(2)

Three Reports on The Frame Mode Interface to Digital Networks

MICROPROCESSOR SYSTEMS DEVELOPMENT LABORATORY
DEPARTMENT OF SYSTEMS ENGINEERING AND COMPUTING SCIENCE
CARLETON UNIVERSITY

Microprocessor Systems Development Laboratory
Department of Systems Engineering and Computing Science
Carleton University
Ottawa, Canada

15 November 1978

(2)

Three Reports on The Frame Mode Interface to Digital Networks

- A Critical Review of Proposals for the Frame Mode Interface to Digital Networks
- BASIC FDTE: A New Frame Mode Interface to Digital Networks
- An Experimental Investigation of the Frame Mode DTE

James K. Cavers (1)	Principal Investigator, Designer
Jean-Louis Paquet	Design, Implementation
William Sullivan	Design, Implementation

Prepared for the Department of Communications, Government of Canada, Under Contract OSU78-00182.

A CRITICAL REVIEW OF PROPOSALS FOR THE FRAME MODE INTERFACE TO DIGITAL NETWORKS

by

JAMES K. CAVERS
Microprocessor Systems Development Laboratory
Faculty of Engineering
Carleton University
Ottawa

A report prepared for the Department of Communications, Ottawa under contract OSU78-00182. The opinions expressed herein are those of the author and do not necessarily reflect a position on the part of the Department of Communications.

November 15, 1978.

ABSTRACT

This is the first of a set of three reports on the Frame Mode Interface to digital networks. The objectives and constraints of this somewhat controversial subject are summarised, as are a number of practical considerations sometimes overlooked in proposals to standards bodies; some of the objectives are thereby shown to be incompatible.

As an aid to discussion of the various Frame Mode DTE (FDTE) possibilities, the report presents a classification of such DTEs based on the location of the call control function with respect to the data link control (DLC) function. It is shown that X.21-like protocols present the greatest possibility for uniformity of access to circuit and packet networks, although X.25-like call control protocols provide the greatest flexibility for use over packet networks.

Interworking of various DTEs on the same or different networks is considered and it is shown that interworking during the data transfer phase of a call presents few problems. Call control through existing standards, however, remains stubbornly nonuniform, particularly in the case of internetwork calls; a DTE must perform different start of call actions depending on the nature of the remote DTE. As a digression, it is shown that X.21 terminals on a circuit network can be augmented by a layer of end-to-end protocol to facilitate internetworking with packet or other circuit networks. This observation is in preparation for a truly universal call control method presented in a companion report.

Three representative proposals on the FDTE are summarised using the framework established by the report. None of them meets all the requirements of the desired Frame Mode Interface.

CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

Three or four years ago the urgency of defining interfaces to public packet networks (PPNs) led to rapid acceptance of CCITT Recommendation X.25. This interface has been adopted and supported by most public and private carriers in North America and Europe. Now that the crisis has passed, and both the networks and the user community have gained experience in the implementation and use of X.25, it is possible to ask if there are improvements or alternatives which could make access to data networks easier.

The single channel Data Terminal Equipment (DTE) is a case in point. For synchronous DTEs such as IBM's 2780 or 3270 which require only a single virtual channel, the full X.25 interface is an obvious overkill. Of the Level 3 functions, multiplexing of logical channels is unnecessary, flow control is redundant (since it is also present at Level 2), and only call control of some sort is required. During the data transfer phase, Level 2 alone (the frame level, HDLC) is adequate for providing sequencing, flow control and error control. This observation has motivated CCITT activity to define a simpler frame mode access method which would require less DTE memory space and CPU time than that of a full X.25 packet mode DTE(PDTE). This rather straightforward objective was generalized when it was realized that DTE/DTE communication through circuit switched facilities- analog (voice) or digital (X.21-like) - presented the same limitation to a single channel, and possibly a similar potential for simplification.

Accordingly, access to digital circuit switched networks was included with simplified packet network access as a design goal for a new access method and a new type of terminal: the frame mode DTE (FDTE).

More specifically the objectives assumed implicitly or explicitly by most authors are:

a) FDTE/FDTE communication through a leased line, or through a switched digital circuit service, or through a switched virtual circuit (packet switched) service;

b) FDTE/PDTE communication through a switched virtual circuit service, where the PDTE accesses the network through full X25;

c) FDTE/FDTE or PDTE communication through a tandem connection of real digital circuit and virtual circuit. The PDTE, if present, is connected to the packet network;

d) multipoint access to the network for FDTEs, each FDTE on the access line having the ability to establish a call to a different destination.

That some of these objectives are mutually incompatible, or perhaps over-ambitious, will be discussed in the next chapter.

The definition of a simple FDTE is complicated by a number of constraints:

a) Existing standards should be observed to the greatest extent possible; in particular Recommendation X.21 for access to circuit switched networks and HDLC elements of procedure, at least, for access to virtual circuit

packet networks (existing HDLC classes of procedure are even more to be preferred). In addition, de facto industry standards such as SDLC and even BSC should at least be considered.

b) DTE memory requirements and CPU load should be kept small.

c) Development time, which is related to complexity, should be must less than for full X.25 and not significantly greater than for HDLC alone, and modification to existing software should be minimal. It is true that software development can be amortized over the number of terminals sold, but if the development effort is prohibitive then only a few manufacturers will attempt it and competition will be reduced.

This report examines some of the general considerations relating to the frame mode DTE with a view to reassessment of the commonly accepted objectives listed above. Chapter 4 presents a framework within which the various existing proposals can be placed. The three alternatives thereby defined are examined separately and then compared, and the most promising approach is identified. Several Appendices summarize position papers on the FDTE which have been submitted to CCITT.

2. GENERAL CONSIDERATIONS

This chapter contains some obserations relating to communication protocols and services which the reader should bear in mind during the arguments of subsequent chapters.

2.1 Complexity

Data link control procedures such as HDLC, DDCMP or BSC are difficult and expensive to implement properly. HDLC in particular, because of its many encrustations and occasional ambiguities, has a reputation of requiring a big development effort. By contrast, the packet level (level 3 of X.25) is relatively simple because of its clean structure and the fact that the complexity of sequencing and error control is handled within HDLC (level 2). Informal reports from the Computer Communications Group at Bell Canada put the breakdown of effort as 80%-90% on HDLC and 20%-10% on the packet level. The experience of the author's group in the Computer Systems Development Laboratory at Carleton University was similar, though perhaps less extreme. Significant modifications to working HDLC software are not relished by communication software groups.

2.2 Memory Space

Memory space required for the code alone of X.25 is typically about 12 K bytes on a microprocessor though the figure varies depending on implementation (processor type, peripheral hardware, software structure, features included, etc.). The split is approximately even between HDLC and the packet level.

We turn now to the question of roughly how much reduction in memory space can be expected in the FDTE as compared with the PDTE. Recall that, of the main packet level functions of X.25, only flow control is redundant and can be relegated to the frame level. Since flow control typically accounts for less than half the code of the packet level, we see a maximum space reduction of only 3 K bytes. Perhaps X.25 is not so bad after all!

As for the remaining function of call control, over 3 K bytes are required in a typical X.25 implementation. No matter what scheme is finally adopted for the FDTE, this function must be present somewhere, and there is no reason to expect drastic changes in its memory requirements. As a consequence, all FDTE proposals will take up approximately the same amount of space. Selection of one proposal over another will therefore be based on considerations of complexity and functional capabilities, not on memory space.

2.3 Interworking During the Data Transfer Phase

Interworking during the data transfer phase of a call becomes a matter of defining pairs of anchor points for the data link control procedures between which flow control, sequencing and error control are exerted. In the case of a leased line, voice circuit or digital circuit connection (Figure 2.1) the anchor points are the two FDTEs. For packet network connections (Figure 2.2) the anchor point pairs are: (FDTE, DCE), one access link; (DCE,DCE), the network; and (DCE,FDTE), the other access link. Finally for tandem connections between circuit and packet networks (Figure 2.3), the anchor point pairs are; (circuit FDTE, gateway DCE), the real circuit; (gateway DCE, DCE), through the packet network, with some variations possible here depending on the level of interconnection; and (DCE,FDTE or PDTE), the access link. Note that there is no particular requirement for the DLCs to be the same for all anchor point pairs. Mixtures of HDLC, BSC and DDCMP are workable, although there is normally a strong incentive to keep the number of packages to be developed and maintained to a minimum.

2.4 Call Control on Circuit Switched Networks

The present CCITT standard interface to circuit switched networks is X.21. In the context of the present discussion, the important feature of this interface is that the network is notified of changes in call status (e.g. call request, clear request) by simple voltage changes on specified pins of the interface plug. Address information and call progress and service signals are exchanged as special bit sequences, but once the call is established, the real circuit is completely transparent to the data exchanged between terminals.

Consider now the use of special frames or bit sequences, rather than special pins, to alert the network to changes of call status. Several proposals before CCITT (e.g. Canadian, IBM Europe) have made this suggestion for FDTE call control. It is clear, however, that any such method violates the very essence of real circuits: complete bit transparency (or at least byte transparency). The user would have to be careful to avoid dangerous bit patterns during the call. The ultimate result of this approach would be a circuit network which works only for, say, HDLC terminals!

Another problem associated with use of special bit sequences to perform call control on a circuit network is that the network must provide each access line with a scanner which is dedicated to watching the bit stream for call control frames, and alerting the switch controller when one is received. (Note that a single processor would be insufficient for this task in the circuit switching context). Compare this requirement for extra equipment with the simplicity of X.21, in which the call control

leads of many access lines can be polled at a relatively slow rate, and the more sophisticated bit sequence (address) interpretation firmware can be directed to the DTE presently requesting service.

We close this list of the deficiencies of bit sequence-based call control with a minor complaint. Those frames emitted by one DTE to close a call will emerge at the other DTE before the call is cleared, since the node cannot buffer entire frames without violating another characteristic of circuit switching, the low constant delay. The closing protocol would have to be designed with this eavesdropping in mind.

2.5 Multipoint Access to Circuit and Packet Networks

Multipoint access means that several DTEs share the same access line to the network. Such an arrangement would be convenient if, for example, there were several low data rate or low activity rate terminals at a customer's premises. It would not be necessary to install separate lines to each terminal.

Multipoint access to a packet network presents few problems, since multipoint DLCs have been in existence for years. The basis of sharing the access line is frame interleaving; coordination among the DTEs is supplied by the rules of the DLC. The variable queueing delay on the access line appears as one component of the larger queueing delay through the network as a whole.

Multipoint access to circuit network, on the other hand, is more difficult. The basis of sharing the access line could not be frame interleaving, since queueing delay would violate one of the characteristics of real circuits: that of low, constant delay. Byte interleaving is the obvious approach (in fact, the present X.21 specifies one pin of the interface as supplying byte timing), but it would require multiplexing and synchronization hardware which is not commercially available. If it were available, it would be expensive, since the problem amounts to setting up a special sort of TDM line.

2.6 Internetwork Calls

Consider an internetwork call placed by an X.25 PDTE on a packet network. The call request packet will be directed by the local network to the appropriate gateway machine, which can examine the address information in the packet and place a call to the destination DTE on behalf of the originating DTE. The originating DTE experiences only one stage of call control, in principle, no matter how many intermediate networks are placed between it and the destination DTE. The same service can be given to a FDTE accessing the packet network through an X.21-like interface.

On the other hand, a gateway node connecting a circuit net to other circuit nets or packet nets might see an X.21 interface to its circuit nets and an X.25 (or X.75) interface to its packet nets. In this case, an X.21 FDTE on the circuit net, when placing an internetwork call, will have to go through two or more stages of call control, since no address information is given to the gateway by the circuit net. The X.21 FDTE will experience different call control services over packet and circuit networks.

2.7 Signals Exempt from Flow Control

A protocol normally contains some signals which are exempt from flow control; they are allowed to be transmitted to the node (or to the remote DTE) even if the destination has indicated unwillingness to accept more data packets. Examples are call clear requests, interrupt packets, and frame level set mode commands.

The usual mechanism involves establishing some "out of band" identification. In X.25, for example, the packet header establishes packet type, and only data packets are subjected to flow control (though restrictions may apply to other packet types, such as a limit of one outstanding interrupt request). In X.21, a special interface pin signals a clear request electrically. It is possible to extend X.21 with more pins for other out of band signals, but considering the fact that the number of such signals may grow, it seems wiser to have a single out of band pin with an associated bit sequence transfer to provide details.

One problem we encounter when relying on frame level flow control, therefore, is that all packets transmitted through the DLC are subjected to the same flow control. This can cause delays in sending clear request or interrupt request packets. There are several approaches:

- live with it;
- embed the flow control exempt signals as special frames within the DLC [1];
- precede each transmission of such a packet by a resetting of the frame level, on the understanding that reset clears the transmit and receive queues.

The second approach is cleanest, but involves the extension of existing standards with all its associated debate. The third approach is brutal, in that potential data loss is associated with each reset, but needs little, if any, adaptation of existing software. Further, the data loss may be tolerable at those points in the end-to-end dialogue where a flow control exempt signal must be sent, such as interrupt or call clear.

