefly describes the implementation of "Etherphone 0," a test program used to simulate the operation of a telphone. The program is written in BCPL, and uses the Auburn Audio interface for an Alto I or an Alto II. The source to the program is in `[IVY]<Audio>AubI>Etherphone.dm`. The program itself is `Etherphone.Run` in this dump file. There is a boot file version available on Ivy (as `Ep.Boot`), so a local disk is not needed. The display is not needed, except for diagnostics information.

States

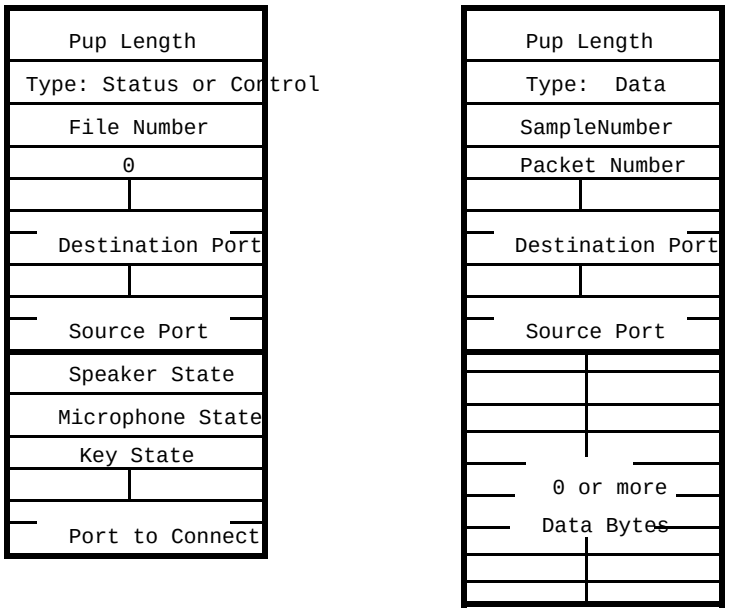
The following diagram illustrates the general operation of the EtherPhone program. There are also two other major binary state variables, which determine if the microphone and switches are active.



Protocol

The Etherphone protocol is currently very simple. It is a socket-level protocol with packets directed to Pup socket 200001 (octal). There are three kinds of packets: data (Pup type 372 octal), control (Pup type 373 octal), and status (Pup type 374 octal). The data packets consist of a sample number and packet number as the Pup ID, plus the 160 voice samples packed two to a word. The status and control Pups contain the state of the speaker (as given in the above diagram) in the first word, the state of the microphone in the second word (1 means on, 0 means off), and the state of the keys in the third word. Starting in the fourth word is the remote port to which the Etherphone is connected.

Samples are continuously digitized from the microphone at 8000 per second, using standard eight bit m-law encoding. The packets are sent on the Ethernet if the microphone state is on and the sum of the absolute values is greater than a given threshold (currently 40). This results in 50 packets per second when silence is not detected (one packet per 20 milliseconds). In the appropriate state the packets are played back with m-law decoding, with a 10 millisecond silence interval added if the sample numbers are out of sequence.



Etherphone Packet Formats

Structure

The program uses as many pre-existing packages as possible, including the standard Pup package (with checksums turned off!), the context package, and the queue package. `AubIControl.Bcpl` (written by Dan Swinehart and Larry Stewart) is the interface to the special audio microcode. `Tones.Bcpl` implements the generation of tones (touch tones, ringing, busy, etc.) including counting the correct number of 10 millisecond control blocks required to acheive AT&T standard timings.

There are four major processes (besides the Pup processes). `KeyControl`, in the file `EpKeys.Bcpl`, reads the keys and changes the state appropriately. It also prints out the status and the delay histogram. Each number in the histogram corresponds to a one millisecond interval. The first number is the number of packets played when the speaker queue was dry, the next number is for those with between one and two milliseconds of samples left, and so on.

The main code is in `EtherPhone.Bcpl`. `PhoneToNet` monitors the microphone input queue, recycling audio control blocks and sending packets when it is turned on. `NetToPhone` does most of the work, detecting off-on hook transitions, and reading packets from the network. The Speaker state is changed according to the state diagram by this process. Finally, `ACBscavanger` recycles audio control blocks that have been played, continuing the tones when they should be continued, and handles timeouts.

Use

When the Etherphone program is invoked, the display is turned off except for a telephone shaped cursor. The Shift-Lock key can be depressed to simulate taking the handset off hook, or you can use a modified telephone set which brings out the switchhook signal. A dial tone should be heard at this time. A number can be dialed as a 3 followed by he octal network address of the station to be called, or by typing a letter H and the hostname. If the called station is also running EtherPhone.Run and is on-hook, a ring-back should be heard and the remote station will ring. If the called station is off-hook, a busy signal will be played. If no response is received, a "fast busy" sound is heard. The digit 1 followed by three digits will store into the given file on a voice file server. A 2 followed by three digits plays back the given file.