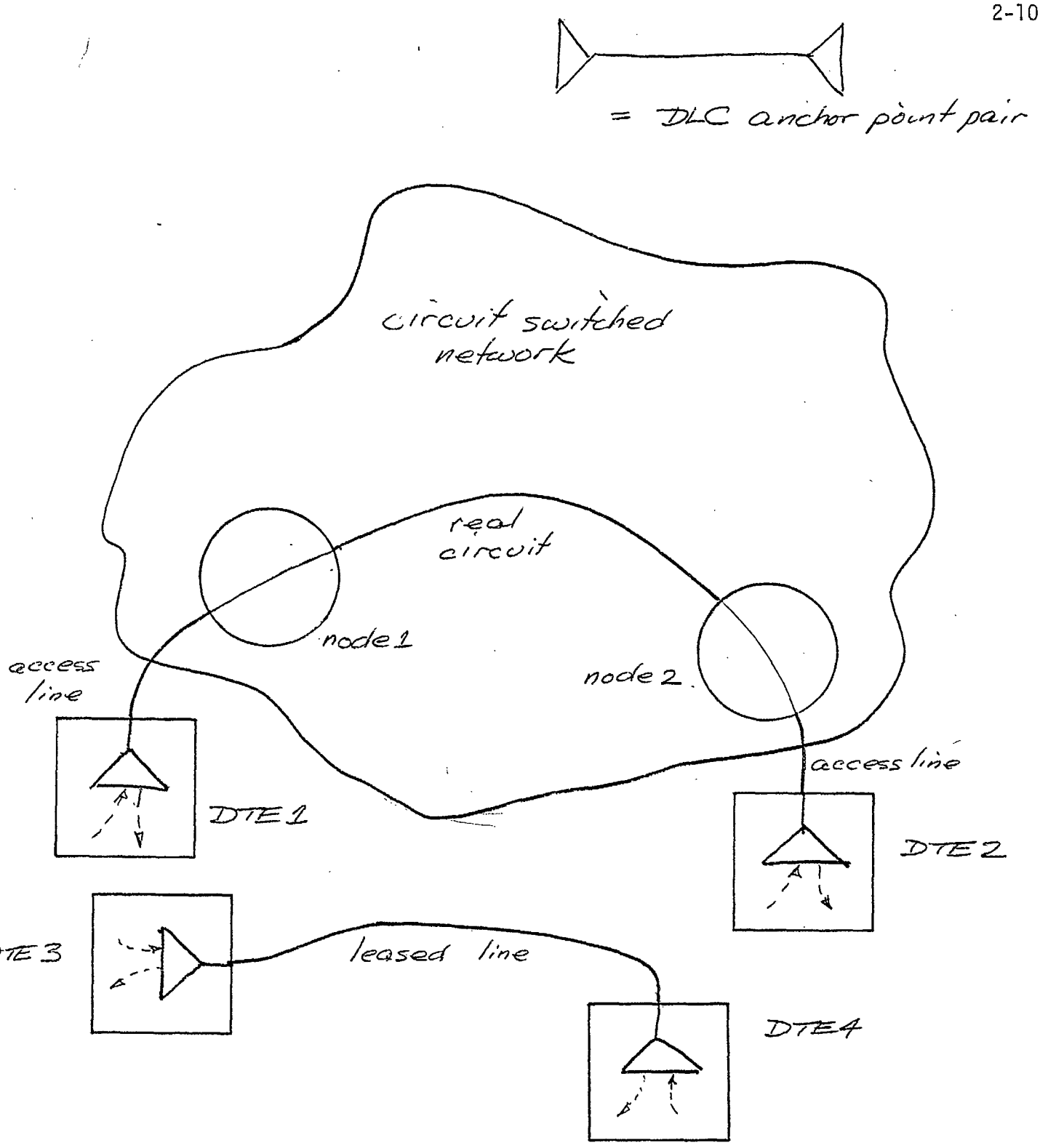


Figure 2.1 DLC Anchor Points: Real Circuits

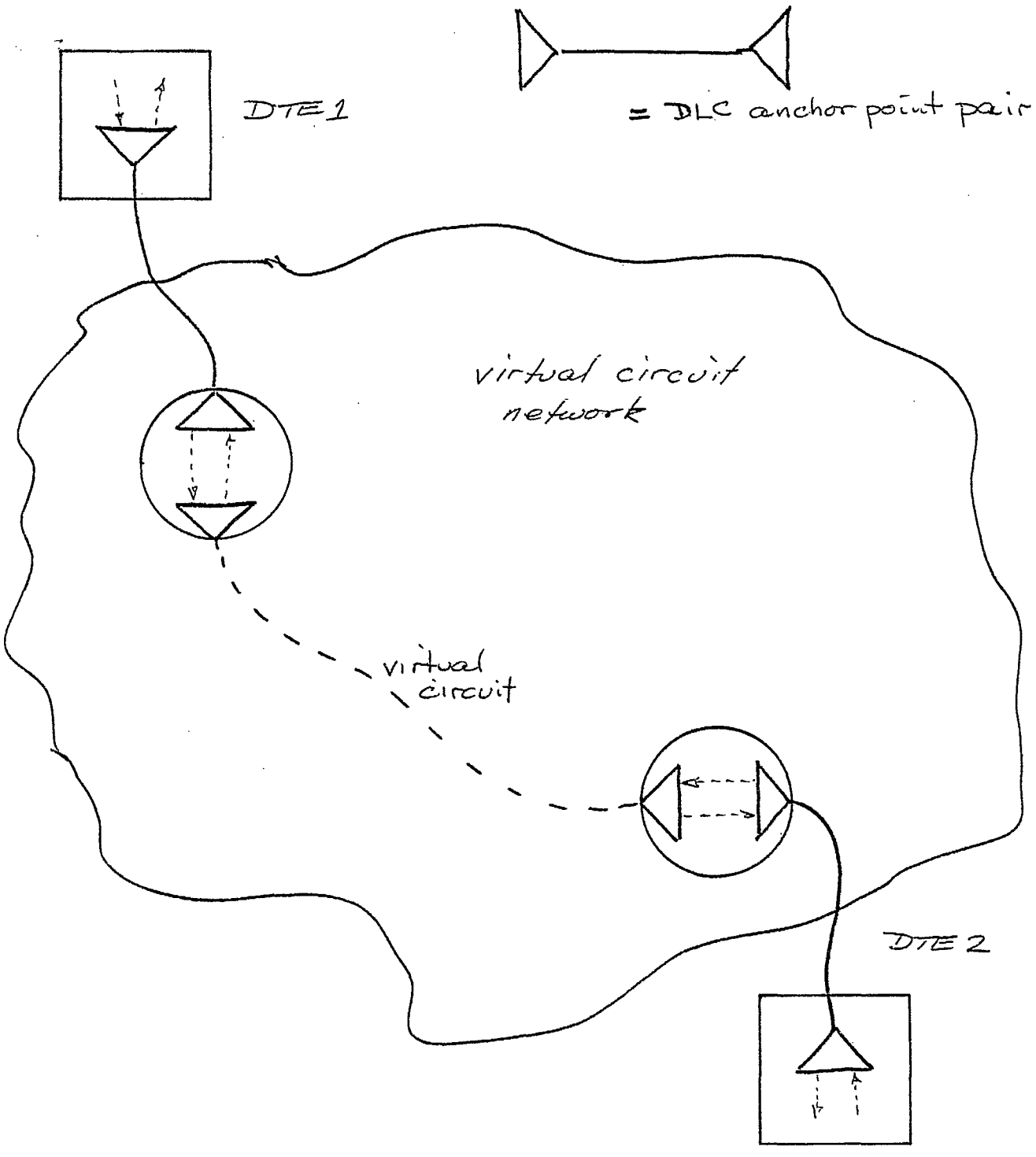


Figure 2.2 DLC Anchor Points: Virtual Circuit

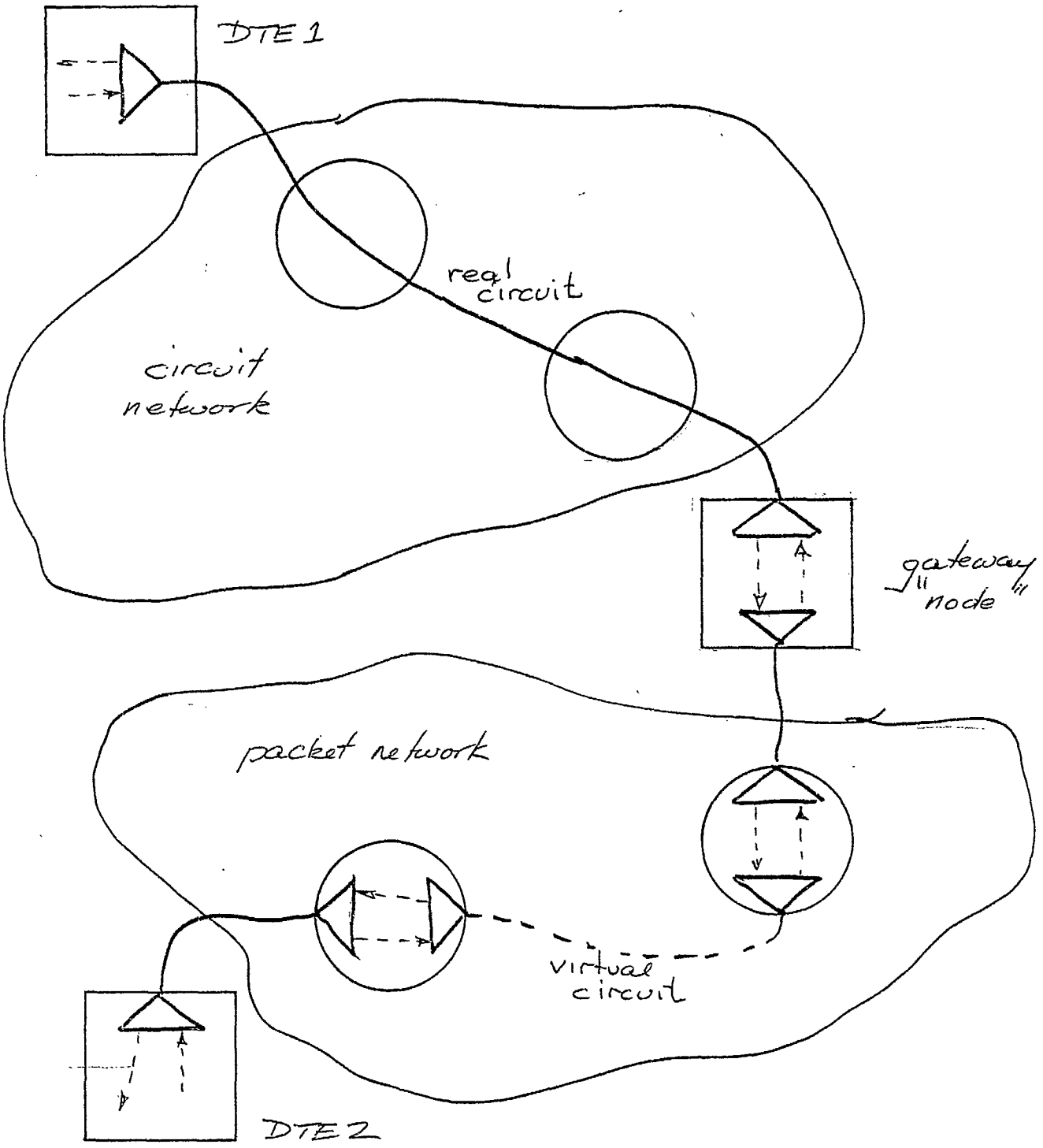


Figure 2.3 DLC Anchor Points: Tandem Connection

3. IS THE UNIVERSAL DTE POSSIBLE?

We have seen in Chapter 1 a list of objectives and constraints which apparently must be met by the FDTE. We have also seen in Chapter 2 some of the considerations which must be taken into account in any realistic FDTE definition. This chapter is a reassessment of the stated objectives in the light of the considerations of Chapter 2; it will be seen that, while the objectives taken singly are reasonable, taken as a whole they contain mutual contradictions.

Consider first the requirement for a uniform call control method on both packet and circuit networks. As discussed in Section 2.4, this leads to rejection of call control based on exchange of special frames or bit sequences. Notification of changes in call status must take place by voltage changes on specified interface pins, in a similar fashion to the present X.21. In fact, X.21 is a satisfactory solution for this requirement.

Next consider the multipoint access requirement. Multipoint access can be achieved easily by bit sequence based call control (as, for example, by X.25-like call control above a multipoint DLC). Section 2.5, though, demonstrated that use of call control based on voltage changes on a multipoint line, while possible, is expensive.

We seem to be faced with a choice: do we abandon the requirement for multipoint access or the requirements for identical call control procedures for access to circuit and to packet nets? Multipoint access is too attractive to dismiss, since a single access line to a group of customer terminals offers substantial economies in the case of packet net services.

On the other hand, the concept of a single uniform call control method is intriguing, and its adoption may conceivably lead to some economies of scale in production.

Figure 3.1 summarizes the situation: X.21 gives uniform call control but uneconomic multipoint access, while bit sequence based call control (e.g. by X.25 or in the DLC) provides multipoint access but no access to circuit nets.

One other point deserves consideration. Section 2.6 showed that, if uniform call control on circuit and packet networks were required to extend to internetwork calls, then a new gateway interface to a circuit net would have to be defined. We find ourselves in the position of having to create, not one new standard, but two: the gateway interface and the FDTE interface.

Why are we drifting farther from our objective of a universal FDTE, instead of closing in on it? It is the author's contention that there has been a confusion of goals in the objectives listed in Chapter 1. The similarity of operation in the data transfer phase of a call between the single-channel DTE operating on a packet network, on one hand, and the DTE operating over a circuit switched network, on the other hand, led to the conclusion that there should or could be a complete correspondence, a single uniform FDTE, even to the call control protocol. The conflicts and awkwardness turned up so far suggest strongly that this is not so.

A reassessment of the objectives is the obvious means to resolve the dilemma. Listing them in what now appears to be order of decreasing priority we have:

- the ability of FDTEs and PDTEs to interwork during the data transfer phase;
- a single, uniform call control method for both circuit & packet networks;
- simplified access to packet networks for single channel DTEs;
- multipoint access to packet networks.

No one method achieves all these goals economically, as we have seen. So why should we not abandon the second goal and allow two or more methods? The arguments in favour are simplicity and the possibility of using existing standards. Chapter 5, in fact, demonstrates that a simple layer of end to end protocol to augment X.21 on circuit nets will facilitate internetworking and interworking with X.25 DTEs on packet networks. Countering these arguments is the fact that for the terminal manufacturer to support multiple options is an increased load on his distribution and field service organizations.

It is possible, however, to supply two or more call control software modules in the same terminal, with a switch to select between them. The falling costs and rising densities of semiconductor components will make this even cheaper in future (certainly cheaper than the data sets forced on us if we try to combine the second and fourth objectives above). In any case, the bulk of the terminal software - the DLC, the keyboard and display support, and any peripheral handlers - would remain the same regardless of call control technique. The argument against such an approach is that the complexity of the resulting software would mean increased development time and software maintenance cost.

As for the third objective, we note that implementation of X.25 for a single channel DTE is not really very difficult, once a working frame level module is available (see companion report [3]). There is some irony in the fact that the third objective was the original motivation for FDTE investigations.

The fourth objective, multipoint access to packet networks, is still considered important because it reduces the number of ports at the node. Further, many locations in Canada are still served by expensive analog backhaul facilities which should be shared by a cluster of terminals.

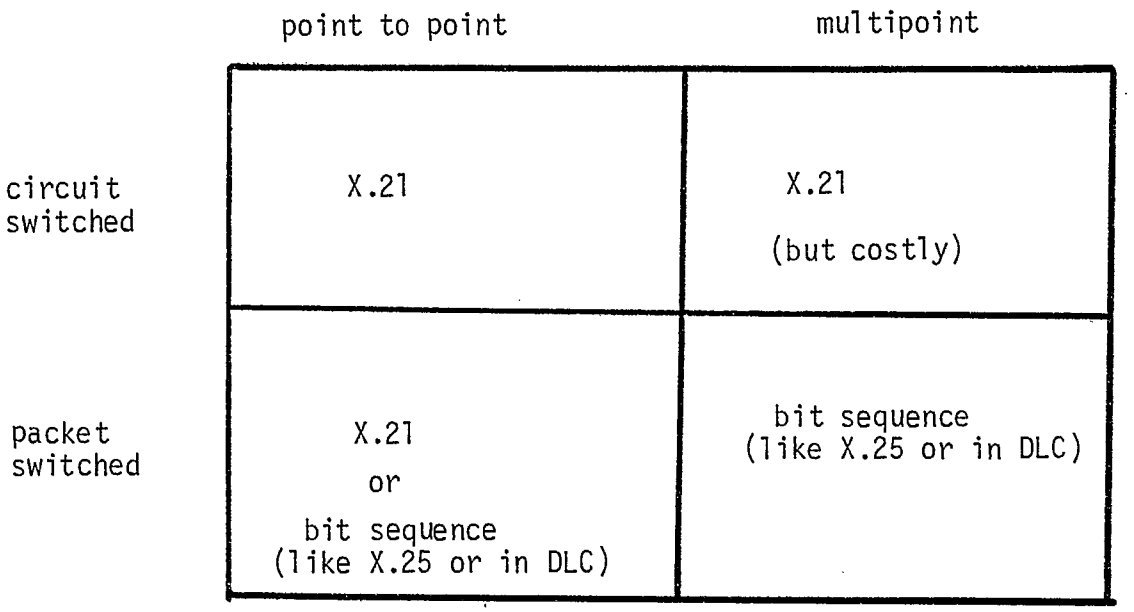


Figure 3.1 Comparison Between X.21 and Bit Sequence Based Call Control

4. A CLASSIFICATION OF FRAME MODE TERMINALS

4.1 Location of Call Control

We have seen that interworking of FDTEs and PDTEs on a packet network and, indeed, with FDTEs on a circuit network, can be achieved during the data transfer phase of a call by suitable definition of anchor points. Most proposals for the frame mode interface, in fact, specify some HDLC class of procedure in the expectation of standard interactions during this phase. The fundamental question, therefore, is that of call control and its location with respect to the data link control. There appear to be three distinct choices which will be discussed in detail in following sections.

Call control above data link control (Method A, Figure 4.1) is the first choice. In this a simplified version of the X.25 level 3 would be used for preparation of call setup and cleardown packets which would be exchanged with the network through HDLC. Once a call is established, all exchange of data packets would be performed using HDLC alone.

Call control within the data link control (Method B, Figure 4.2) is a second choice. In this configuration, all call control information is exchanged with the network by frames in HDLC format and within the HDLC elements of procedure. Data transfer would take place after exchange of call control and set mode commands.

Finally, call control below the data link control (Method C, Figure 4.3) is typified by X.21, in which elements of call setup and cleardown are associated with voltage changes on specific leads of the interface.

Addressing and call progress signals are not necessarily in the format of the data link control frames. Set mode commands and data transfer take place only after the call is established.

4.2 Call Control Above Data Link Control (Method A)

Call control above data link control (Figure 4.1) involves the exchange of special control packets with the DCE to set up and clear a call. These packets are carried through HDLC as normal I frames, as in X.25. During the call connected phase, data packets are also exchanged directly as I frames, without the multiplexing and flow control found at level 3 of X.25. The sequencing, flow control and error control functions of the data link control are adequate to regulate the data transfer.

Such an arrangement has a number of compelling advantages for single-channel DTE access to packet networks. Perhaps the most important from the viewpoint of network support of existing intelligent terminals is its relative independence of specific data link control procedures. Point-to-point versions of HDLC, such as LAP A and LAP B, can be replaced with unbalanced, multipoint protocols. Industry standards such as BSC and SDLC are equally acceptable. Even proprietary data link controls such as DEC's DDCMP could serve if the networks wished to support them; this would be particularly attractive to users owning PDP-11's equipped with the DMC-11 link control peripheral. The actual DLC used would be transparent to the call control functions, and network would become accessible to a much wider class of terminals than at present. There would be nothing to prevent interworking of IBM 3270 FDTEs, PDP-11 FDTEs and full X.25 PDTEs. Figure 4.4 sketches the open nature of the connections that could be allowed.

Another considerable advantage of this organization is its simplicity. Because a DLC is used for transmission of call control packets, the

designer can assume that all exchanges are error-free and in sequence. No retransmissions should be necessary. The implementer's job is made easier also by the fact that only a small amount of X.25 level 3 function need be developed; the fractional increase in required memory space above the basic HDLC module is minimal. In other words, placing call control over data link control leads to development of only a small amount of relatively simple code.

A fascinating aspect of the simplicity issue is the fact that, for intelligent terminals with this call control arrangement, the call state diagram can be maintained in the operator's head, rather than in the FDTE software. Messages equivalent to X.25 call control packets, when displayed on the screen, would be sufficient to allow the operator to keep track of call progress.

One minor disadvantage of call control over the DLC is the requirement for a dedicated octet in the I field of transmitted information frames to distinguish between call control and data packets - in other words, a way of achieving out-of-band signalling. This fact of life is characterised as a disadvantage here only insofar as it uses up a byte which could otherwise be used for data or not transmitted at all. Actually, more than one byte is involved; if this organization is adopted there will be overwhelming pressure for wholesale adoption of the X.25 packet structure, with the logical channel number, P(S), and P(R) fields ignored. This amounts to three wasted bytes in a data packet, or about a 1% overhead on full packets.

The major drawback to call control over DLC is that it cannot be used on circuit networks, for reasons discussed in Section 2.4. Nevertheless, this location for call control can be seen to provide great flexibility in accessing packet networks since it allows both point-to-point or multipoint connections, and use of any DLC which is mutually agreeable to the customer and the network administration. We predict also that it will be an early and inevitable offering of the packet networks (and Datapac in particular)regardless of the outcome of CCITT deliberations.

4.3 Call Control Within Data Link Control (Method B)

Call control within data link control means that call set up and cleardown requests are exchanged as distinct frame types within the DLC definition rather than, as in X.25, as special contents of ordinary data-carrying frames. In other words, the "out-of-band" indication needed for call control is supplied by the same method used to distinguish other DLC frame types: a special bit pattern in one of the frame header bytes. The call control frames are carried across the access link in the standard DLC envelope with its delimitation, transparency and error detection functions. This method has been proposed frequently in submissions to CCITT (See Appendices). The Canadian proposal falls into this category, as does any proposal which performs call control by frames which are required not to conflict with recognized frame types of the DLC.

Perhaps the best that can be said of this organization is that it is equally ill-suited to circuit switching and to packet switching. It is not compatible with any existing or proposed circuit network. It is also more difficult and less general than call control above DLC for access to packet networks.

To begin, it is usually difficult and expensive to modify a working DLC package. In the case at hand, we would have to add another level of retransmission error control for call information along with existing set mode, reset and information frame levels. There is no reason to suspect that the resulting software would consume less memory than that of call control above data link control - it may take up more space and it is certainly more difficult to write.

Another objection is that it locks all FDTEs into a specific DLC for network access (with some version of HDLC as the obvious choice). This inflexibility compares unfavourably with the independence from DLC found in call control above data link control for access to packet networks.

Finally, we note that this method has the same major virtue and major drawback as Method A. It allows multipoint access to packet networks but precludes access to circuit networks (see Sections 2.4 and 2.5 and Chapter 3).

4.4 Call Control Below Data Link Control (Method C)

Call control below data link control implies that elements of call establishment and clearing involve very primitive actions, such as voltage changes on a lead, to obtain out-of-band signalling. Further, any bit sequences exchanged during call setup or cleardown need not be in the same format as the DLC frames (although it certainly helps). The principal (and only) example of this technique applied to switched data networks is X.21, unless one includes calls established over the voice network. The DLC is idle during call setup, and the exchange of link restart (set mode) commands occurs only during the data transfer phase.

Perhaps the most attractive feature of this configuration is that both circuit and packet networks can comfortably offer this type of interface. CNCP, in fact, presently offers an X.21-like interface in which a single extra addressing digit can select real circuit service (InfoExchange) or virtual circuit service (InfoCall), at call setup time. Although InfoCall is not an X.25-like packet interface, it can be considered as a "pure virtual circuit" without extraneous features of DTE/DCE data link control and flow control. These latter functions take place on an end-to-end basis in both services. A more conventional packet network with an X.21 call control interface would give the DLC local significance only, that is, over the DTE/DCE access line.

Another feature of this placement of call control is its independence of data link control procedure. Any DLC is permissable during the data transfer phase, though of course the two anchor points (FDTE/FDTE for circuit switching, FDTE/DCE for packet switching) must use compatible protocols.

A final point in favour of this configuration is that it is easy to implement - at least more so than Method B.

There are two principal disadvantages. The first, and obvious one, is that multipoint access by this technique to packet or (especially) circuit networks is exceedingly difficult without the development of special, as yet non-standard, hardware. A less obvious problem of X.21, in particular, is its asymmetry. This feature precludes loopback testing of software, a technique which can be used with HDLC LAPA and with X.25. It also rules out interaction through a leased line unless a permanent call connected state is assumed. Finally, X.21 provides no address information to the recipient of incoming calls. There is therefore no basis on which to selectively refuse calls, and no way that a gateway for internetwork calls can be constructed without being buried in the software of the nodes themselves.

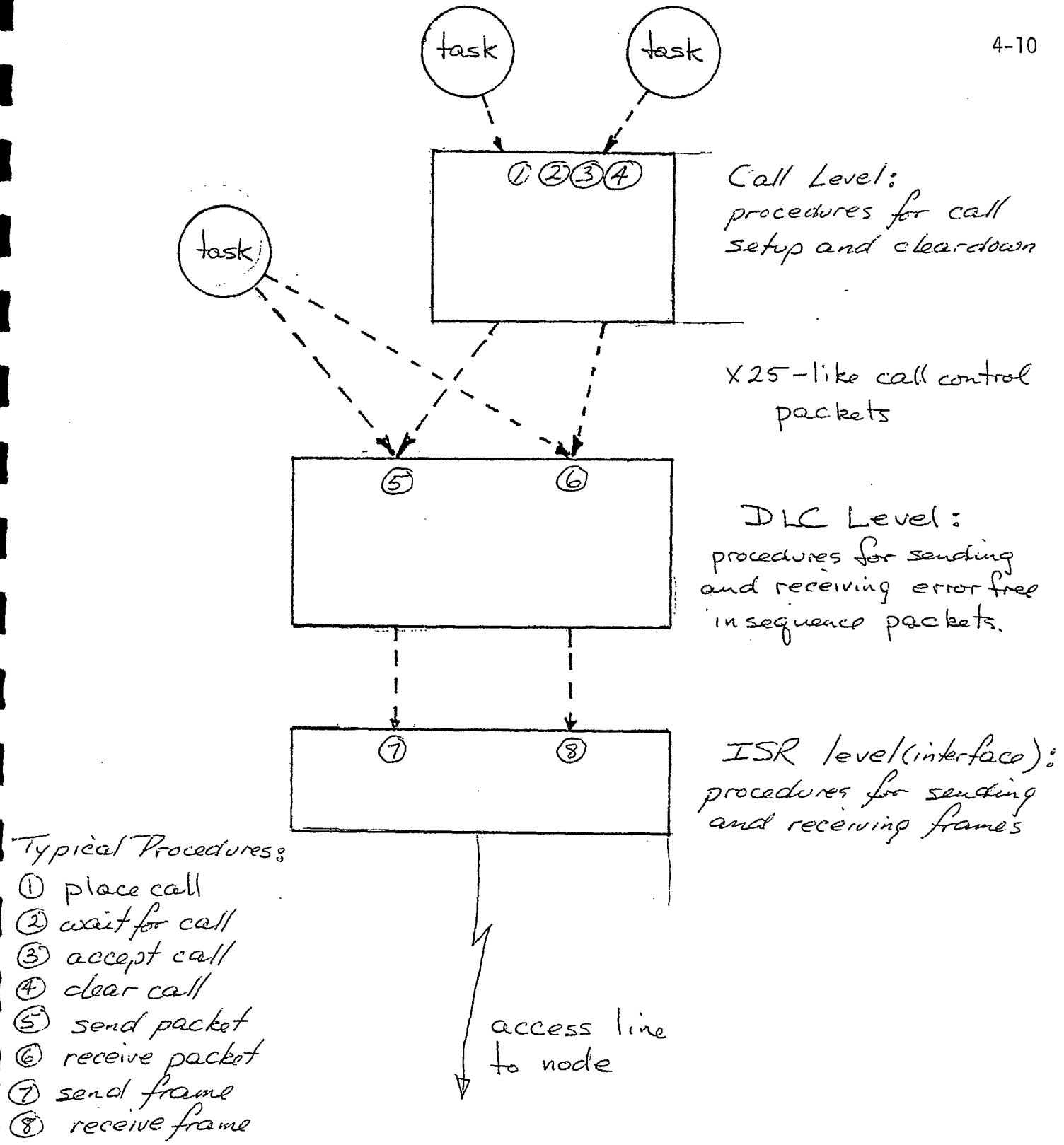


Figure 4.1 Method A Call Control

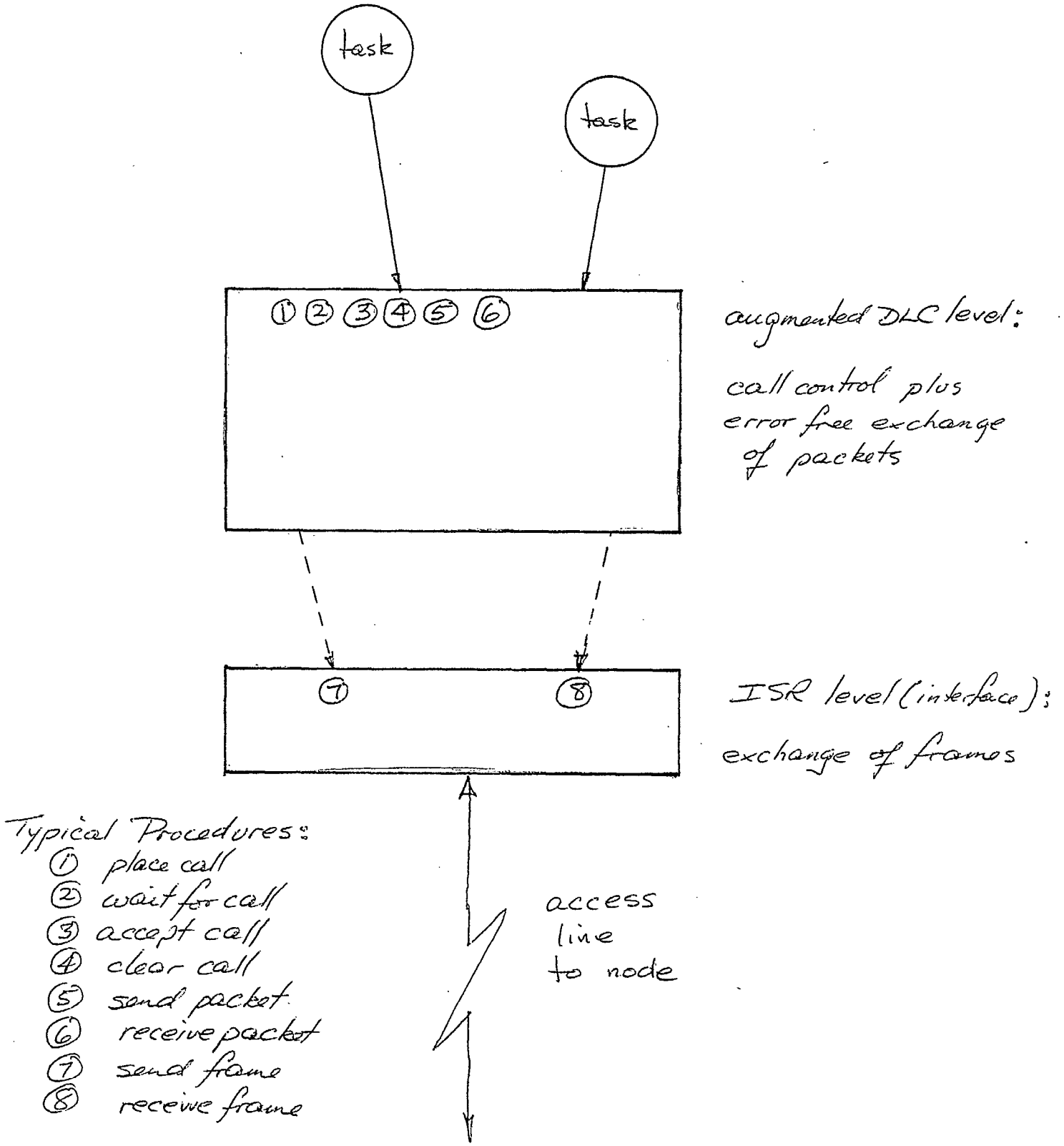


Figure 4.2 Method B Call Control

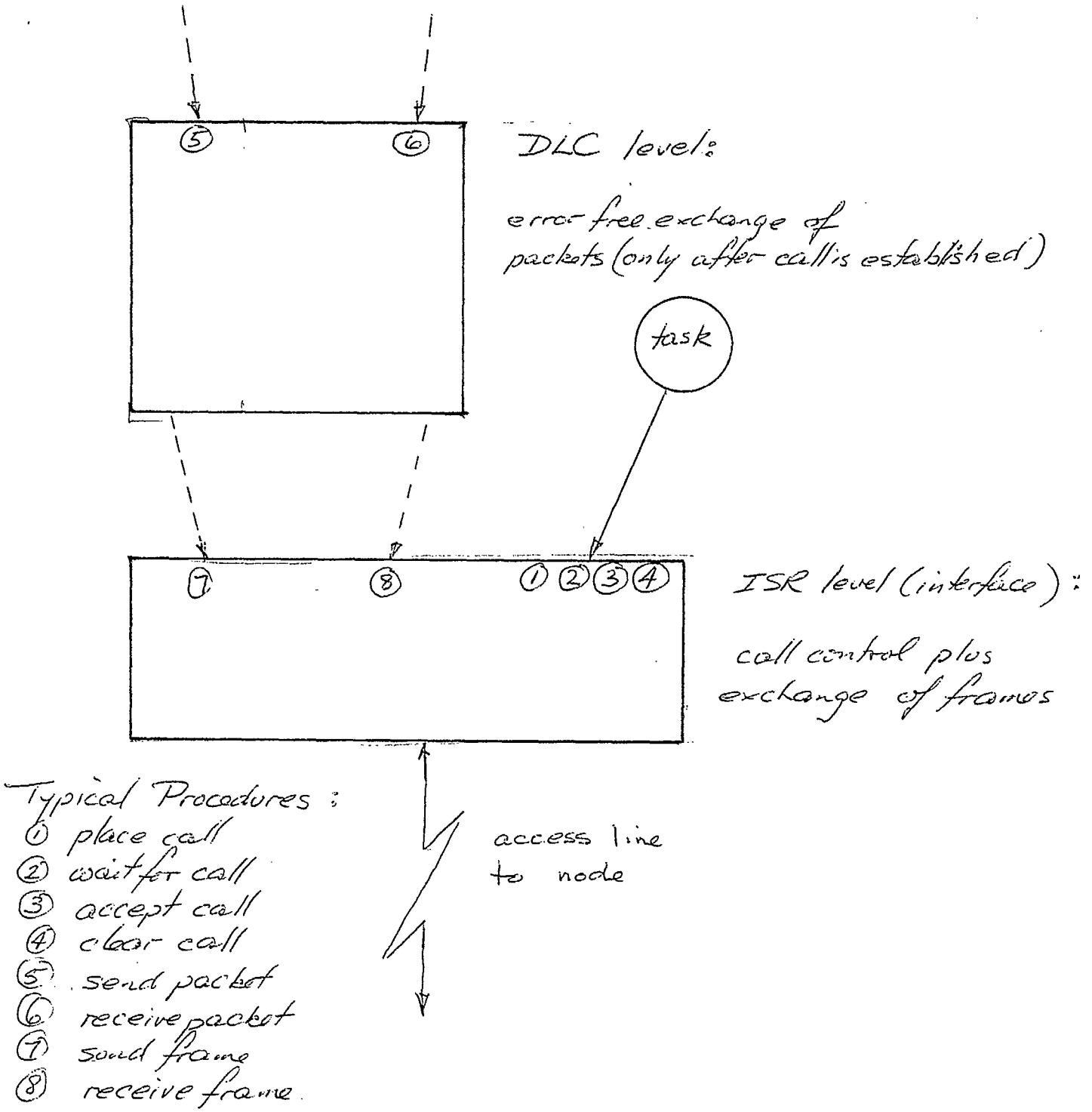


Figure 4.3 Method C. Call Control

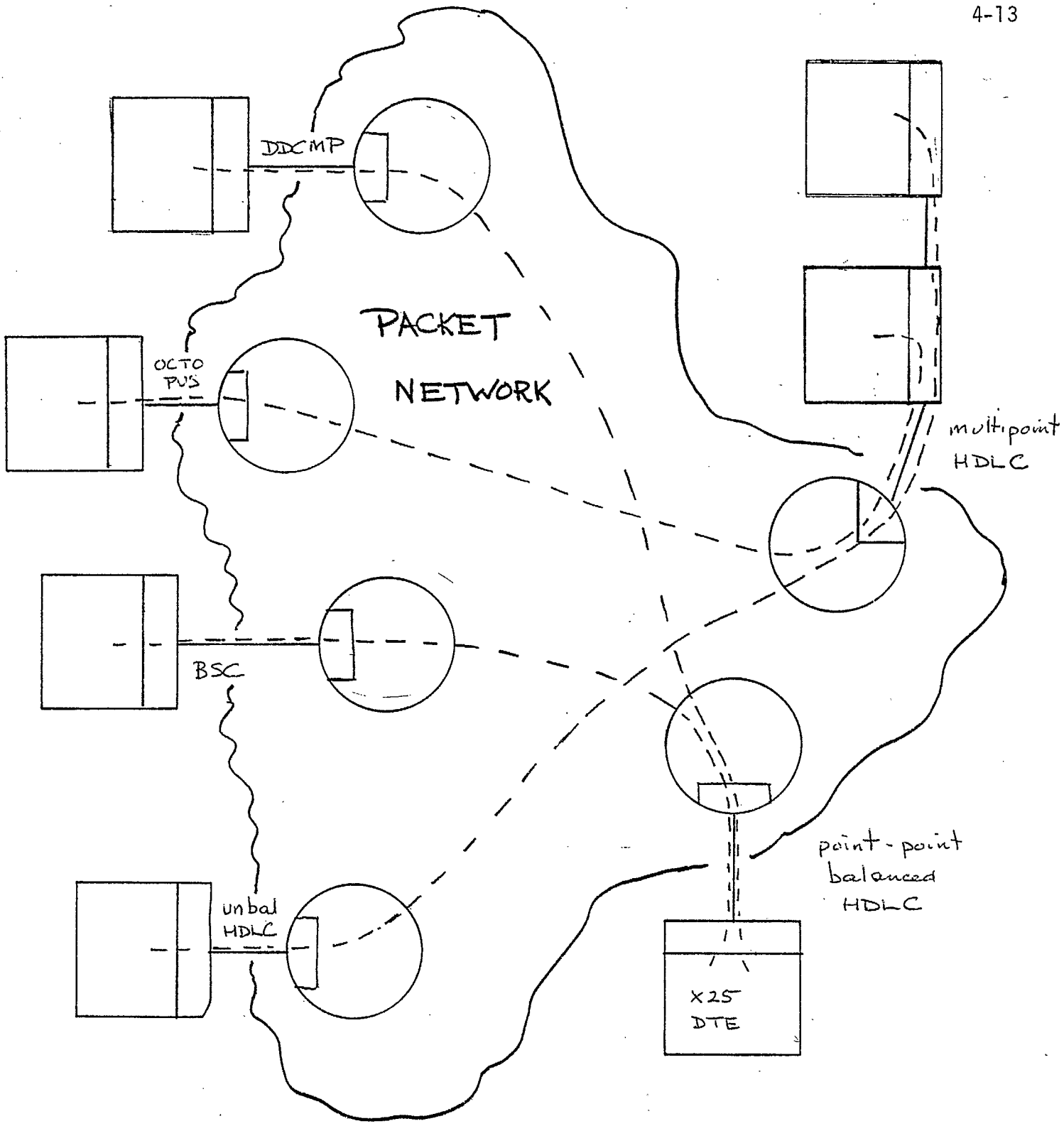


Figure 4.4. Method A Mixtures

5. INTERWORKING

It could be argued that total uniformity of call control methods is of secondary importance provided that the various FDTEs can interwork with each other and with PDTEs during the data transfer phase. Interworking should not present a problem with suitable definition of anchor points (Chapter 2). This chapter examines some of the awkward areas in call control within such an inhomogeneous population of DTEs located in the same or different networks. In addition, it is shown that a useful FDTE for both circuit and "transparent packet networks"[1] can be defined with no change to existing standards.

Both Method A DTEs (call control above data link control) and Method C DTEs (call control below data link control) will be considered. They will be represented by X.25 and X.21, respectively. Method B DTEs (call control within data link control) will be assumed not to exist, for the reasons given in Chapter 4.

We will consider two requirements in addition to the basic establishment of a call. First is that the called DTE be able to discover the identity of the calling DTE. This is automatic in X.25, since the "incoming call" packet contains this information. In X.21, of course, a second exchange is necessary following call establishment. The second requirement is that it be possible to place internetwork calls. We will assume that a global numbering

scheme exists and that the network will automatically route an internetwork call to the appropriate gateway.

5.1 Through a Packet Network

Consider a packet network supporting both X.25 and X.21 interfaces. We will examine the asymmetries in call control among the four pairwise permutations.

First, when X.25 DTEs contact each other, identification of the calling DTE is included in the call setup mechanism. When the X.25 DTE calls an X.21 DTE, on the other hand, typically the first packet it transfers would contain its network address. Consider the reverse situation next, in which the X.21 DTE calls the X.25 DTE. The first packet transferred by the calling DTE would typically be an identification message (since this would be required when it calls another X.21 DTE). The called X.25 DTE, however, would not be prepared for such a first packet, having already obtained calling address during call setup.

We have a strange collection of procedures already, without yet considering internetwork calls.

5.2 Through a Circuit or Transparent Packet Network

Only Method C techniques (exemplified by X.21) are appropriate in this situation. At least in intra network calls, therefore, we can expect uniformity of procedure.

By convention the first bit sequence transferred from calling to called DTE after call setup will be caller identification. This information can be conveyed in a frame before the DLC is initialized or in a packet (an I frame) after DLC initialization. The former choice is somewhat more general in that it is not necessary for the terminals to communicate through a DLC at all.

5.3 Through a Tandem Circuit/Packet Connection

This is potentially the most troublesome configuration, since it involves questions of internetworking, mixed services and frame mode access. To simplify the discussion, the gateway will be assumed to have an X.21 interface to the circuit network and an X.25 or X.75 interface to the packet network, and the DTEs will be X.21 on the circuit net and X.25 on the packet net.

If a circuit net X.21 DTE calls the X.25 DTE on the packet network, two stages of addressing will be required: one to call the gateway through the circuit net and the other to specify the called DTE to the gateway. The addresses transferred in each stage will be identical. In addition the calling DTE identification must be supplied to the gateway. This complexity is in contrast to X.21 DTE actions when calling another terminal on the same network.

The X.25 DTE calling an X.21 DTE on the circuit net is more fortunate, however. The gateway will receive both calling and called addresses and can take appropriate actions on behalf of the calling DTE to supply the called DTE with identification information.

5.4 Convenient Interworking With Existing Standards

This section demonstrates a FDTE for circuit and transparent packet networks which has three significant features: first, it provides for exchange of identification among such FDTEs; second it facilitates "invisible" placement of internetwork calls to similar FDTEs on the other circuit networks and to X.25 DTEs on packet networks; and third, it is an end to end protocol, and therefore requires no new network interface to be supported by the carriers. In particular we will assume that packet networks support only X.25 access, and that circuit networks support only X.21 access.

The mechanism is simple: FDTEs on the circuit network go through up to three stages of setup:

- they establish the circuit by means of standard X.21
- the calling terminal transmits an X.25 "call request" packet containing both network addresses to the called terminal. The frame format will be that used in the X.21 call setup. The called terminal replies with an X.25 "call accepted" packet in the same frame format, or clears the X.21 connection;
- both terminals initialize the DLC.