Other commands which can be typed on the keyboard are:

`D`	Gets a "drop" factor. This factor is taken to be the number of packets out of 32768 to drop on the average, to simulate missing packets.
`H`	Gets a host name to connect to, as if you had dialed a 3 followed by the host number in octal.
`L`	Gets block length. This is the number of samples to include in each packet sent. The default value is 160, or 20 milliseconds.
`Q`	Quits the program, after restoring the display and displaying the statistics.
`S`	Displays the statistics, including the delay distribution. Type any other key to turn the display back off.
`T`	Gets a new silence threshold value. Default is 40, -1 means continuously send packets.
`V`	Asks for the name of a voice file server host.

When the remote station comes off-hook, the conversation can begin. The cursor moves horizontally when data packets are received, and vertically when packets are sent. To exit, terminate the call, and then use the `Q` command. The program prints the number of packets digitized, sent, and received in the last completed call before exiting.

VFS0 - Voice File server

The prototype voice file server program is written using Pilot Mesa. The source code is in `[Ivy]<Nowicki>VFSImpl.Mesa`, with a `.df` file under `[Ivy]<Nowicki>VFS.df` which refers to the `.Config` file and the many packages it needs. Just compile `VFSImpl.Mesa,` bind `VFS.Config,` and run the resulting BCD. Due to a bug in the Mesa Pup package, you must reboot Pilot after running the program. Control delete interrupts at any time. Currently only one connection is handled at any time.

References

[AT&T 80]
Notes on The Network, American Telephone and Telegraph Company, 1980.

[Bullington & Fraser 59]
K. Bullington and J. M. Fraser, "Engineering Aspects of TASI," *Bell System Technical Journal*, Volume 38, 1959 pp. 353-364. *

[Cohen 76]
D. Cohen, "On Network Protocols for Speech Communication Communication," Proceedings of the Ninth Hawaii International Conference on Systems Sciences, Honolulu, January 1976, pp. 83-86.

[Cohen 77]
D. Cohen, "Issues in Transnet Packetized Voice Communication," Proceedings of the Fifth Data Communications Symposium, Snowbird, Utah, September 1977, pp 6-10 - 6-13.

[Cohen 78]
D. Cohen, "A Protocol for Packet-Switching Voice Communication," *Computer Networks*, 2:4/5 September/October 1978, pp. 320-331. *

[Cohen 80]
D. Cohen, "Flow Control for Real-Time Applications," *Computer Communication Review*, 10:1-2, January/April 1980, pp. 41-47. *

[Cohen 81]
D. Cohen, "Packet Communication of Online Speech," Proceedings of the International Conference on Computer Message Systems, Ottawa, April 1981. *

[Cohen 81a]
D. Cohen, "Packet Communication of Online Speech," AFIPS Conference Proceedings, National Computer Conference, Chicago Illinois, May 1981, pp. 169-181. *

[Forgie 75]
J. W. Forgie, "Speech Transmission in Packet-Switched Store and Forward Networks," AFIPS Conference Proceedings 44, National Computer Conference, May 1975, pp. 137-142 *

[Forgie & Nemeth 77]
J. W. Forgie and A. Nemeth, "An effecient Packetized Voice/Data Network Using Statistical Flow Control," International Conference on Communications, Chicago Illinois, June 1977. *

[Forgie 80]
J. W. Forgie, "Voice Conferencing in Packet Networks," International Conference on Communications, Seattle, June 1980. *

[Gold 80]
B. Gold, "Digital Speech Networks," *Proceedings of the IEEE*, December 1977, pp. 1636-1658. *

[Gonsalves 81]
T. A. Gonsalves, Personal Communication, September, 1981.

[Hayes & Sherman 72]
J. F. Hayes, and D. N. Sherman, "A Study of Data Multiplexing Techniques and Delay Performance," *Bell System Technical Journal*, Volume 51, November 1972 pp. 1983-2011. *

[Hatch 76]
Hatch, "Models for the Subjective Effects of Loss, Noise, and Talker Echo on Telephone Connections," *Bell System Technical Journal*, Volume 55, November 1976. *

[Johnson & O'Leary 81]
D. H. Johnson and G. C. O'Leary, "A Local Access Network for Packetized Digital Voice Communications," *IEEE Transactions on Communications*, 29:5, May 1981, presented at National Telecommunications Conference, December 1979, paper 13.4.

[McAuliffe 78]
D. McAuliffe, "An Integrated Approach to Communication Switching," International Conference on Communications, Toronto, June 1978. *

[O'Leary 80]
G. C. O'Leary, "Local Access Facilities for Packet Voice," Proceedings of the Fifth International Conference on Computer Communications, Atlanta, August, June 1980 pp. 281-286. *

[Ross 77]
H. Rudin, "Design Approaches and Performance Criteria for Voice/Data Switching," *Proceedings of the IEEE*, September 1977. *

[Rudin 78]
H. Rudin, "Studies on the Integration of Circuit and Packet Switching," International Conference on Communications, Toronto, June 1978. *

[Shoch 80]
J. F. Shoch, and J. A. Hupp, "Measured Performance of an Ethernet Local Network," Xerox PARC Report CSL-80-2, also presented at the Local Area Communication Network Symposium, Boston, May, 1980.