Call clearing is accomplished by X.21 only. Error protection of selection and progress signals is achieved by the usual mechanism of discarding received frames failing the CRC check. If either party times out waiting for a response, it simply clears the call.

The virtues of this end to end convention for intra network calls are obvious. The called DTE has an early identification of calling DTE which, among other features, provides a basis for early termination of calls. It is independent of the DLC, so that any mutually agreeable flavour of HDLC or BSC, for example, could be employed. In fact, implicit in the calling terminal identification could be the data link protocol to be used, which may be of interest to hosts supporting multiple DLCs.

Next consider internetwork calls. The gateway is assumed to have only X.21 access to the circuit networks and only X.75 access to the packet networks. Now if one of the FDTEs on the circuit network places a call to an address external to its network,by the global numbering assumption the node will set up a circuit to the appropriate gateway. The next act of the calling FDTE is to transfer a call request frame, as usual, to the called DTE, which in this case is the gateway. The gateway, having received the call request, attempts to place the call in the next network, and returns a call connected frame to the calling FDTE if it is successful. Finally, the DLC between the FDTE and the gateway is initialized. Note that this procedure works regardless of whether the called DTE is another FDTE on a circuit net or an X.25 DTE on a packet network. Further, a similar mechanism allows X.25 DTEs to call FDTEs on a circuit network.

It has been shown that allowing limited inhomogeneity in the

DTE population (X.25 on packet networks and X.21 on circuit network with the FDTE convention described above) conveys a number of advantages of which the most important are "invisible" internetwork calls, and a protocol by which terminals on an X.21 network can identify the calling party.

A minor disadvantage in comparison with protocols like BASIC FDTE [2] is that the circuit net customer pays for connect time from the instant of establishment of his X.21 call to the gateway, even if the packet net destination terminal eventually refuses the call.

6. Conclusions

The following is a brief statement of the conclusions reached in this report.

ONE The principal objectives to be achieved by a FDTE can be stated in order of decreasing economic priority as:

- the ability of FDTEs and PDTEs to interwork in the data transfer phase of a call;
- a single uniform call control technique for both circuit and packet networks;
- simplified access to packet networks for single channel DTEs;
- multipoint access to a packet network.

The second and fourth objectives conflict.

TWO The first objective above is realized fairly easily for both intra and internetwork calls, so the focus of any FDTE investigation should be call control.

THREE The second objective is based on the increased expense of distribution and field service when a terminal manufacturer maintains several options. The alternative of supporting several switch-selectable call control methods in a single terminal is feasible with respect to memory space, since call control is relatively small [3] and memory densities are increasing, but adds to the software complexity and hence development time and maintenance cost.

FOUR The various approaches to definition of an FDTE can be classified by the location of call control with respect to data link control:

- Method A: above DLC;
- Method B: within DLC;
- Method C: below DLC.

Table 6.1 summarizes the properties of the three general methods. It can be seen that Method C (X.21-like) interfaces come closest to realization of all but the fourth objective above. Method A (X.25-like) call control methods provide greatest flexibility for access to packet networks and satisfy all but the second objective; they are unsuitable for use on circuit networks. Method B techniques, which include the Canadian and most other proposals before CCITT, provide the fewest advantages and the greatest rigidity.

FIVE No existing proposal satisfies all four objectives above, and only the Nordic proposal satisfies the first three. Even it results in nonuniformities in start-of-call actions when calling DTE identification and internetworking are considered. A companion report [2] presents a proposed interface which satisfies the first three objectives and facilitates exchange of DTE identification and internetworking.

SIX The experimental phase of the project will consist of implementations of Method A and Method C only.

	MULTI-POINT ACCESS?	FOR PACKET NETS?	FOR CIRCUIT NETS?	INDEPENDENT OF DATA LINK CONTROL CHOICE?	ESTIMATED IMPLEMENTATION DIFFICULTY	INTERNETWORK CALL CONTROL
METHOD A	YES	YES	NO	YES	EASY	ONE STAGE
METHOD B	YES	YES	NO	NO	LESS EASY	TWO STAGE
METHOD C	POSSIBLE BUT COSTLY	YES	YES	YES	EASY	TWO STAGE

Figure 6.1 Comparison of FDTE Approaches

REFERENCES

[1] Gregor V. Bochmann, "The Frame Mode DTE Interface," Report on Contract 02SU,36100-7-0304, Department of Communications, Ottawa, October 1977.

[2] J. K. Cavers, "BASIC FDTE: A New Frame Mode Interface to Digital Networks," Report on Contract OSU78-00182, Department of Communications, Ottawa, November, 1978.

[3] J. K. Cavers, "An Experimental Investigation of the Frame Mode Interface to Digital Networks", Report on Contract OSU78-00182, Department of Communications, Ottawa, November, 1978.

APPENDIX A

GLOSSARY OF ABBREVIATIONS

ISO	International Standards Organization
CCITT	Comité Consultatif International sur Telephones et Telecommunications
PPN	Public Packet Network
DTE	Data Terminal Equipment (i.e. the customer equipment)
DCE	Data Channel Equipment (i.e. the serving node)
CPU	Central Processing Unit (the part of a computer which executes intructions)
PDTE	Packet mode DTE equippped with full X.25
FDTE	Frame mode DTE (generic term for DTE which accesses a packet network without use of a full X.25 packet level; the focus of this study)
DLC	Data Link Control. The protocol which handles at least retransmission error control of blocks of data on a point to point or multipoint communication line. Usually also provides for sequencing and link initialization.
HDLC	High Level Data Link Control. A bit oriented DLC standardized by ISO; bears a close resemblance to IBM's SDLC and ANSI's ADCCP.
SDLC	Synchronous DLC. Supported by IBM.
ANSI	American National Standards Institute
ADCCP	Advanced Data Communication Control Procedure (another DLC)
BSC	Binary Synchronous Communication, IBM's old workhorse DLC.
BiSync	BSC

DDCMP	Digital Data Communication Message Protocol. DEC's DLC.
DEC	Digital Equipment Corp.
ADTE	FDTE employing Method A call control (over the DLC).
BDTE	FDTE employing Method B call control (within the DLC).
CDTE	FDTE employing Method C call control (below the DLC).
TDM	Time Division Multiplexing.

APPENDIX B

Summary of FDTE Proposal to ISO from Canada

Intended Application

- dedicated circuit, circuit-switched service, packet-switched service (virtual circuit and others);
- multipoint access;
- internetworked DTEs (including mixed circuit and packet);
- applicable to most synchronous DTEs.

Data Link Control

HDLC; class of procedure left unspecified

Call Control

- Method B
- exchange of HDLC UI frames in X.25 format and interpretation
- call setup in DM followed by set mode exchange (link setup)
- call clear also a UI frame in X.25 format (followed by DISC?)

Comments

- not suitable for access to circuit switched nets since call control based on bit sequences
- locks customers and networks into HDLC, hence not applicable to most DTEs, (e.g. BSC and derivatives).
- use of unnumbered frames (UI) for interrupts with retransmission on timeout can lead to reception of multiple interrupts when only one was intended.

APPENDIX C

Summary of FDTE proposal to CCITT from IBM Europe

Intended Application

- FDTE to FDTE through leased line, switched circuit or packet network;
- FDTE to X.25 DTE through packet network;
- multipoint access to packet net.

Data Link Control

- two way alternate channel operation
- HDLC in NRM with DCE the primary

Call Control

- Method B
- call establishment in up to four stages;
  - physical in the case of X.21 or switched analog link between DTEs
  - SNRM, VA
  - SIG, an HDLC command/response pair in X.25 call control format
  - XID (exchange ID) if tandem call, circuit to packet
- call clearing is in three stages
  - SIG
  - DISC, VA
  - physical clear

Comments

- will not work with leased lines or circuit switched connections, as claimed, since both DTEs are HDLC secondaries;
- if X.21 or circuit switched, SIG is unnecessary;
- locks network and customer into HDLC as the data link control;
- hopelessly complex for so little return.

APPENDIX D

Summary of FDTE Proposal to CCITT from Nordic Countries

Intended Application

- FDTE to FDTE through leased lines, circuit switched services, packet switched service
- FDTE to X.25 DTE through packet network with packet multiplex function

Data Link Control

- HDLC in Asynchronous Balanced Mode (LAPB), used only during data transfer phase. Originating DTE has address A, receiving has address B.

Call Control

- Method C
- X.21 to establish and clear the call;
- set mode exchange after call establishment (SABM) and before call clear (DISC);

Comments

- accomplishes goals within existing protocols;
- no multipoint access;
- modification to X.21 call control needed for leased lines;
- X.21 does not facilitate internetwork calls.

BASIC FDTE:

A NEW FRAME MODE INTERFACE

TO DIGITAL NETWORKS

BY

JAMES K. CAVERS

MICROPROCESSOR SYSTEMS DEVELOPMENT LABORATORY

FACULTY OF ENGINEERING

CARLETON UNIVERSITY

OTTAWA

A report prepared for the Department of Communications, Ottawa, under contract OSU78-00182. The opinions expressed herein are those of the author and do not necessarily reflect a position on the part of the Department of Communications.

November 15, 1978.

ABSTRACT

No existing or formally proposed standard for access to digital networks provides identical call control methods on circuit and packet networks, and facilitates internetworking and exchange of terminal identification. This report presents an interface which satisfies all these conditions. Its generality is achieved by restricting attention to call control functions only; data transfer functions are left for further agreement or standardisation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

A companion report [1] surveyed the major proposals for the Frame Mode interface to digital networks, both circuit switched and packet switched. Two significant conclusions were drawn in that study. First, the twin objectives of multipoint access and uniform call control on circuit and packet network, are incompatible. Second, the only hope for a call control method applicable to both circuit switched and packet switched networks lies in a class of procedures in which call control sits below data link control (termed Method C in that report).

Here we present the required "universal interface". Termed BASIC FDTE, it is an X.21 derivative which provides a great deal of flexibility in a very simple protocol. The design approach was to devise the most primitive actions which would allow uniform call control techniques, both intra- and internetwork, for both circuit switched and packet switched networks. Specifically,

BASIC FDTE does:

- specify procedures for call control (setup and clearing);
- give access to circuit and packet networks in an identical way;
- facilitate internetwork calls.

BASIC FDTE does <u>not</u>:

- proscribe or prescribe any actions during the data transfer phase: frame formats, error control, flow control, etc. are left open;
- provide for multichannel or multipoint access.

BASIC FDTE is a new access protocol which combines the important features of X.25 and X.21 call control. As such it is incompatible with any existing or planned network. The question, therefore, is:

Why adopt BASIC FDTE? Because:

- Use of the interface leads to signal major changes in call status allows uniform call control techniques on circuit and packet networks. By contrast, methods based on special frames or bit sequences to clear a call cannot be used over real circuits because of the requirement on real circuits that they be transparent.
- It does not constrain customer or network to use a particular DLC or higher level protocols. These are left for further standardization or for mutual agreement between customer and network administration in the case of virtual circuits, or customer and customer in the case of real circuits. BASIC FDTE therefore allows public switched networks to be incorporated into frameworks like IBM's SNA and DEC's DECNET.
- The X.25-like format of addressing and call progress signals allows simple and uniform treatment of internetwork calls.
- Although designed for synchronous terminals, it can be adapted easily to serve asynchronous terminals.
- It is a symmetric protocol (unlike X.21) and can therefore be used over leased lines as easily as it can over either type of switched network.
- It provides the customer with a means for selective refusal of incoming calls, a feature not provided by X.21.

- It can be used by such diverse DTEs as fax reader/printers and vocoders, and is not restricted to use by a special class of synchronous terminal (e.g. HDLC in some class of procedure).

2. BASIC FDTE PROTOCOL

2.1 Mechanical and Electrical

The mechanical and electrical characteristics of the interface are as specified in X.21 or X.21 bis. In particular, four of the interchange circuits are:

C control (to DCE)

I indication (to DTE)

T transmit (to DCE)

R receive (to DTE)

2.2 Interface Procedures

The state diagram of the interface is shown in Figure 2.1. Like X.21, it combines voltage changes on interface circuits and exchange of bit sequences.

The ready state (0) is characterized by C=off, I=off, and T and R are in the "mark hold" condition of binary one. No call exists and the interface is ready to establish one.

If the DTE wishes to place a call, it asserts C=on, thus placing the interface in state 1 (call request). A response by the network of I=on places the interface in state 3 (ready for address). Similarly if the network wishes to indicate the presence of an incoming call, it sets I=on (state 2, incoming call), and a response from the DTE of C=on puts the interface in state 3. Note that the preliminary handshake on both outgoing and incoming calls results in state 3 (ready for address). Call collisions are resolved in the next stage of handshake.

From state 3, the DTE trying to place a call moves to state 4 (DTE waiting) by transferring across circuit T selection signals in the format given in section 2.3. A "Call connected" progress signal returned by the network over circuit R completes the handshake with the interface in state 6 (data transfer). Any response other than "call connected" will cause the DTE to clear the call. Alternatively if the network wishes to continue setup of an incoming call from state 3, it transfers across circuit R an "incoming call" signal in the format of section 2.3, placing the interface in state 5 (DCE waiting). A "call accepted" response from

the DTE over circuit T completes the handshake and the interface is in state 6 (data transfer). Any other response from the DTE causes the DCE to clear the call. There is therefore no "call collision" state required since collisions automatically result in clearing of both incoming and outgoing calls.

Call clearing is accomplished simply by setting C and I to "off". The DTE can initiate clearing from any state (except 0 or 2) by setting C=off thereby placing the interface in state 7 (DTE clear request). The network response I=off places the interface back in state 0 (ready). In a symmetric fashion, the DCE can clear the call by setting I=off (state 8, DCE Clear indication), and the DTE response of C=off returns the interface to state 0 (ready).

Limits on the duration of certain states may be prescribed by the network. For example, state 0 may have a minimum duration to ensure that both DTE and DCE are agreed on the state, and have not missed the ready state in the other party's eagerness to set up the next call. State 3 (ready for address) may have a maximum duration to account for selection signals being wiped out by transmission errors.

2.3 Signalling Formats

Selection and call progress signals are exchanged across the interface. In defining these it is important to distinguish between the information content and its format, on one hand, and the envelope and its format, on the other. The latter simply realizes the frame structure functions of character synchronization, frame delimitation, transparency (if required) and error detection. In any case there is never any conflict between these signals and the frame types and formats used during the data transfer phase, since the interpretation is determined by the interface state.

2.3.1 Information Format An important feature of the protocol is the use of an X.25-like structure for the "call request" and "incoming call" selection signals. Specifically, these contain the network addresses of both the calling and called DTEs, and the two selection signals have identical format. We shall see that this symmetry aids in the setup of internetworking calls.

Having adopted the content of X.25 "call request" and "incoming call" packets we may as well adopt the format (Figure 2.2), and thereby avoid format conversions when crossing an X.25 network boundary. If simplicity were the only criterion, though, a preferable format would be the one described in a Pascal-like notation in Figure 2.3. Certainly less logic would be required at the interface if this format were used.

Call progress signals are the next to consider. The simplest approach is to allow only one such signal: "call connected" and "call accepted". Again these two have identical formats (Figure 2.2) but are given different names depending on direction: to DTE and to DCE, respectively.

If the outgoing call is unsuccessful for any reason, or if the incoming call is to be refused, the appropriate side of the interface simply returns its lead (I or C) to off.

Thus only two information formats are required: "incoming/call/request" and "call connected/accepted". If this method of indicating inability or unwillingness to complete call setup seems unnecessarily brutal, the reader should remember that this is a BASIC FDTE. Certain network administrations may wish to supply their customers with a richer set of call progress signals, but all networks and terminals must use BASIC FDTE as an operational subset.

2.3.2 Envelope Formats The envelope provides a carrier for the information of section 2.3.1. Typical envelopes are the HDLC frame defined in ISO DIS 3309.2, BSC, transparent BSC, DDCMP, etc. It is important to note, though, that the envelope used during call setup can be different from that used by the DLC during data transfer, since the interface state (Figure 2.1) removes any ambiguity.

In the interest of simplicity and greatest accessibility, we reject frame structures based on bit stuffing for transparency and delimitation (e.g. HDLC, SDLC, ADCCP etc.). These all require special hardware which is installed in only a few existing terminals. A character-oriented envelope format, like BSC, seems advisable. But what of the terminals using bit-oriented protocols? Fortunately, of the few terminals which do so, most employ a communications controller chip to perform the envelope function.

Almost without exception, these chips are programmable (i.e. software-settable) to either character-oriented or bit-oriented structures.

Accordingly, the BSC frame (Figure 2.4) is proposed as the envelope format.

3. INTERNETWORK CALLS

The companion report [1] examined the problem of interworking frame mode DTEs with packet mode DTEs, and the problem of internetwork calls between packet networks, between circuit networks and between circuit and packet networks. It was shown that there are no major problems during the data transfer phase of a call, providing DLC anchor points are suitably chosen.

Call control procedures were not uniform, however. A DTE on an X.25 network can contact a DTE on an X.21 circuit network with a single stage of call control, since the gateway can place the circuit call on behalf of the originating DTE. The originating DTE would have to identify itself to the destination DTE after call setup, because X.21 does not provide the calling DTE address. Calls to other terminals in the X.25 net, on the other hand, require no such secondary identification. Further, a terminal on an X.21 net contacting a terminal on an X.25 net must go through two stages of addressing: first to contact the gateway; and second, to tell the gateway which X.25 DTE is to be called. By contrast, calls to other terminals on the X21 net involve only one stage of addressing.

This chapter demonstrates that BASIC FDTE exchanges sufficient information for an absolutely uniform treatment of intra- and internetwork calls, through circuit or packet networks.

Consider first a BASIC FDTE on a packet network. To contact another terminal (BASIC or X25) on his own network, he emits a selection signal (Chapter 2) containing his own address and that of the called DTE, and with luck his call will be connected. To contact a DTE on another network he again emits the selection signal. This time it may be necessary to

prefix the called DTE address with other information according to some global numbering scheme, eg.:

network id. area code. address

The network, recognizing this as an internetwork call, routes the "call request" to the appropriate gateway. The gateway receives the signal as an "incoming call" containing the called DTE address, and then places the call on the next network by passing on the signal virtually untouched, even if the next network is a circuit net with a BASIC FDTE interface to the gate. Evidently the actions are very similar to the X.75 internetwork protocol. It is clear that an arbitrary number of network can be traversed in this fashion, and that the "call accepted/connected" signals or call clears can ripple back in reverse order till they finally reach the calling DTE.

A BASIC FDTE on a circuit network is next to be considered. Intra-network calls are established in a fashion identical to that used for intranetwork calls in a packet net, with exactly the same information crossing the interface. When an internetwork call is to be setup, on the other hand, the circuit network will route the call to the appropriate gateway. The gateway, equipped with the BASIC FDTE interface, will receive an "incoming call" signal with both calling and called DTE addresses. As before it can then place the next network call on behalf of the originating DTE, as described above.

Uniform call control procedures have been demonstrated.

4. SUMMARY

BASIC FDTE is a simple protocol by which terminals can gain access to switched digital networks. It defines only call control functions, and is therefore applicable to a very wide range of terminals - fax reader/printers, vocoders, and interactive terminals are some examples - and is a suitable interface for both circuit and packet networks. Data transfer functions are left for a separate agreement between the entities involved in the call; in fact, if they were included in the FDTE interface the range of application would be restricted.

BASIC FDTE is an improvement on X.25 in that it can be used as an interface to circuit, as well as packet networks. It is an improvement over X.21 in that internetwork calls are facilitated.

BASIC FDTE can be considered a universal interface for point to point, single channel DTEs.

REFERENCES

[1] J. K. Cavers, "A Critical Review of Proposals for the Frame Mode Interface to Digital Networks", report on Contract OSU78-00182, Department of Communications, Ottawa, November 1977.

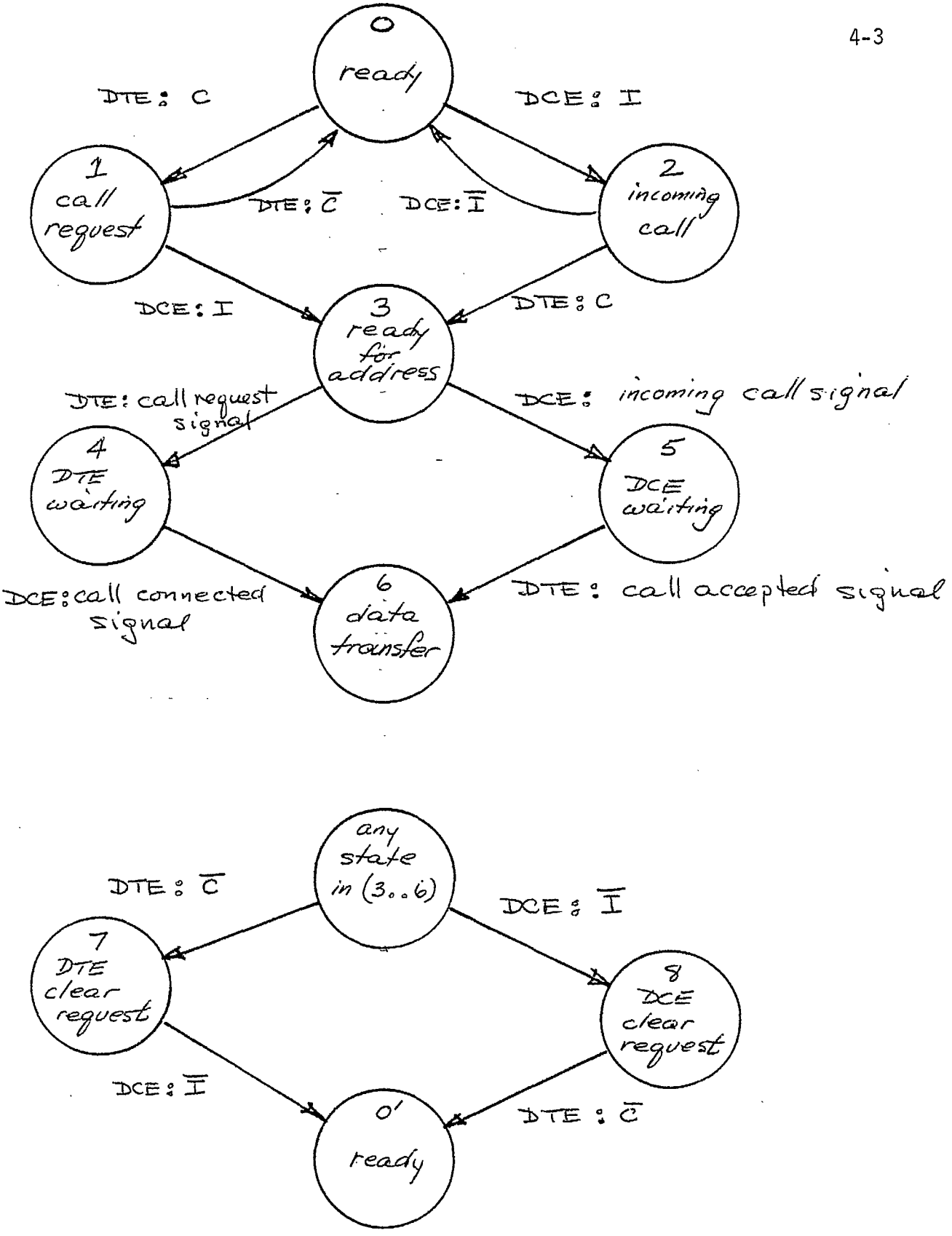
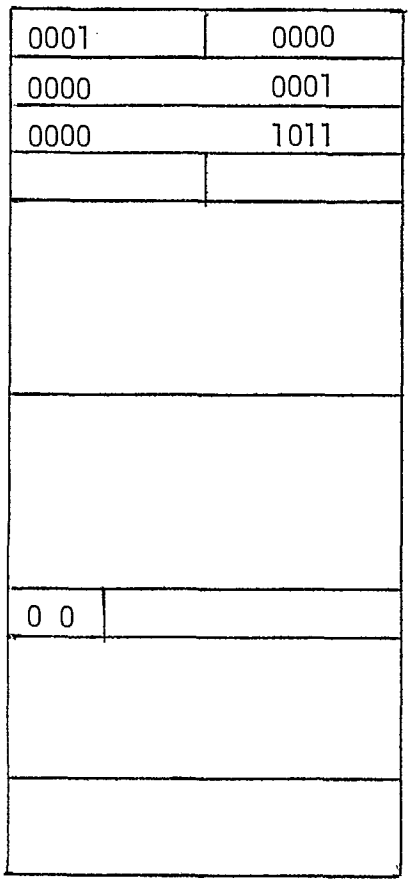


Figure 2.1 BASIC FDTE STATE DIAGRAM



Channel 1

"call request/incoming call"

address bngth calling/called DTE

address called DTE

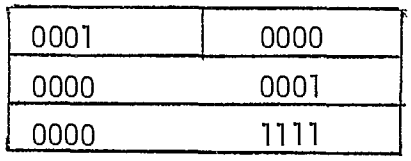
address calling DTE

facilities field length

facilities

call user data

Call Request and Incoming
Call Packet Formats



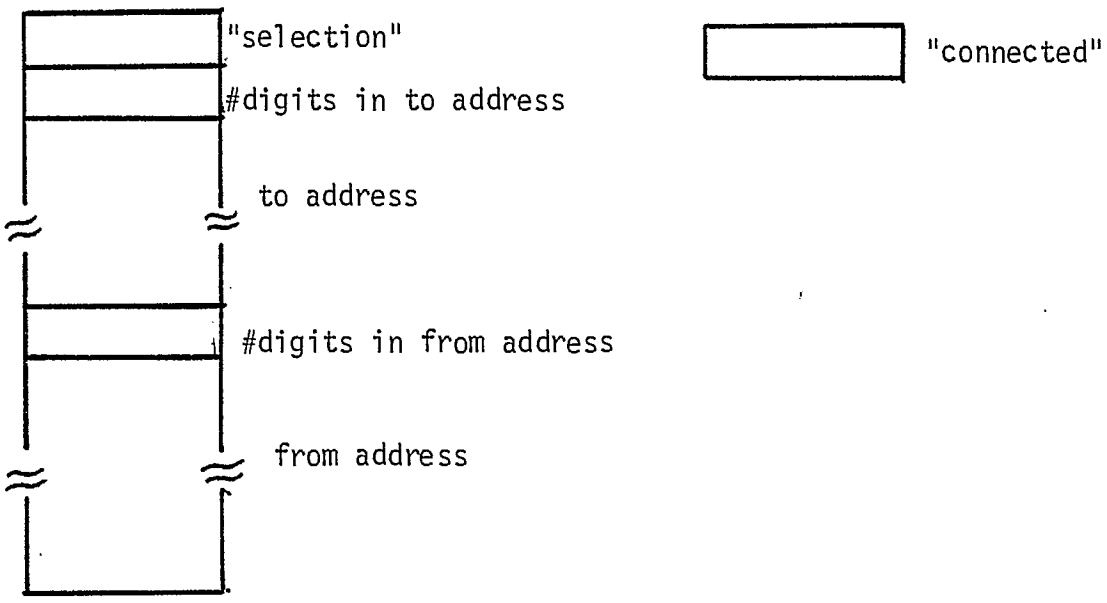
Channel 1

"call accepted/connected"

Call Accepted and
Call Connected Packet Formats

Figure 2.2 INFORMATION FORMAT OF CALL CONTROL SIGNALS

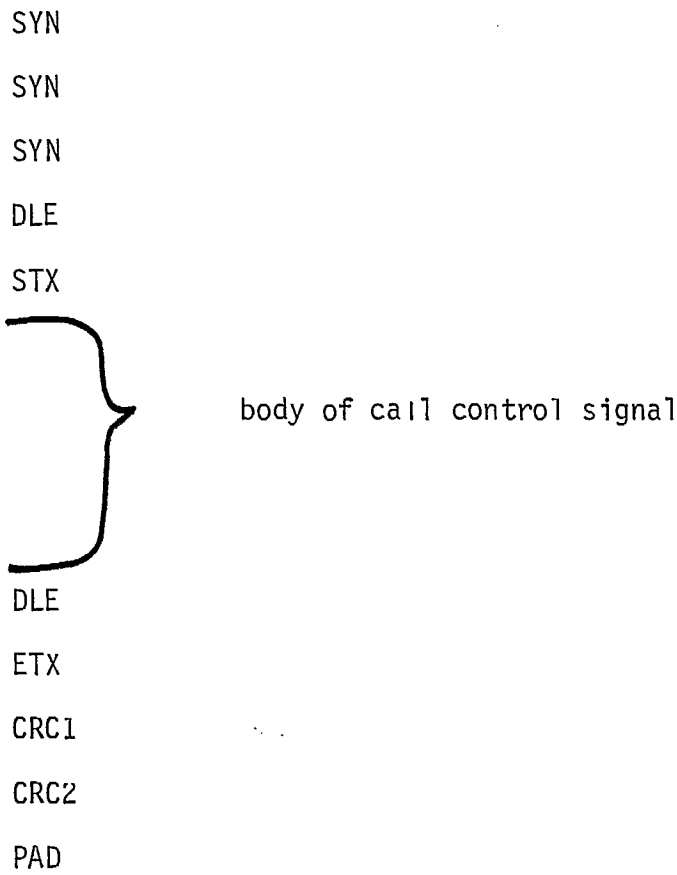
```
type signal = record
      kind: (selection, connected);
      case kind of
          selection: (to length: byte;
                      to address: array [1...length] of byte;
                      from length: byte;
                      from address: array [1...from length] of byte)
          end
      end;
```



Call Request, Incoming Call Call Accepted, Call Connected

All numeric values coded in natural binary, one per byte.

Figure 2.3 A Simplified Format for Call Control Signals



Notes: - Naturally occurring DLEs in body are doubled

- CRC is $x^{16} + x^{15} + x^2 + 1$ initialized to zero remainder
- CRC is computed over body (except transparency DLEs) and ETX

Figure 2.4 ENVELOPE FORMAT OF CALL CONTROL SIGNALS

AN EXPERIMENTAL INVESTIGATION OF THE FRAME MODE DTE

BY

JAMES K. CAVERS
MICROPROCESSOR SYSTEMS DEVELOPMENT LABORATORY
FACULTY OF ENGINEERING
CARLETON UNIVERSITY
OTTAWA

A report prepared for the Department of Communications, Ottawa, under contract OSU78-00182. The opinions expressed herein are those of the author and do not necessarily reflect a position on the part of the Department of Communications.

November 15, 1978.

ABSTRACT

Several prototype frame mode DTEs were constructed and compared on the basis of functionality, memory space and implementation difficulty. These FDTEs included a skinny X.25 and an X.25 variant without packet level flow control, as well as two based on X.21-like procedures. To demonstrate that dissimilar DTEs could interwork, an experimental internetwork connection was set up between the Infoswitch circuit switched service and a Datapac-equivalent X.25 network.

CONTENTS

LIST OF ILLUSTRATIONS

...continued

1. INTRODUCTION

1.1 Review

The frame mode interface to digital networks [1, 2, 3] has received recent attention, as various individual and corporate authors attempt to define what it is and how it should operate. There is general (though not universal) agreement on two objectives: that it provide a simpler access method to packet networks than X.25 for single channel DTEs; and that both circuit switched and packet switched networks be able this interface comfortably. A terminal which implements the DTE side of such an interface is termed a Frame mode DTE (FDTE).

A detailed analysis of the FDTE is contained in a companion report [1], which classifies FDTEs by the location of the call control function with respect to the data link control (DLC) function: the terms Method A, B, and C refer to call control above, within and below data link control, respectively. Figure 1.1 summarizes the properties of the three general methods. One conclusion of that report was that only Methods A and C showed sufficient promise to warrant experimental investigation.

1.2 The Project

In broad outline, the project consisted of the implementation of a number of candidate FDTEs and the internetworking of DTEs on

Infoswitch and a Datapac-equivalent network. The objectives were to assess memory requirements, implementation difficulty and functional capability of the FDTEs.

In more detail, two of the DTEs were of the Method A type, with call control performed by special packets carried through the DLC in the same fashion as data packets. One was X.25 itself (which is not strictly a frame mode DTE), implemented in a Datapac-compatible single channel version in order to ascertain its real implementation difficulty. The other Method A terminal was essentially X.25 with packet level flow control stripped out. Comparison of the two gave an estimate of the difficulty of implementing the redundant second level of flow control in single channel X.25.

Two Method C DTEs, in which call control involves manipulation of specified interface pins, were also built. They were in fact implementations of the X.21-like procedures specified by Infoswitch for use on the InfoExchange and InfoCall services (circuit and packet mode respectively). One of these FDTEs simply initialized the DLC after X.21 call setup. The other went through a second stage of address exchange after X.21 call setup and before DLC initialization; this extra level facilitated identification of the calling DTE and a uniform treatment of intra- and internetwork calls.

The same application was used in all DTEs for ease of comparison; truly "conversation" terminals in which messages were exchanged on a real time basis between DTE operators. It functioned rather like an

interactive high-speed Telex with local editing support.

The internetworking experiment employed the Infoswitch circuit switched service IES and a Datapac - equivalent network to confirm that the FDTEs could interwork over a tandem circuit/packet switched link.

1.3 Outline of Report

Chapter 2 describes an existing X.25 interface and gives a memory space breakdown for purposes of comparison with later implementations. It can be considered to be a benchmark network access interface.

Chapter 3 is concerned with Method A interfaces. Two are described, one a special single channel X.25, and the other a true FDTE, in which flow control is managed by the DLC. Again both the software architecture and memory space requirements are outlined. This chapter also introduces the application level which is common to all the FDTEs constructed.

Chapter 4 deals with the two Method C interfaces, both of which were used to access Infoswitch. Architecture and memory space are given for bare X.21, and an X.21 augmented by a layer of end to end protocol.

Chapter 5 describes the experimental facility for internetwork calls between an FDTE on Infoswitch and an X.25 on a Datapac-equivalent network.

Finally the conclusions drawn from our experience in implementing several candidate FDTEs are presented in Chapter 6.

	MULTI-POINT ACCESS?	FOR PACKET NETS?	FOR CIRCUIT NETS?	INDEPENDENT OF DATA LINK CONTROL CHOICE?	ESTIMATED IMPLEMENTATION DIFFICULTY	INTERNETWORK CALL CONTROL
METHOD A	YES	YES	NO	YES	EASY	ONE STAGE
METHOD B	YES	YES	NO	NO	LESS EASY	TWO STAGE
METHOD C	POSSIBLE BUT COSTLY	YES	YES	YES	EASY	TWO STAGE

Figure 1.1 Comparison of FDTE Approaches

2. WHO'S AFRAID OF X.25?

X.25 has been cast as the villain in the FDTE controversy because it is unsuited for circuit switched networks and because its flow control is redundant in a single channel DTE, that service already being provided by the frame level. Although the first charge is certainly accurate, the second demands closer investigation. What fraction of channel capacity is wasted as a result of duplicate flow control? How much extra code is involved? Is it difficult to write? For that matter, how big is a typical X.25?

It is to answer these questions that this chapter dissects an existing X.25 implementation. A breakdown of memory requirements by module is given for each of three separate versions with differing levels of generality. It will be seen that X.25 is not intolerable for single channel operation, after all.

2.1 Reduction in Channel Capacity

To calculate the reduction in channel capacity we will consider the worst case of full data packets being transmitted continuously in one direction only; certainly it is in this case that throughput becomes important, and "piggybacking" of acknowledgements provides no help.

Note first that HDLC frames have the following format:

F	A	C	I (if present)	FCS1	FCS2	F
1	1	1	<259	1	1	1

where the numbers indicate the field length in octets; F is the flag used for frame delimitation; A and C are the address and control bytes respectively; I is present only if the frame is carrying a packet, and FCS is the checksum. The trailing flag is not required if frames are adjacent.

The format of an X.25 packet (the contents of the I field above) is as follows:

LCG	LCN	FLO	D
1	1	1	<265

for data packets, and

LCG	LCN	FLO
1	1	1

for flow control packets (RRs), where LCG and LCN are the logical channel group and logical channel number, and FLO is a byte containing flow control sequence numbers.

We shall derive efficiencies by computing the number of forward channel bytes required to send each 256 byte data packet, ignoring bits stuffed for transparency and the possibility of transmission errors. Reverse channel traffic will not directly reduce throughput.

Beginning with X.25, we note that each transmission of a packet will result in transmission of an RR flow control packet in the reverse direction. Depending on the HDLC implementation, this packet may require its own frame level acknowledgement (RR frame). This results in an I frame of 264 bytes plus an RR frame of 5 bytes to carry the 256 bytes of data. This results in an efficiency

$$eff_{p1} = \frac{256}{269} = .9517$$

for packet level flow control. A smarter HDLC would have piggybacked the frame level ack on the next forward channel frame, for an efficiency of

$$eff_{p2} = \frac{256}{264} = .9697$$

Next we compute the efficiency of frame level flow control alone. The FLO byte is unnecessary and no frame level acks need be inserted in the forward channel bit stream. We therefore see the efficiency:

$$eff_{f} = \frac{256}{263} = .9734$$

Comparing the above figures, we see that packet level flow control reduces throughput by at most $(eff_f - eff_{p1})/eff_f = 2.2\%$. More typically the reduction would be $(eff_f - eff_{p2})/eff_f = 0.3\%$. The conclusion is that packet level flow control does not significantly reduce throughput.

2.2 A Benchmark X.25 Package

This section is a guided tour through an existing X.25 implementation. The package was built by the Microprocessor Systems Development Laboratory at Carleton University during the winter of 1977-78 for COSTPRO (Canadian Organization for the Simplification of Trade Procedures) for inclusion in their "Tradex terminal", also built by Carleton. This terminal is a trade forms preparation unit with local storage and electronic mail capability over Datapac; physically it consists of a multiprocessor with a mixed common and private memory architecture in which Intel 8080 CPUs provide the processing power. All code was written in a high level language, PL/M, which was modified by the Lab to allow concurrent programming.

General Capabilities The X.25 package supports an arbitrary number of physical links, with an arbitrary number of logical channels on each, to the limit of the number of USARTs, the processor speed, and the memory space available. In the Tradex terminal it was used to run X.25 to Datapac, to the disk, and to the printer with simultaneous full duplex communication on all links.