[Shoch 80a]
Shoch, J. F. , "Carrying Voice Traffic Through an Ethernet Local Network," IFIP Working Group 6.4 International Workshop on Local-Area Computer Networks, Zurich, August, 1980.

[Spector 81]
A. Z. Spector, "Performance Remote Operations Effeciently in Local Networks," Ph.D Thesis, Stanford University, June 1981.

[Swinehart 81]
Swinehart, D. C., "Etherphone Server Protocol Specification," Xerox PARC Internal Memo, September, 1981.

[Tobagi 79]
Tobagi, F. A. and V. B. Hunt, "Performace Analysis of Carrier Sense Multiple Access with Collision Detection Local Area Communication Network Systems," Stanford University CSL TR-173, June, 1979. *

[Weinstein 80]
C. J. Weinstein, A. J. McLaughlin, and T. Bially, "Effecient Multiplexing of Voice and Data in Integrated Digital Networks," International Conference on Communications, Seattle, June 1980. *

References marked with * have not been located by the author.

Inter-Office Memorandum

To	File	Date	September 13, 1981
From	L. Stewart	Location	Palo Alto
Subject	Voice vs. Data: Internet Issues	Organization	PARC/CSL

XEROX

Filed on: [Ivy]<Audio>Doc>VoiceVsData.press, .bravo

Abstract

Data communications and real-time communications, in particular, packet voice, put substantially different demands on a datagram based internet. Relevant facts about the requirements of voice communications are presented. The differing needs of real-time (voice) users and data users are discussed. Some possible ways of managing the operation of a combined voice and data internet are described. A proposal for incorporation of voice and other real-time applications into the OIS Communication Protocols is made.

Facts about voice and telephony

Medium-bandwidth

Telephone quality digital voice transmission can be achieved with rates as low as 8000 bits per second, but at this writing, the required techniques are computationally expensive. Intermediate bit rates are a possibility, but 64,000 bits per second represents the present telephone industry standard. For this reason, we restrict our attention to 64 Kbps telephone industry compatible voice. Such digital voice signals consist of sampling the analog voice waveform 8000 times per second and representing the sample amplitude as an 8 bit quantity.

Real-time

Voice communication from human to human (telephony) is a real time communications problem. The perceived delay must be fairly small and constant. Tolerable delays are generally below 100 milliseconds. [Notes on the Network].

TASI advantage

While a conversation between people is usually full duplex (both people *can* talk at once), usually only one participant at a time is speaking. In addition, when a person is speaking, there are often gaps between words and sentences. On the other hand, both participants occasionally speak at once. Over conversations in general, something like 47% of the full-duplex channel is used.

The laws of large numbers apply to these statistics. Useful data points derive from the telephone industry use of *Time Assigned Speech Interpolation* (TASI), in which a certain number of trunk circuits (e.g. transoceanic cable circuits) are overcommited. If 24 full duplex trunks are available, usually 36 conversations can be supported, for a ratio of 1.5. If 150 circuits are available, 300 conversations can usually be supported, for a ratio of 2.0. [Notes on the Network, BSTJ] These statistical effects are usually referred to as the *TASI advantage*.

Error-tolerant

To a certain extent, the human ear is tolerant of distortion in speech. For digital speech this means that, to a certain extent, the ear is tolerant of errors in the digital representation of speech.

Consistant service

Once set up, a voice connection should maintain an adequate quality. In the presence of network overloading it is better to reject (block) connection attempts altogether than to offer poor quality to the new caller. A corollary is that it is certainly better to block new calls than to degrade old ones.

Other possible real-time applications

Connection to non-flow controlled data circuits

Suppose a medium bandwidth (9600 bits per second to 56,000 bits per second) asynchronous (start-stop) serial line with no flow control is connected to an OIS gateway. If the internet side of the gateway cannot keep up, the gateway buffers will eventually overflow and data will be lost. This example exists today with the Research Internet Data Line Scanner (DLS); if an internet client cannot keep up with the speed of the DLS line, data is lost as the DLS buffers overflow.

This example is not strictly real-time. There are no particular *delay* requirements, but there is a *bandwidth* requirement. As a somewhat contrived example, suppose an internet is used to link *two* such non-flow controlled circuits of the same speed. If the input line is full, the delay introduced by the internet by buffering and transit delay can only increase. Once capacity on the outgoing circuit is left idle, the time lost can never be made up.

Connection to slow speed printers

It might be desired to transmit the bit-map for a raster printer in "real-time" through an internet, with some finite (less than full page) buffering at the exit from the internet.