In its most general form the package consumes almost 18 K bytes of code and about 150L + 160C K bytes of data where L is the number of links and C is the total number of logical channels on all links. Packet buffers are also required. A single link, single logical channel version, obtained by eliminating based structures, provides a significant reduction in space requirements to less than 12 K bytes of code and 650 bytes of

data. Further reductions can be obtained by recoding portions of the package in assembler, but the extent of such savings is unclear.

As for processor load, it appears that a single CPU is adequate to handle all of X.25 for one link under conditions of continuous transmission at 9600 bps. It should similarly be able to cope with two lines at 4800 bps or other combinations totalling 9600 bps.

Although HDLC procedures form the data link control (level 2), the associated bit-oriented frame structure has not been used. Instead, the character-oriented transparent BSC frame format, implemented in software, provides frame delimitation and transparency. Similarly a CRC-16, also performed in software, replaces the FCS of the bit-oriented frame. Since most of the processor load is associated with these per-character operations, use of an HDLC frame chip (when one is available on a MULTIBUS-compatible board) will greatly increase the limiting data rate.

One advantage of the present character-oriented frame, implemented in software, is that it works equally well for synchronous and for asynchronous (start-stop) transmission. The local links to the disk and to the printer can therefore employ cheap asynchronous data sets.

The Communications Interface The X.25 package presents itself to higher levels of software as five primitive procedures of the packet monitor and two procedures of the buffer pool monitor (Figure 2.1). These procedures supply the user with a set of independently switchable, flow-controlled virtual circuits by means of which other network addresses can be contacted.

Three procedures are concerned with call control. To contact a remote station, the user process calls: PLACE$CALL(CCT$ID$AT, STATION$AT, TIME, STATUS$AT) in which there is first a potential wait for a free logical channel, followed by preparation and sending of a call request packet, and finally a timed wait for notification from the node of the outcome of the call request: a "call connected" or a "clear indication" packet. Both the call status and the channel number (if successful) are returned. Reception of incoming calls would normally be performed by a logger process in the procedure: WAITFORCALL(CCTIDAT, STATION$AT, TIME, STATUS$AT) in which the calling process waits for an incoming call packet on any logical channel, prepares and sends a call accepted packet, and returns the calling station address and the channel number. Finally, the normal termination of any call by both parties is the procedure HANGUP(CCT$ID, TIME, STATUS$AT, VOLUME$AT) in which a "clear request" or "clear confirmation" packet is prepared and sent, followed by a potential timed wait till the channel returns to the ready state, after which the availability of the channel is signalled to any processes waiting in the PLACE$CALL procedure.

The remaining procedures are associated with data transfer once the call is established. To send a message a process will packetize it by copying it into one or more packet buffers from the buffer pool, sending each buffer as it is filled; typically:

```
DO WHILE MESSAGE$NOT$FINISHED;
     PTR = GET$FREE;
     COPY INTO BUFFER INDICATED BY PTR;
     CALL SEND$PKT(CCT$ID, PTR, TIME, STATUS$AT);
     END;
```

In SEND$PKT there is a timed wait if the flow control window is closed,

until it is rotated by an incoming packet; the procedure then prepares the packet header and sends it. To receive a message, a process will receive one or more packet buffers and depacketize by copying them into the message area, returning each buffer to the buffer pool after copying; typically:

```
DO WHILE MESSAGE$NOT$FINISHED;
    PTR = RCVE$PKT(CCT$ID, TIME, STATUS$AT);
    COPY FROM BUFFER INDICATED BY PTR;
    CALL PUT$FREE (PTR);
    END;
```

In procedure RCVE$PKT there is a timed wait until an incoming packet arrives. A window rotation is then transmitted to the node.

The buffer pool procedures are trivial. GET$FREE causes the calling process to wait if the pool is empty, and PUT$FREE awakens any processes waiting for a buffer.

Modules and Processes As shown in Figure 2.2 X.25 is constructed from five modules, including the buffer pool. The buffer pool (BP), the packet module (P), and the frame module (F) are monitors, while the interrupt service routine (ISR) module is protected by interrupt lockout but, used in conjunction with the associated semaphores, acts as a monitor. The CRC module needs no protection since its routines are each called by a single process. The P, F, and ISR modules each encapsulate the data structures and procedures of a different level of function.

Although the packet level of X.25 is contained in a single monitor, the frame level has been split into two sets of logically independent functions, each of which is contained in a separate monitor. The interrupt service routine (ISR) monitor manages framing and transparency as bytes

are moved between a buffer area and the hardware port. CRC generation is performed while the frame is in transit between the ISR monitor and the frame monitor. The frame monitor handles the remaining functions of HDLC: sequence numbering and retransmission, etc. The separation was motivated by the fact that advances in LSI technology and variations among link access procedures of packet networks make the ISR level functions realizable with a number of physical techniques, even though the control structure in our frame monitor remains fixed. With this separation, only the ISR level need be changed to accommodate variations such as an HDLC chip which performs framing, bit stuffing and CRC accumulation itself, or a DMA interface which also handles the move of bytes into main memory, or even a hybrid protocol combining, perhaps, a BSC or DDCMP frame structure with the HDLC control structure.

Adjacent monitors are linked by a pair of transport processes, one for each direction of traffic: PF (packet to frame), FI (frame to ISR), IF (ISR to frame) and FP (frame to packet). The basic actions of the transport processes are simply the cycle of two procedure calls: get a buffer from one monitor, and deliver it to another. The intelligence - the manipulation of data structures and interaction with other processes - is embodied in the code of the monitor procedures. Also of importance are the timeout (T) process which periodically enters the F and P monitors to cause timer expiry actions and the garbage collector (G) process, which removes acknowledged packets from the F monitor send queue and returns them to the free pool. Finally, the control (C) process coordinates actions in the two monitors P and F, particularly at startup and on link failure and recovery.

Multiple Link X.25 As discussed earlier, the X.25 software can handle several physical links. Each of these links has its own set of transport, timeout, garbage collector and control processes interacting in the single copies of the F and P monitors. In the interest of speed, however, the ISR module is not reentrant, but is instead replicated for each physical link. Further, because of the single USART on board the SBC 80/20, the ISR modules run on different processors.

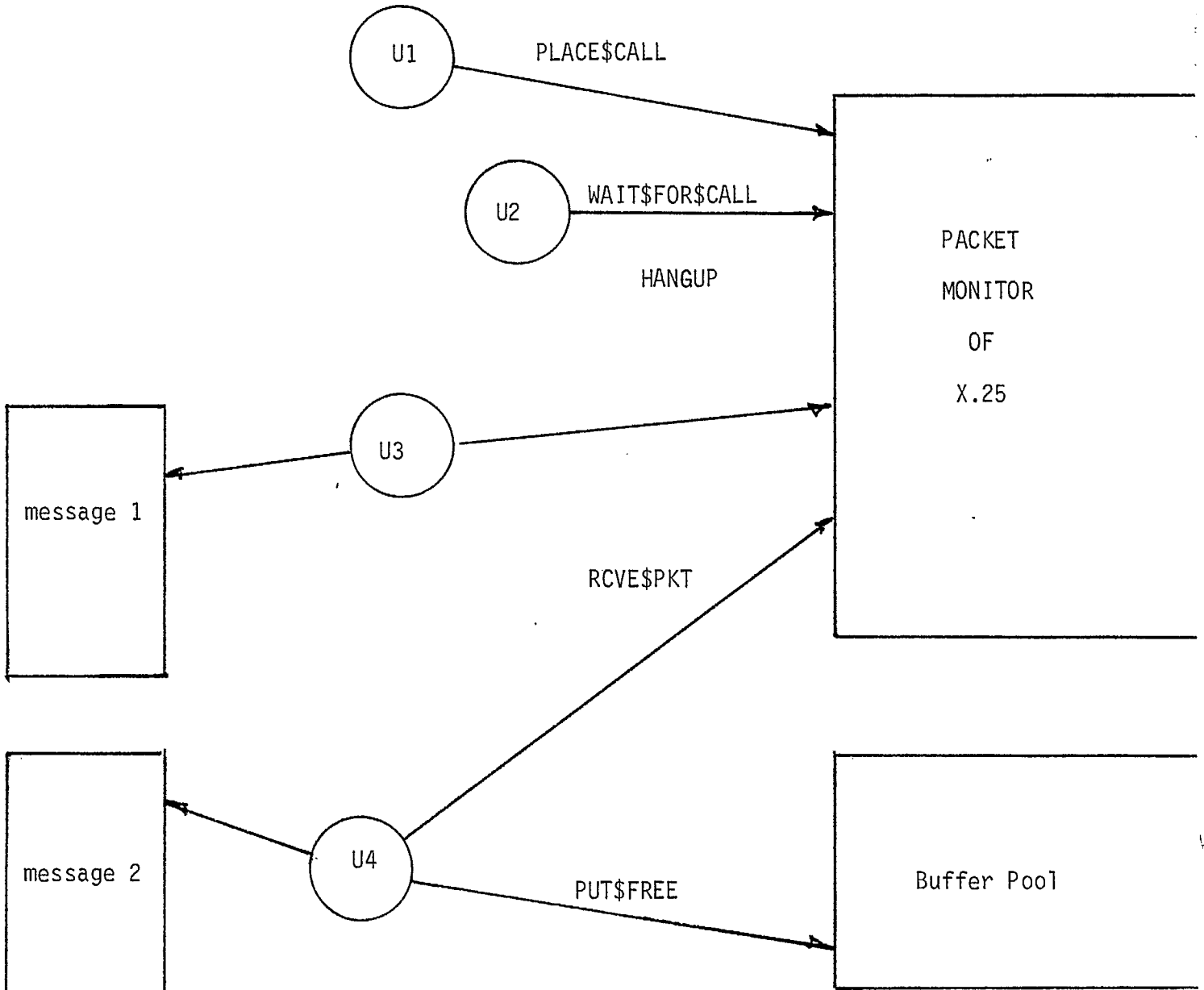
One consequence of distributing the ISR modules and the F modules over different private memories is that a transport process may be required to "hop" onto one CPU to pick up a buffer, and onto another CPU to deliver it.

2.3 Benchmark X.25 Memory Requirements

As mentioned in section 2.2, the package is available in three levels of generality. First is the version which supports multiple physical links, each with multiple logical channels (MLMC). Next, with reduced capability and reduced memory requirements, is the version which supports a single link with multiple channels (SLMC). The principal saving here is in HDLC, which now manages only one link and in the ISRs of which only one need be supplied. Finally we have the single link, single channel version (SLSC) which is of principal interest to the FDTE discussion; simplified access to packet networks for single channel DTEs being a major goal of the study.

In interpreting the figures to be given, the readers should bear in mind a few significant facts. First, all code, including interrupt service routines, was written in PL/M. Estimates of the ratio of size of PL/M-generated code to equivalent code written in assembler by a good programmer have ranged as high as 2:1. Second, the Intel 8080, like most other microprocessors and unlike such medium scale machines as the PDP-11, is clumsy at performing indexed or based adressing. Multiple instructions are often required. Since a multilink HDLC module would be prepared with identical data structures for each link packed into the equivalent of an array indexed by link, we will see a significant reduction in the amount of code space in going from a two link to the single link version. A similar phenomenon holds when the packet module is reduced to a single channel.

For comparison among the versions of X.25 (MLMC, SLMC and SLSC) we consider them each to be applied to the FDTE requirement of single channel operation. The resulting breakdown of memory requirements is shown in Figure 2.3. It can be seen that the version specially compiled for single channel operation (SLSC) weighs in at just over 12K bytes. Of this, about 5.5K bytes can be attributed to the packet module which contains the call control and flow control protocols.



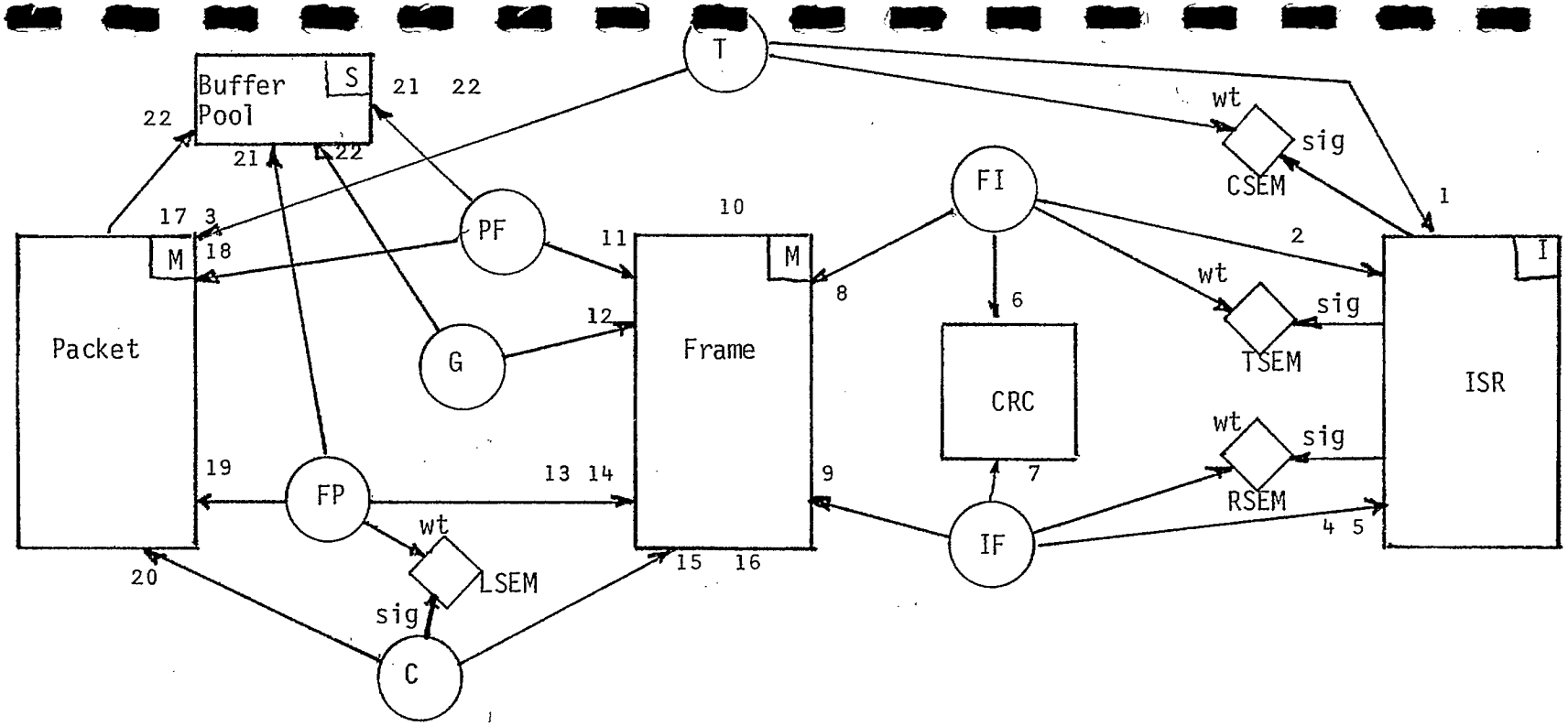
U1 is waiting for a call to be placed

U2 is waiting for an incoming call

U3 is sending message 1

U4 is receiving message 2

Figure 2.1 The X.25 Interface



1 LINK$CHECK
2 TRANSMISSION$SEND
3 RESTART$TOCK (pkt monitor)
4 RECEPTION$SEND
5 RECEPTION$GET
6 ADD$CRC
7 CHECK$CRC
8 FI$GET
9 IF$PUT
10 TICK
11 PF$PUT
12 DEADPKTPICKUP
13 GIVE$EMPTY
14 FP$GET
15 RE$INIT
16 STATUS$WAIT
17 TOCK
18 PF$GET
19 FP$PUT
20 PKT$STATUS$REPORT
21 GET$FREE
22 PUT$FREE

Figure 2.2 X.25 Software Organization

	MLMC	SLMC	SLSC
Interrupt routines and CRC	2236-121	2236-121	2236-121
All process code	660-28	660-28	660-28
Frame module	5336-404	3459-179	3459-179
Packet module	9299-734	9299-734	5276-218
Total	17531-1287	15654-1062	11631-546

Legend: MLMC Multi Link, Multi Channel;

SLMC Single Link, Multi Channel;

SLSC Single Link, Single Channel;

code-data, all figures in bytes

Figure 2.3 Memory Requirements of Three Versions of X.25 in Single Channel Operation

3. METHOD A: CALL CONTROL OVER DATA LINK CONTROL

3.1 Review

As defined in the accompanying report, "A Critical Review of Proposals for the Frame Mode DTE" [1], Method A procedures are those in which call control is located above the DLC. All information regarding call status is exchanged between DTE and DCE in information-bearing frames like all others conveyed by the DLC; in particular, the DLC does not examine any header bytes used to distinguish call control from data or other packets. Clearly X.25 falls into this category, as do X.25-like procedures which rely on flow control at the DLC level (frame level) only.

The principal advantages of Method A are two: first, it is easy to implement in conjunction with an existing DLC software or firmware package; and second, it is independent of the actual DLC used. Therefore point to point and multipoint access can be supported, and BSC, HDLC, SDLC, DDCMP etc. are all potential frame level procedures, subject to mutual agreement between customer and network administration. The principal disadvantage is that it will not work with circuit networks. A second, but minor, disadvantage is that if reliance is placed on frame level flow control, then call control packets are subject to the same flow control as data packets; however a solution is presented in section 3.5.

3.2 Outline of Experimental Work

A principal objective of FDTE definition is to keep the procedures small and to keep them simple. Accordingly, the two designs to be described take ruthless advantage of all features of the environment: the single channel, the DCE (node) support, and the presence of a terminal operator with keyboard and screen. Many cosmetic features have been excluded in the interest of demonstrating only the essential aspects of Method A protocol.

The first Method A protocol to be implemented was, of course, X.25, since it is of critical interest to determine just how simple a DTE can be and still remain compatible with existing X.25 networks, as exemplified by Datapac 3000. The surprising result is that such a device was very simple to build, given a working DLC, and that all call control and packet level flow control procedures account for less than 1 K byte of memory even when coded in PL/M, not assembler.

The other Method A protocol implemented made use of X.25 call control procedures, but removed the redundant packet level flow control. All flow control was therefore based on the "stop-and-go" techniques of the frame level, which in this case was HDLC. The result is incompatible with any existing network, but can be operated back to back for demonstration purposes. Comparison with the skinny X.25 described above will show how much complexity is added by the duplicate flow control of single channel X.25.

3.3 The Application Layer

The application in question is that of conversational terminals. Two DTEs are connected through the network as shown in Figure 3.1, and the operators can prepare messages on their screens and then send them to the other operator's screen. The exchange of messages is not constrained to be half duplex - several messages can be sent in one direction before a reply is returned in the other direction - but use of each screen is regulated so that printing of an incoming message is deferred until the local operator is finished preparing his own message. No display capability greater than "teletype level" is required, that is, no split screens, no independently positioned blocks of text, etc.