The general idea of a real-time protocol

Suppose there is a producer of data for the real-time application that delivers data at a constant rate. The data is collected at the originating end until a full packet is accumulated. The packets are sent (at a constant rate) to the receiver, where the data is doled out (at a constant rate) to the consumer of data. Some amount of data, perhaps less than a packet's worth, perhaps more, is buffered at the receiver to smooth out jitter in the arrival of packets.

Naturally there must be adequate *average bandwidth* to support the application. There must also be sufficiently low variation in the rate at which packets arrive at the receiver so that the receiver's buffer never becomes empty.

For a printer application, an empty buffer might mean a missed scan line. For a voice application, an empty buffer might mean a momentary hiccup in the conversation. Probably neither case is absolutely catastrophic, but such hiccups must not occur at more than some acceptable rate.

To the extent that the types and numbers of errors in the internet are acceptable to the application, a real-time protocol does not need acknowlegements or retransmissions.

A voice protocol, in order to benefit from the TASI advantage, might detect periods of silence and utilize reduced bandwidth while silence is present.

The needs of real-time vs. data users

Some general principles:

1) Data users must make progress.

2) Voice users must get the bandwidth they need, or none. It is better to block a phone call than to offer a bad connection. (Similar principles hold for other real-time applications.)

To illustrate the operation of the first principle, we offer two examples, the Ethernet and point to point links between gateways. On the Ethernet, everyone is equal. When the load is low, everyone gets the bandwidth they need and the issue is moot. When the offered load exceeds the bandwidth of the net, the contending users share equally [Hupp & Shoch]. (In fact, the contending users share the actual bandwidth in proportion to the rate at which they become ready.) For the point to point line case, the current gateway program allocates the line first come first served, and maintains a queue of packets to transmit on the line. If the queue exceeds a certain limit, packets are dropped. By symmetry, everyone's packets are dropped with equal probability. (Actually, gateways promote small packets; by generating great numbers of small packets, a client could get 100% of a phone line, locking out other users. I think this is a bug.)

The effect of all this from the standpoint of a data-only internet, is that even when the communications capacity is greatly overcommitted, all users get at least some of it, thus all users make forward progress (if you wait long enough, your file transfer *will* finish).

Real-time communications (including voice) are fundamentally different. Once a connection is set up, it should get the bandwidth it needs. It is better to refuse service altogether than to consume internet resources providing useless service. Consider what would happen if an "equal-sharing" network were slowly loaded with telephone calls. As the first users pile on, everything works fine; there is enough capacity for all. At some point, the demand exceeds the supply and the sharing property of the network allocates the available bandwidth equally to all contending users. *All* the telephone calls fail at once! It would have been better to refuse service to the "last-straw" phone call, thus limiting the outrage.

These matters can be interpreted as optimizing an objective function. The objective function for data users might be the sum of the logarithms of the bandwidths per user: more bandwidth is better, the channel is shared equally, and getting zero bandwidth is infinitely bad. The objective function for real-time users might be the sum of step functions with the various jumps at the required bandwidths for the various users: more than a certain amount is ok, less than that amount produces nothing. The maximum of this function is achieved by allocating the requested bandwidth to each user until capacity is reached, other users get nothing (but may try again later). It is not clear how to combine voice and data users within this model without adding information on the relative worth of data and voice.

There is an interesting analogy between data vs. voice users of communications and time sharing vs. personal computers. The capacity of a time shared computer is allocated equally (usually) among contending users, the capacity of a collection of personal computers is allocated in "sufficient size" chunks up to the limit of the number of computers and none thereafter. In one case the advantage of adding capacity is that more people can work, in the other, everyone can still work, but their work gets done faster. It is possible to "successfully" overload a timesharing computer by piling on users. The same is not true of a collection of personal computers.

Traffic Engineering

In the telephone industry, there is the notion of *probability of blocking*. Given a certain number of physical trunks between A and B, and certain statistics of the numbers and durations of calls placed, there will be a certain probability that all the trunks will be busy when a call arrives. Traffic engineering is the business of providing enough trunks so that the probability of blocking is acceptably low, subject to economic constraints. (Typically, users are charged more during periods of high load than at other times. This tends to even out the loading and raise the *average* utilization of the trunks.)

So far in our construction of internets, the notion of traffic engineering is one of persuading an organization to invest in capacity when their data communications become *too slow* rather than on any objective grounds.

The advantage of combined real-time (voice) and data networks

In a network, a single link is more efficient than two half-speed links. Separate links suffer from two disadvantages: one link can be overcommitted while the other has idle capacity, and, in a datagram network, the separate links will have higher delays simply because it takes longer to transmit a packet over a slow line than over a fast one. (But remember that separate links may be more reliable than a single link. One of them can go down without breaking communications altogether.) Similar advantages accrue to the use of a single network for both voice (real-time) traffic and data traffic. Both voice and data networks are (should be) engineered to handle the peak loads expected. If the voice load peaks at different times than the data load, then both kinds of traffic can use the same network capacity at different times. [Refs somewhere]

Problems

Our existing networks cannot reliably handle real-time traffic.

Our existing networks have only rudimentary notions of congestion control.

Our existing routing cannot handle class-of-service notions.

Proposals

How can the differing needs of voice users and data users be reconciled? In general, the system must recognize that the classes of users have differing needs and apply different "objective functions".

Basic Proposals

Class-ofService. Stray from the ideologically pure notion of a stateless datagram network and build a system that understands some semantics of the kinds of traffic using it. We have already departed from purity by recognizing "interactive traffic" and promoting small packets to the head of queues. Legitimizing these activities will require a *class-of-service* field in our internet packets.

Employ traffic engineering. In our present datagram-only internet, we have escaped with only rudimentary traffic engineering because we had only one class of users. With the addition of voice traffic and with larger internets in general, we will have to keep loose track of "blocking probability", line utilization, and user populations and add capacity as appropriate.

Mechanisms

Load Control. At least for real-time applications, users should be turned away once the load on a network or link has reached capacity. The same information used on a minute by minute basis to handle loading can be used in the longer term to guide traffic engineering.

Hints. Although routers, gateways, and other load control points must keep track of who is using how much bandwidth for what, they can do so in a nearly stateless fashion by using hints. We want the advantages of centralized control without the reliability problems. The same bandwidth and delay requirements that cause real-time or voice packets to pass fairly often permit the "state" information in routers to time-out rapidly. Bad information will not persist long enough to disturb the internet.

Counterproposal

Dan Swinehart has suggested that sufficient traffic engineering may remove the need for load control. Most of our networks and systems behave quite well until just below overload. At the overload point, there is a sharp "knee" and above it our systems behave quite badly. (On the Ethernet, the "badly" is in terms of delay; network utilization remains high.) Suppose we kept track of average and peak loading on networks and links, and used the information to keep capacity ahead of demand. If we arrange that enough capacity is in the right places by the time it is needed, we might be able to hold the number and durations of overload periods to an acceptable level with no minute by minute load control.

Anti-counterproposal arguments

The required degree of success at traffic engineering is too hard to achieve.

We (and our customers) don't have the money to obtain endless network capacity, we will almost always be operating near overload in order to be economically competitive. (It is not that Ethernet cable is expensive; routers and long haul circuits are expensive.)

Internet traffic is too bursty to handle on a statistical basis. While I do think we should "think big" and plan on linking 10 Mbit Ethernets with 1.5 Mbit links, we already have *individual machines* which are capable of overloading such links. The laws of large numbers are not very effective for small numbers of users.

Class of Service proposal for OISCP

The class of service for a packet should provide an indication of in which class the packet belongs plus whatever class-specific information seems useful. The purpose of class-of-service is to give hints to the internet to aid it in routing the packet. More and better quality information will permit the internet to do a better job of meeting everyones needs.

There are at least five classes of traffic: ordinary datagrams (e.g. packet exchange protocol), file transfers (e.g. Sequenced Packet Protocol with large packets), interactive traffic (e.g. SPP with small packets), real-time traffic, and voice traffic. The general idea for class of service is that it indicate in which group a packet belongs and that it provide an indication of *how much* traffic is involved. (Is this packet the only one or are more expected in the near future?)

Proposal

A *Class-of-service* field wide enough to encode the classes of traffic mentioned, with room for expansion. A *how-much* field, to indicate, variously, that the given packet is part of a stream requiring so much bandwidth, or that (loosely) so much traffic is expected in the near future. The how-much field need not be very precise but it should identify the exact requirements of heavily used real-time applications.

> Ed Taft has suggested that a single bit in the present Transport Control field may be enough. Protocols using this bit would place bulkier class information in the data portion of a packet. No present software would need to be changed and new routers could be taught about the new bit. One drawback of this approach is that real-time users are precluded from using any of the existing protocols -- which might otherwaise make good sense.

Semantics

Ordinary packets. The how-much field could be used to indicate whether more packets are expected.

File Transfers. File transfers are not usually delay sensitive, but the sooner they are completed, the happier the clients. Route along the path with maximum excess bandwidth. The how-much field

might indicate the rough amount of data being transferred, so that the routers can make more intelligent decisions.

Interactive traffic. Interactive traffic (keystrokes etc.) is highly delay sensitive, low bandwidth, and intermittent. Route for minimum delay.

Real-Time traffic. Real-time traffic requires consistant delay below some maximum, plus guaranteed bandwidth. It is better to refuse such traffic that to offer less than the requested performance. The how-much field might indicate the required bandwidth. If the best achieveable delay is too long, the end users will find out after the first few packets.

Voice. Voice is not quite the same as "real-time". In order to obtain the TASI advantage, silence detection must be used. When silence detection is used, the voice stream becomes intermittent. For voice messages, the duty cycle might be quite high. For conversations, the duty cycle is somewhat below 50%.

Examples

Managing the bandwidth of a point to point line

Consider the case of two 10 Mbit Ethernets connected by a point-to-point 1.5 Mbit link. There is plenty of bandwidth around, but it is not infinite. A pair of routers connected by a 1.5 MBit line would have a parameter indicating that up to 1 MBit of line capacity may be used for voice (or other real-time traffic), with the remainder reserved for data. When there is less than 1 Mbit of real-time traffic flowing, the idle capacity can be used for data datagrams: (and the data queue empties faster), but when there is real-time traffic around, it gets reserved capacity. The routers keep an eye on packets coming in. Suppose the router sees a real-time, how-much=64 Kbit packet for a new source-destination pair. The router takes this as a hint that a new "stream" is being set up and makes a table entry "reserving" capacity for the connection. By using the how-much field together with the packet length, the router can predict when the next packet of the connection is expected. The table entry can be deleted (timed-out) if the next packet doesn't show up. (Thus there is no "stream setup" protocol, it is all done with hints.) When it happens that the n-th+1 apparent stream shows up, the router drops the packet and sends an error reply "no capacity now".

Now consider the case of "voice" traffic rather than just "real-time". A typical phone call uses each half-duplex path slightly under 50% of the time. A voice connection would send 50 160-data-byte packets per second while talking and would also send small packets at a lesser rate during silence in order to let the routers know (via the hint mechanism) that the 'connection' was still there. The router could actually get away with allowing, say, 20 'connections' over the 1 Mbit of capacity rather than only 16. Only for brief periods would the offered load from the 20 conversations exceed 1 Mbit. When that happens, the router could intrude momentarily into the "data" bandwidth.

Managing the bandwidth of an Ethernet

Consider the use of an Ethernet for telephones. So long as the total offered load is below the "knee" in the delay curve, the Ethernet works very well. Much above the knee, its performance may not be adequate for voice. The exact position of the knee is dependent on the distribution of packet sizes and on the average number of stations contending for the channel but it is in the 50% to 80% area for voice packets.

If too many people attempt to make calls at the same time, the Ethernet delays would grow rapidly, disrupting service for all. One solution is to register calls with a server -- callers would not get dial-tone if the Ethernet could not handle their call. Another solution is to monitor the general levels of Ethernet traffic and to split the network into two parts (adding capacity) well before the loading reaches dangerous levels. (This is just a localized version of the counterproposal described above. Its successful application might depend on separate Ethernets for voice and for data.)

More complex is the problem of using an Ethernet as a transit network in an internet. While a telephone server might register calls and perform load control on a local basis, who could take the responsibility for internet traffic? One approach might have the routers (perhaps using special hardware), watch *every* packet on the Ethernet and keep track, by hints, of the traffic levels. Transit connections could be blocked before entering a congested region.

Internet Issues

Routing

These hint schemes have the flavor of fixed routing. Once a call is set up along a particular path, the path is hard to alter. There is nothing wrong with this! Think big: there is a *lot* of traffic in our hypothetical internet. That particular connections are not rapidly rerouted is fine, load-sharing can be done by routing some connections one way and some another. If a particular connection's bandwidth requirement cannot be met without load sharing, the internet is probably operating too close to overload. Our present routing mechanisms can not deal effectively with these concepts, but then, they cannot, at present, handle excess bandwidth or delay either.

Accounting

The same hint based measurement machinery that is used to perform load control can also handle accounting.

Traffic Engineering

The load information can be used to prepare summaries of line and net utilization for traffic engineers. The information might be detailed enough to identify new candidate networks in the internet for direct connection.

XEROX

Office Products Division

Office Systems Business Unit

To:	Ed Miller	Date:	August 19, 1981
From:	Bob Belleville	Org:	OSBU/SD&T/A&S
Subject:	**Voice and Telephone Management for OIS Workstations**	Filed:	[Iris]<Belleville>Voice81-2.memo

During March 1981, I wrote the first version of this memo to describe the potential for a computer controlled product to better integrate voice communication in the office with our OIS workstations. During the six months since that time, I have refined the concept through converstations with people in planning, user interface design, senior management, and potential users.

Summary

Product concept. A small device (approximately 7" x 14" x 2.5") to be placed under the user's own standard telephone set. The box would be connected to the local telephone wiring via the standard modular jack and the user's telephone would plug into the box. The box would connect to the users workstation via a cable. Workstations could range in performance from the low end microprocessor based systems (820 etc.) to the full capacity Star systems; however, the range of features and performance would vary from station to station. The box would be separately powered. The principle features include:

Management of the users telephone - auto dial, redial, telephone answering machine, speaker phone, etc.

File annotation and voice mail in conjunction with Star software.

Dictation machine.

Detail Product Configuration

The voice box provides a large number of relatively simple components which are configured dynamically by the host processor via the 300 Baud, RS232C control link. In this section, I will describe the component parts in detail. The previous memo (Voice81-1) gives scenarios of use. Please refer to the "Overall Organization of the Voice/Telephone Interface Box" diagram.

Relay. The user inserts the voice box "between" his telephone set and the wall outlet of the telephone local loop (which goes either to a central office of the phone company or to the users PABX (Private Automatic Branch Exchange). The relay insures that when the power fails the users phone will still operate. The Control Processor switches the relay to either the "normal telephone service" or "computer assisted service" positions. (In the figure the relay is shown in the "normal" position.)

Digital Control Bus. The voice box requires a number of very low speed digital control signals to operate. Rather than provide dozens of signal wires in the host workstation interface cable, a Control Processor is included to communicate with the host computer via a simple two wire RS232C link. Commands to the voice box are sent from the host and decoded by the Control

Processor into a number of digital control signals on the digital control bus. By the same token, signals from the voice box (the ring indicator for example) are encoded into commands and sent to the host via the control link. In general, the voice box has no intelligence but simply passes information to and from the host. For example, any user interface aspects of the product are a function of the host processor software not of the voice box firmware.

Telephone Set Interface. In order to use the telephone set as one of the audio input/output devices an interface must be provided. The interfaces provides -48 VDC power to the phone set, senses "off hook", and includes a hybrid to convert from the phone's two wire interface to the separate input/output signals required by the rest of the unit. (No ability to ring the users phone is provided because this requires control of 100 VAC 20 hertz power. The built in speaker and tone generator provided by the Codec/DTMF encoder provide audio signalling to the human user.)

Phone Coupler and Hybrid. The connection to the local loop requires another interface which is similar to the interface above but provides other features. The coupler limites the audio level presented to the phone line to meet FCC requirements, senses the ringing signal, places the coupler On/Off hook, and provides a FCC acceptable connection to switch network. Another hybrid is needed here to to the "two to four wire" conversion. (Control of the "yellow/black" wires of the standard phone connection is still under evaluation. This pair controls the light on a secreatries extension phone to indicate that the line is in use.)

Tape Transport. This unit is discussed in detail in the previous memo and provides for temporary analog voice storage, dictation, and telephone answering machine functions. Stop, Run, Record, Rewind, Fast Forward, and tape position information is controlled via command from the host processor.

Indicator Lamp. A single LED, visible to the user on the front of the package, can be used by the host software for any purpose. For example, message waiting.

DTMF Encoder/Decoder. This unit has two different functions. The encoder generates dialing tones to implement the dialer function. The decoder allows the user to command the voice box (or rather its host workstation) from a distant phone - to here messages or send them. (In the past, these functions have been done via host software. If we are to allow low end processors, these functions must be implemented in hardware. Fortunately, there are LSI components available.) Control and status information is passed to and from the host via the control link as with all such functions.

Microphone/Speaker. This is a collection of analog hardware to interface to the microphone and speaker. AGC (Automatic gain control) is used to free the user from adjusting the microphone volume. A control for the speaker loudness is provided on the front of the package.

Codec and Filter. These two chips are the interface between analog and digital voice signals. The digital interface is the standard (T1) protocol used in some central offices and is generated directly by the Codec chip. This protocol allows up to 24 voice boxes to be operated together on a server (a system not yet fully described). Only "Star class" workstation can take full advantage of the digital aspects of the voice box because of the high data rate (total of 128K bits per second for a full duplex converstation). Any processor can use the voice box via the common 300 baud RS232C interface connection provided on all computers. A special interface is required for the digital voice signals to and from the Codec. (Relatively simple interface requirements.)

> Fine point. Many aspects of the digital voice capability of the full voice box can be made available to the low end user by using the tape transport and a centrally located voice box/Star system. The details of such a system are still under investigation.

Power Supply. Because it is impossible to state which computers will use the voice box, separate DC power is produced by the unit. Battery operation is not feasible. This approach is consistant with the modem industry which uses separate power. (Some units are powered from the -48 Battery voltage supplied by the phone company; however, the total power requirements of the voice box

prohibit this approach.)

Analog Switch. This is the heart of the unit. The switch connects almost any analog source to almost any analog destination. It is by this means that most of the power of the unit is realized, because the switch is configured dynamically to meet the users requirements. This actually a 4 x 4 switch with the inputs (the columns) being voice, phone, tape, and synthetic. Possible sources for voice are either handset or microphone. Synthetic is always the analog sum of D/A and DTMF. Output from the switch (the rows) are voice out, phone, tape, and A/D. Voice out is either handset or speaker (or both) and the DTMF decoder is always connected to A/D.

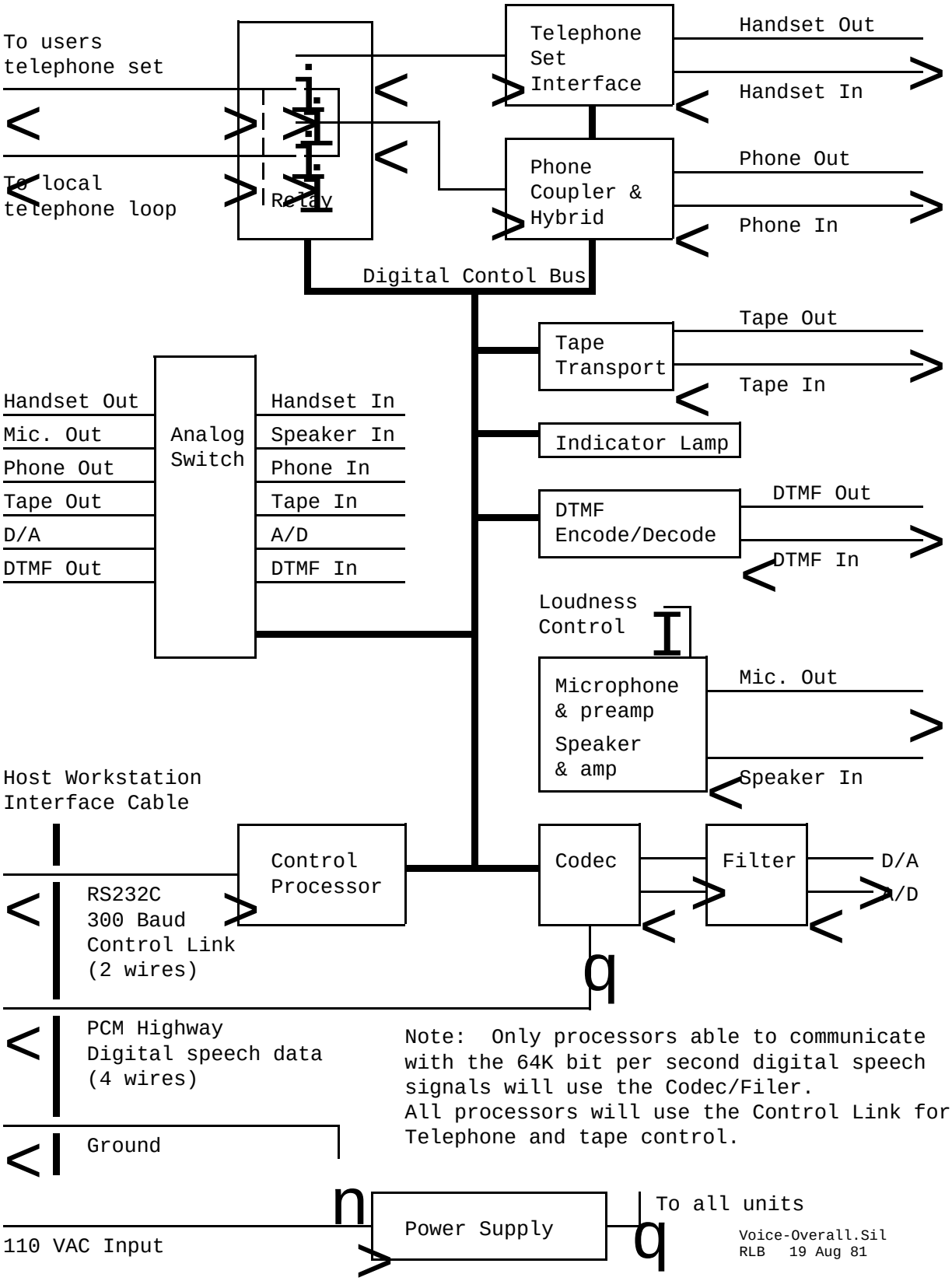
Control Processor. This will probably be a single chip microprocessor of the 8048 family. No high speed data flows on the digital control bus so performance requirements are modest. This is also the reason for limiting the input baud rate to 300. The software for this processor will be in ROM or PROM and will not be loadable in anyway.

Diagnostic features

In certain cases command protocol will be added to the firmware to assist the host in diagnosing failures of the voice box. Since this is a low cost, single PC board unit, I assume that in most cases the user will simply return the unit to a service station for repair. I assume that the whole unit will be a FRU and the field diagnostics need only determine if the box itself is bad. This should be relatively easy by using the programmable analog switch. The switch and a number of test points and signal generator will be needed to complete manufacturing testing. There many have to be a few internal adjustments because of the analog nature of the system. The tape transport will require infrequent cleaning; however, most users are familiar with this operation and many products are on the market to meet the need.

c**:** tbd

Overall Organization of the Voice/Telephone Interface Box



Note: Only processors able to communicate with the 64K bit per second digital speech signals will use the Codec/Filer.
All processors will use the Control Link for Telephone and tape control.

Voice-Overall.Sil
RLB 19 Aug 81