Six function keys give the operator control over the conversation. PLACE CALL and ACCEPT CALL have the obvious interpretations; CLEAR CALL must be pressed to terminate every call regardless of whether the clearing is initiated locally or remotely. PREPARE MESSAGE is a bid to establish local screen ownership to inhibit printing of incoming messages while the operator prepares a message for the remote operator. SEND MESSAGE releases screen ownership and causes the message to be transported to the remote DTE. Finally, KILL MESSAGE simply aborts the message in preparation and releases the screen.

Messages are exchanged in "free running dialogue mode"[4], in which local and remote messages are printed on the local screen as units, in

FCFS order, rather than in "alternating dialogue mode". Central to this organization is the concept of screen ownership; at any time the right to print on the screen belongs to the incoming message, the local message in preparation, or is up for grabs (see Figure 3.2). The operator bids for the screen with the PREPARE MESSAGE function key, and releases it with the SEND MESSAGE or KILL MESSAGE keys. Similarly, the first packet of an incoming message causes a bid for screen ownership on its part; if the screen is in use by the operator, then further reception is simply inhibited by flow control. Once having obtained the screen, the remote message is printed and the screen is released.

Note that the free running dialogue design does not preclude alternating dialogue. If the operator is exchanging messages with a computer program, the program itself would enforce strict alternation.

Figures 3.3 and 3.4 show a transcript of both ends of a typical conversation. The reader can see for himself the "natural" quality of the dialogue.

3.4 Skinny X.25

Skinny X.25 is a SNAP (Datapac X.25) - compatible DTE in which all call control functions have been ruthlessly simplified. The normal packet flow control protocol, however, is observed.

For simplicity, no aspect of call control is automatic; instead, the human operator communicates directly with the node. To send a particular call control packet the operator presses the appropriate function key. Call control packets received from the network are decoded and printed as messages on the screen. In this way, operator and node together keep track of call state, so there is no need to maintain the call state FSM (Figure 3.5) by software. In effect the maintenance of the FSM is performed in the node and in the operator's head. If the operator should loose track of the call state and key in something irregular, the node responds with a "clear indication" (See Datapac SNAP Manual Table 3.1) which will be displayed on the screen.

Another aspect of the simplification is evident in Figure 3.6 which shows the relation between function key, packet type and displayed message. Several aspects of the reset, call state and restart FSMs have been combined. The DTE never completes a RESET handshake; instead the operator clears the call. Call CLEAR handshakes are similarly ignored; instead the operator RESTARTs the interface. Finally, the restart FSM (Figure 3.7) allows the RESTART REQUEST packet to serve as a confirm. Hence when the operator presses the function key CLEAR CALL, serves as RESET CONFIRM, CLEAR REQUEST, CLEAR CONFIRM, RESTART REQUEST and RESTART CONFIRM. The three function keys PLACE CALL, ACCEPT CALL and CLEAR CALL are therefore sufficient for all X.25 call control.

Figures 3.8-3.10 show the organization of the skinny X.25 software installed above HDLC. Three processes and one interrupt service routine (ISR) constitute the active entities. To dispose of the least interesting one first, the control (CNTL) process simply acts as link manager and watchdog by causing the link to be reinitialized after any failure.

The keyboard ISR (KISR) picks up keyboard characters and places them in the ring buffer and signals the semaphore KEYB$CHAR$IN, thereby allowing "type ahead". The keyboard process KEYB$PROCESS picks them up and acts on them one at a time by making calls on the various modules. If the character is from a call control function key, the keyboard process obtains a corresponding message from CNTL$MSSGS and echoes it with a call to DISPLAY (bypassing screen ownership). The process then gets an empty buffer from the BUFFER$MANAGER, fills in the packet with a call on the PACKET$HEADERS module and sends it through HDLC to the node. Next consider the case of a keyboard character from one of the message exchange function keys PREPARE MSSG, KILL MSSG and SEND MSSG. The first of these results in a potential wait for screen ownership in SCREEN$ ACCESS, followed by the obtaining and printing of an echo message. The second of the keys causes a release of all buffers, an echo message and a release of ownership. The last one, SEND MSSG, results in an echoed MESSAGE SENT, a release of the screen, and sequence of transmissions of the packet buffers through HDLC. Permission to send is obtained before every packet transmission in the FLO$CNTL module which encapsulates the flow control rules.

Data characters picked up from the keyboard during local screen ownership by KEYB PROCESS are singly echoed on the screen through DISPLAY and stored in a buffer obtained from the pool in BUFFER MANAGER. If the screen does not belong to the operator the characters are lost.

On the reception side, RX PROCESS picks up incoming packets from HDLC, and decodes them in PACKET HEADERS. If it is a control (not data or flow control) packet then an appropriate message is obtained and echoed (CNTL$MSSGS and DISPLAY) with no wait for ownership. Otherwise, if it is the first data packet of a message (as determined by M bit usage) RX PROCESS may have to wait for screen ownership in SCREEN ACCESS. When ownership is obtained, the heading RECEIVED MESSAGE is printed followed by the text in the packet. Subsequent data packets print directly on the screen and the last one of the message releases the screen. During reception the P(S) and P(R) sequence numbers of the packet header, which govern flow control, are interpreted in FLO$CNTL and an explicit RR packet is transmitted if necessary.

All modules except FLO$CNTL and HDLC are executed as critical sections, each protected by its own semaphore. The remaining two modules are arranged as monitors, a more complex organization in which internal process suspension may be involved.

How big is skinny X.25? Figure 3.11 gives the size breakdown for the system. The question must be answered carefully, though, since the functions are distributed and because a number of modules are unrelated to X.25 in particular. Therefore we will not count the keyboard ring

buffer or the DISPLAY module, since they would be present in some form in any intelligent terminal. Nor will we count the BUFFER MANAGER or HDLC, since they or their equivalents would be present in any FDTE. Similarly CNTL and SCREEN ACCESS have no place in a fair comparison among FDTE structures.

We are left with PACKET HEADER and FLO CNTL as the only two modules of the system containing uniquely X.25 code and data structures. Their combined size is 925 bytes or about 7% of the total memory requirements of the terminal, even when written in PL/M.

It is also easily written code. From the point at which a working HDLC was available and the general architecture had been designed, the remaining design, coding and testing of all modules took under 2 man weeks.

3.5 Super Skinny Method A

Super skinny Method A (SSA) is related to X.25 in that similar call control procedures are used. Flow control, however, is provided exclusively by the frame level, thereby eliminating the redundant packet level flow control.

Since SSA is not compatible with any existing or planned network, it will be worth while to specify it in some detail. Packet types (e.g. call control, data etc.) are specified by a code in the packet header. Unlike X.25, data packets do not contain sequence numbers. All packets are exchanged with the network through the DLC, which neither examines nor modifies the packet header and packet body; error control and flow control are therefore exerted uniformly on all packets.

As in X.25, flow control in SSA is a local affair between the DTE and the serving DCE. Therefore three "anchor point pairs" (see Critical Review, Chapter 2) are active during a typical virtual call: (DTE1, DCE1), (DCE1, DCE2), and (DCE2, DTE2). The network ensures that a reset on any one of these three links results in resets of all three, the same convention observed in X.25.

The major objection to SSA is functional: if the call control packets are subject to the same flow control as data packets, then the operator may not be able to clear the call if the remote DTE is, for some reason, not receiving packets. The American FDTE submission to CCITT explicitly notes this drawback. Fortunately, there is a solution. The DLC modules can be arranged so that a link reset causes clearing of the send and receive queues. In order to clear a call, therefore, the operator

need only reset the link before sending the clear request packet. This rather brutal technique does not extend conveniently, however, to other packets which ought to be flow control exempt. Sending an interrupt packet this way, for example, may result in unnecessary data loss.

SSA has the virtue that it is smaller than X.25. In order to determine how much smaller, the experimental version was a modification of skinny X.25. In particular, call control and data packets have the same formats as in X.25 with the exception that the P(R) and P(S) fields of data packets are ignored. Further, flow control (RR) packets are neighter generated nor expected.

Figures 3.12 to 3.13 show the architecture of the experimental SSA FDTE. There is obviously a strong resemblance to skinny X.25. Differences are in fact confined to replacement of the FLO CNTRL module with the EXT QUEUE module, and insertion of another process, the TX PROCESS. In order to send packets the keyboard process now places them in the external queue module, from which they are transported to HDLC by the TX PROCESS. The objective is that the keyboard process not be delayed in a flow controlled HDLC, but remain responsive to keyboard input.

How much smaller is SSA? Figure 3.11 gives the size breakdown for SSA against skinny X.25. Evidently, the reduction is 795 bytes, which is 86% of the X.25 packet level function, and 6% of total code in the system.

3.6 Interworking of SSA and X.25 Terminals

The interworking of several FDTE types through different network arrangements was discussed in Chapter 5 of the "Critical Review" Report. Because of the local interpretation of sequence numbers for flow and error control, there are no major problems in normal call setup and exchange of data packets. In those cases when it is important to push a packet through a flow controlled link (e.g. clear request, interrupt) it is possible to reset the link, with the associated penalty of data loss.

Because it was felt that this arrangement does not hold any lurking surprises, and that its successful implementation would not give much new information, it was not constructed.

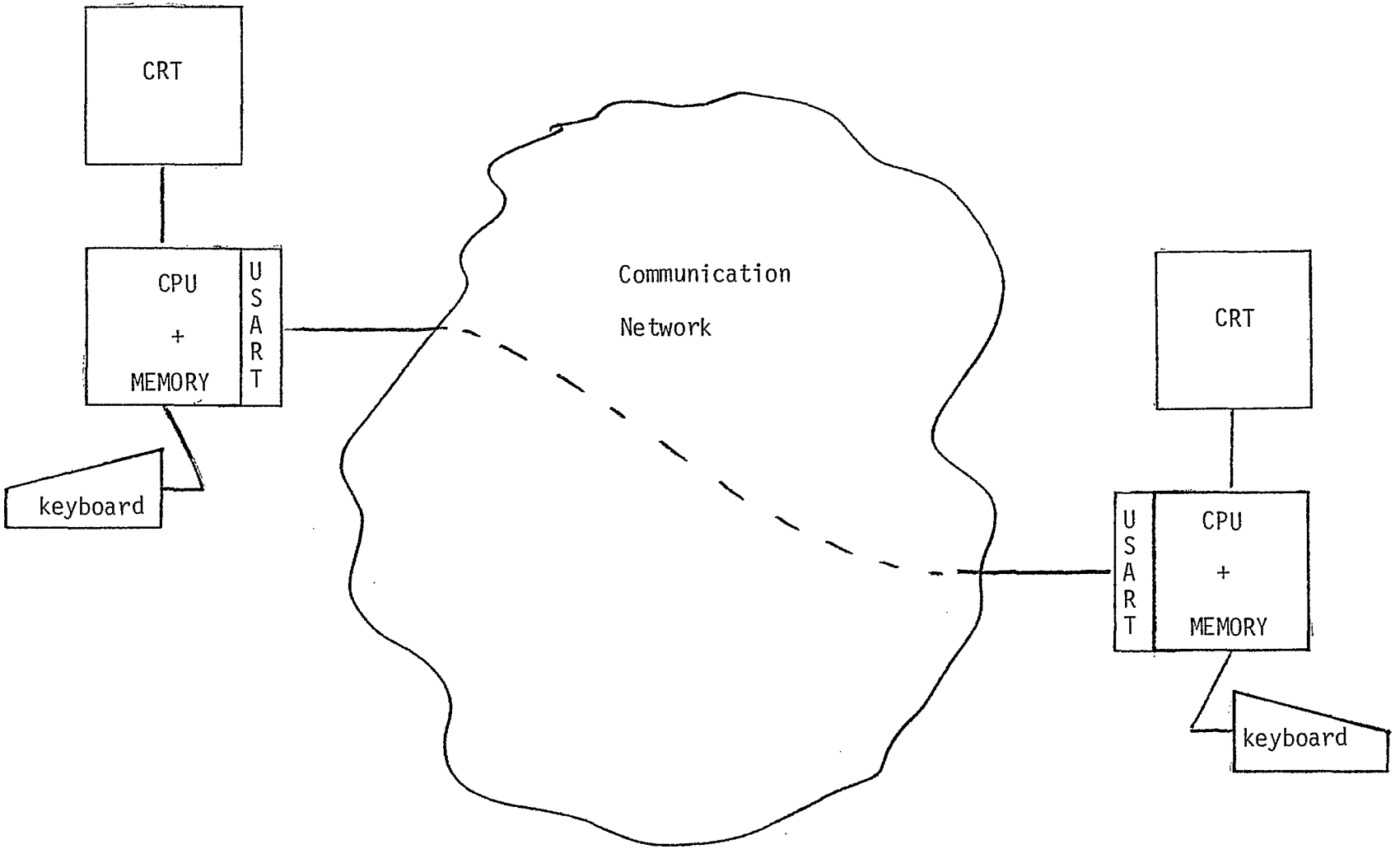


Figure 3.1 Interconnected DTEs

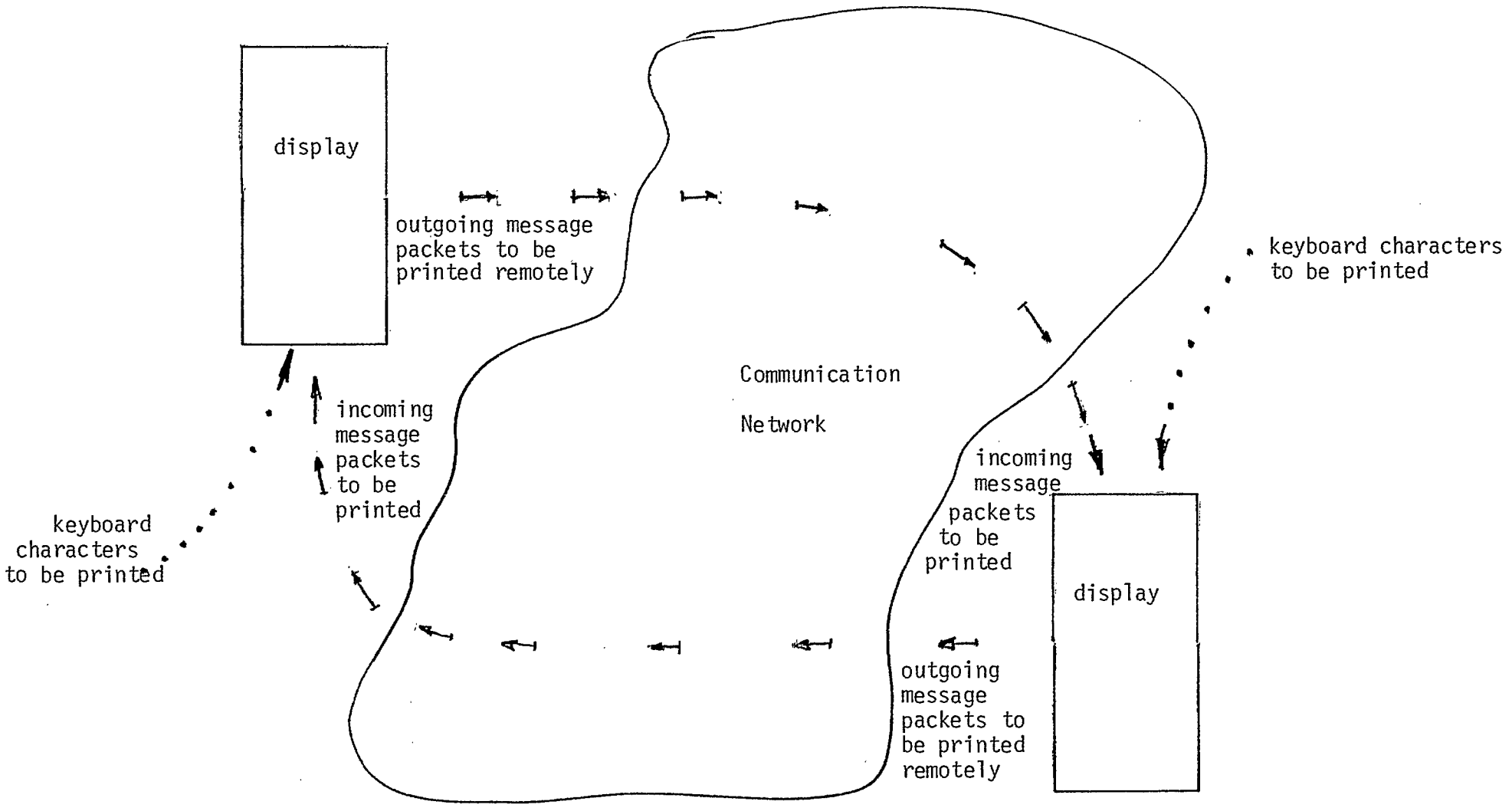


Figure 3.2 Contention for Display Screen

SYSTEM READY

PLACE CALL TO 20400014

CALL CONNECTED

CURRENT MESSAGE IS:

IT'S RAINING IN VANCOUVER

MESSAGE SENT

RECEIVED MESSAGE:

WEATHER IN OTTAWA IS SUNNY

LOCAL CALL CLEAR

REMOTE CALL CLEAR CONFIRMED

SYSTEM READY

CALL FROM 20400019

CALL ACCEPTED

CURRENT MESSAGE IS:

WEATHER IN OTTAWA IS SUNNY

MESSAGE SENT

RECEIVED MESSAGE:

IT'S RAINING IN VANCOUVER

REMOTE CALL CLEAR REQUESTED

LOCAL CALL CLEAR

REMOTE CALL CLEAR CONFIRMED

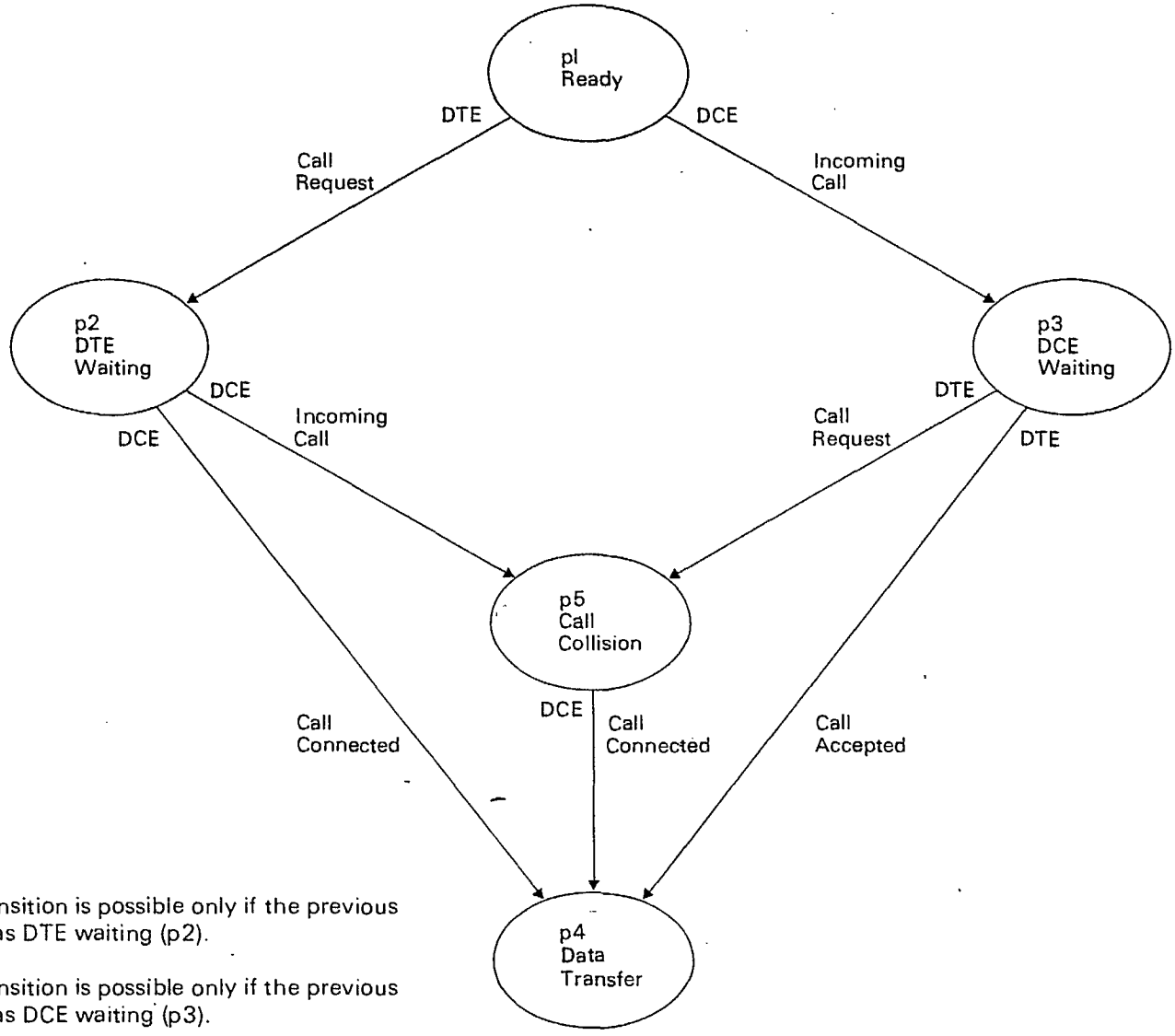
Figure 3.3 Raw Transcripts of a Conversation

CALLING DTE	CALLED DTE	COMMENTS
SYSTEM READY	SYSTEM READY	ON SYSTEM INITIALIZATION
*PLACE CALL TO 20400014	CALL FROM 20400019	CALLING DTE PLACES CALL
CALL CONNECTED	*CALL ACCEPTED	CALLED DTE ACCEPTS CALL
*CURRENT MESSAGE IS:	*CURRENT MESSAGE IS	CALLING / CALLED } DTE PRESSES "PREPARE MSSG" KEY
*MESSAGE SENT RECEIVED MESSAGE	*MESSAGE SENT RECEIVED MESSAGE:	CALLING / CALLED } DTE SENDS MSSG
*LOCAL CALL CLEAR	REMOTE CALL CLEAR REQUESTED[1]	CALLING DTE PRESSES "CLEAR CALL" KEY
REMOTE CALL CLEAR CONFIRMED		RESPONSE FROM CALLING DCE
	*LOCAL CALL CLEAR[2]	CALLED DTE RESPONDS TO NETWORK REQUEST BY PRESSING "CLEAR CALLS" KEY
	REMOTE CALL CLEAR CONFIRMED[3]	RESPONSE FROM CALLED DCE

NOTES: Any detected error will force a 1 2 3 type sequence to clear calls.

- Asterisk * - Indicates use of a function key

Figure 3.4 Annotated Transcript of a Conversation



Note

1. This transition is possible only if the previous state was DTE waiting (p2).
2. This transition is possible only if the previous state was DCE waiting (p3).
3. This transition will take place after a timeout in the network.

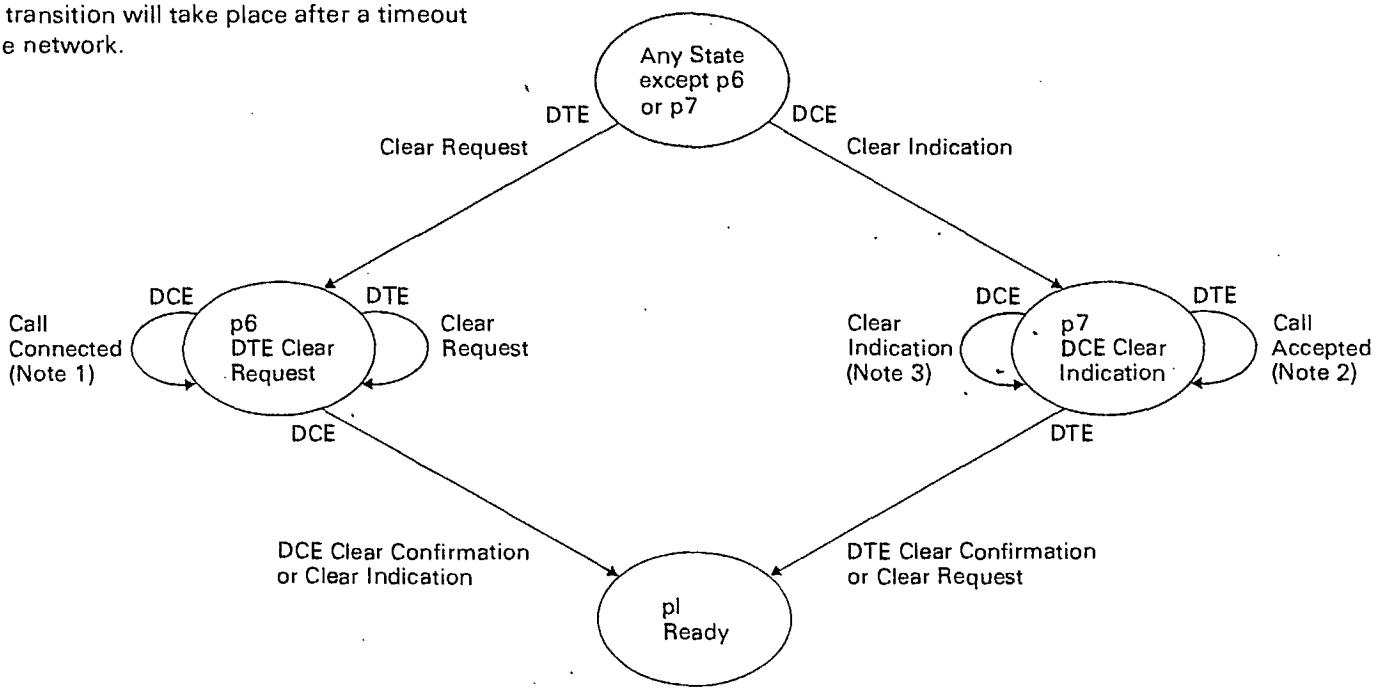


Figure 3.5 X.25 Call State FSM

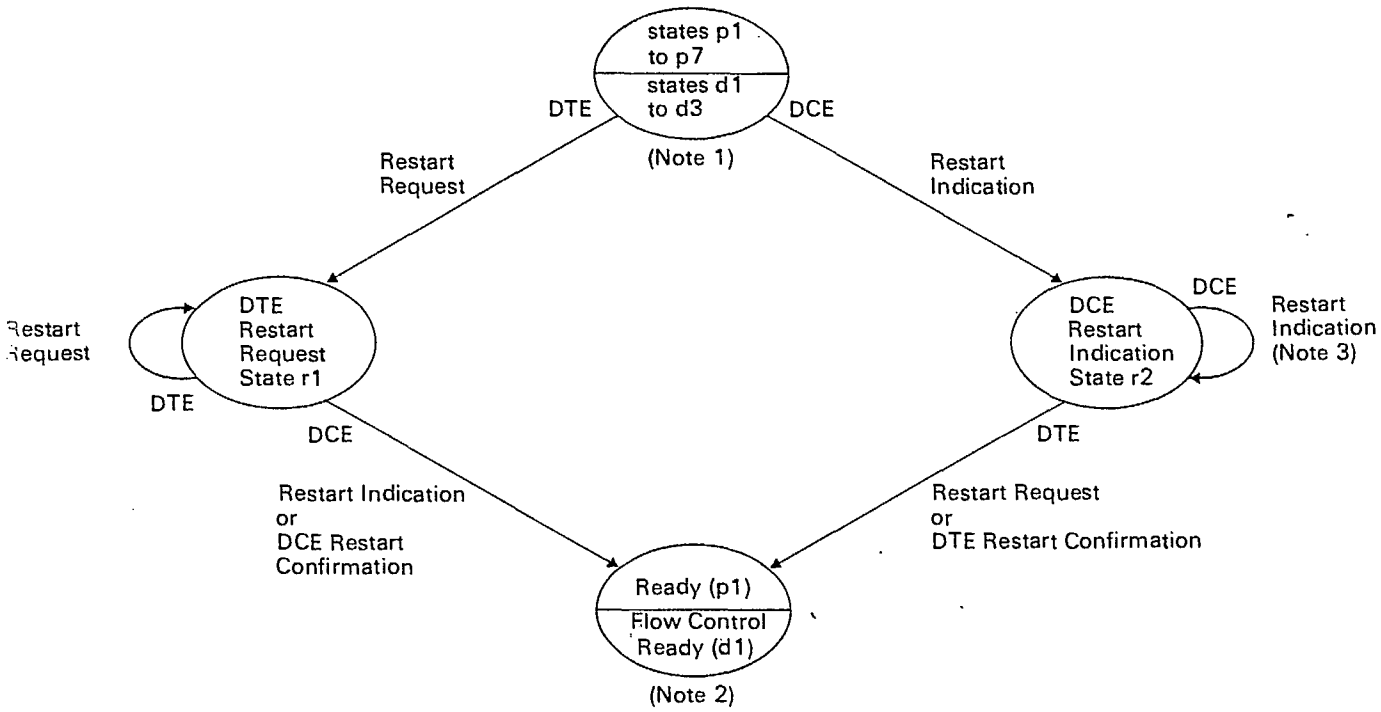
(From Bell Canada, "Standard Network Access Protocol")

Function Key	From DTE	From DCE	Echo Message
		RESTART INDICATION	SYSTEM RESTART, CLEAR CALL LOCALLY
		CLEAR INDICATION	REMOTE CALL CLEAR REQUESTED
CLEAR CALL	RESTART REQUEST		LOCAL CALL CLEAR
		RESTART CONFIRM	REMOTE CALL CLEAR CONFIRMED
		RESET INDICATION	SYSTEM RESET, CLEAR CALL
		INCOMING CALL	CALL FROM XXXXXXXX
ACCEPT CALL	CALL ACCEPTED		CALL ACCEPTED
PLACE CALL	CALL REQUEST		PLACE CALL TO XXXXXXXX
		CALL CONNECTED	CALL CONNECTED

Figure 3.6a Call Control: Function Keys, Packet Types and Operator Messages

Function Key	From DTE	From DCE	Echo Message
PREPARE MSSG			PREPARE MESSAGE
SEND MSSG	Data with M bits chaining		MESSAGE SENT
KILL MSSG			MESSAGE TERMINATED
		Data with M bit chaining	RECEIVED MESSAGE (if first packet) followed by contents of data packet

Figure 3.6b Data Transfer: Function Keys, Packet Types and Operator Messages.



Notes:

1. states p1 to p7 for switched virtual circuits or states d1 to d3 for permanent virtual circuits.

2. state p1 for switched virtual circuits or state d1 for permanent virtual circuits.

3. This transition will take place after a timeout in the network.

Figure 3.7 X.25 Restart FSM

(From Bell Canada, "Standard Network Access Protocol")

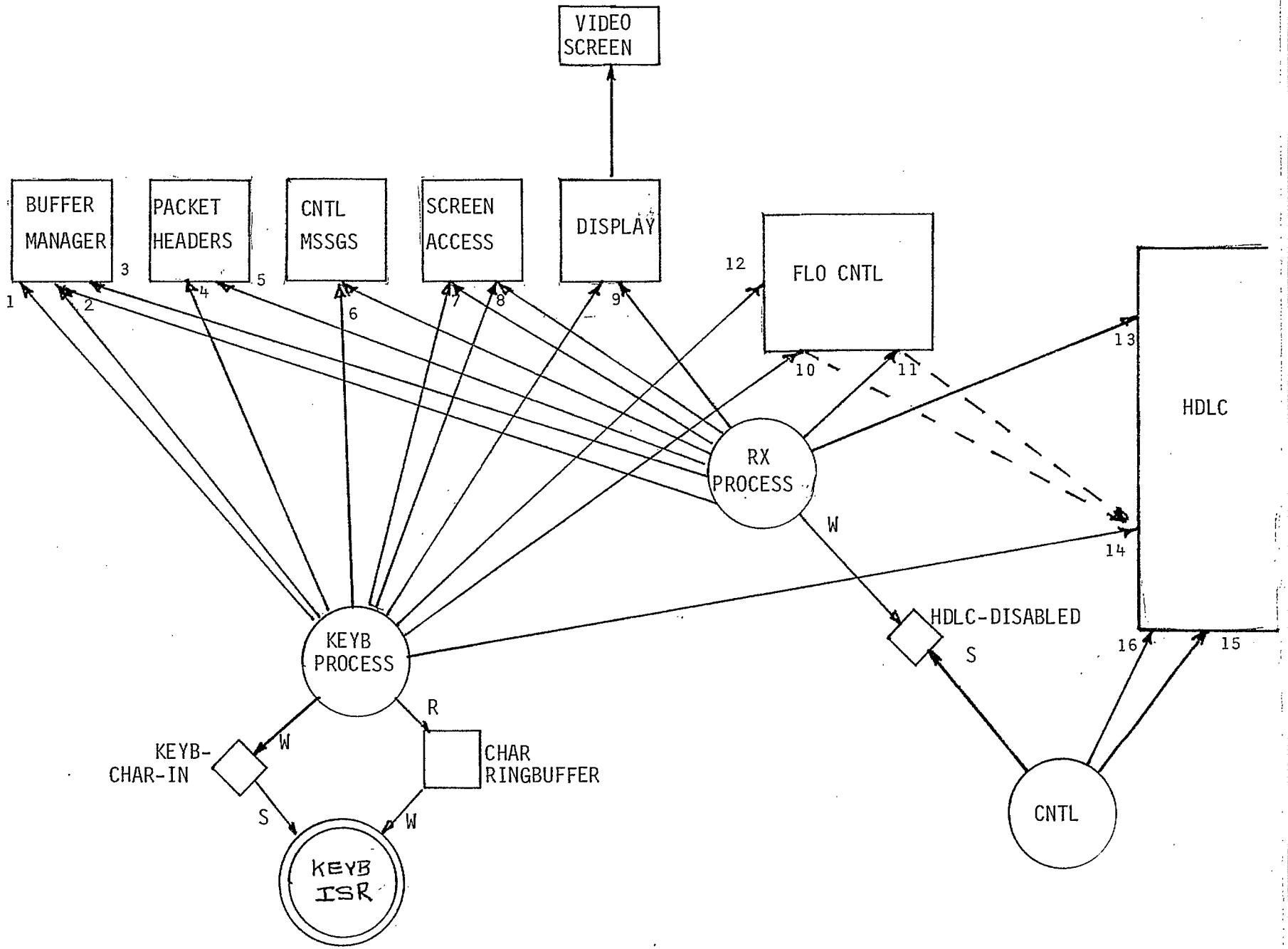


Figure 3.8 a) Skinny X.25 Access Graph

ACCESS GRAPH CALLS:

1) GETPKTBUFFER ← PKT$PTR;
2) PUT$FREE (PKT$PTR);
3) GET$FREE ← PKT$PTR
4) ASSEMBLE$PACKET (PKT$PTR,PKT$TYPE);
5) IDENTIFY$PACKET (PKT$PTR) ← PKT$TYPE;
6) LOCATE$CNTL$MSSG (MSSG$ID,MSSG$AT$PTR,MSSG$LENGTH$PTR);
7) REQUEST$SCREEN$ACCESS;
8) RELEASE$SCREEN$ACCESS;
9) DISPLAY (MSSG$AT,MSSG$LENGTH);
10) SEND$DATA$PKT (PKT$PTR);
11) REPORT$FLOW$CONTROL$RX (PKT$PTR,PR,PS);
12) INIT$FLOW$CONTROL$VARIABLES;
13) FP$GET (PKT$PTR,STATUS$AT) ← PKT$PTR;
14) PF$PUT (PKT$PTR,STATUS$AT);
15) STATUS$WAIT (STATUS$MATCH);
16) RE$INIT (STATUS$AT);

Figure 3.8 b) Legend for Skinny X.25 Access Graph

IDENTIFY$PACKET (PKT$PTR) PKT$TYPE:

A procedure of type byte called to identify the type of packet located in the buffer pointed to by PKT$PTR. On exit, PKT$TYPE is coded as shown below. No delays are possible.

PKT$TYPE	PKT$TYPE
0	incoming call
1	call connected
2	clear ind.
3	clear conf.
4	reset ind.
5	restart ind.
6	restart conf.

SEND$DATA$PKT (PKT$PTR):

An untyped procedure called to obtain transmission of a data packet pointed to by PKT$PTR. If the flow control window is open, the packet will immediately by forwarded to HDLC. Otherwise it will be queued in a buffer queue until the lower window edge is rotated by the reception process. No delay is possible.

REPORT$FLOW$CONTROL$RX (PTR$AT,PR,PS) STATUS:

A procedure of type byte called by the reception process on reception of either an RR packet or a DATA packet. On input, PS or PR specify the send and receive sequence number as conveyed by the last received packet. If these numbers are unacceptable (i.e.: out of sequence or outside the window) then a byte type STATUS with a value of 1 is returned, and no RR packet is generated. It is up to the calling process to take appropriate action (such as clearling the call). If the sequence numbers were within expected ranges then a status of zero is returned. PTR$AT points to an empty packet buffer. Should one be needed to generate an RR packet this buffer will be used and the value at PTR$AT will be reset to zero. Otherwise the value at PTR$AT will remain unchanged, and it is then the caller's responsibility to dispose of the empty buffer. No delay is possible.

INIT$FLOW$CONTROL$VARIABLES:

An untyped procedure called to reset all flow control variables to initial values. This procedure is called by the keyboard process each time it sends a call request or a call accept packet.

Figure 3.9 Skinny X.25 Packet Level Procedures

The display module is responsible for all activities associated with the display. In particular it will handle the protection of the screen as a shared resource, assemble all control messages before they are displayed and provide the necessary mechanisms for displaying a string of contiguous characters. The public procedures of the module are as described below.

REQUEST$SCREEN$OWNERSHIP:

An untyped procedure called by any process wishing exclusive access to the screen for the display of multiple sequential strings. (A multiple packet message, for example). The calling process is delayed until the screen is available for use.

RELEASE$SCREEN$OWNERSHIP:

An untyped procedure used to release access to the screen, thereby enabling its use by other processes.

LOCATE$CNTL$MSSG (MSSG$ID,MSSGAT$PTR,MSSG$LENGTH$PTR):

An untyped procedure called to locate an ASCII string corresponding to the control message specified by the byte type MSSG$ID parameter. On exit, MSSGATPTR will point to an address variable containing the starting address of the string, while MSSG$LENGTH$PTR will point to an address variable containing the length (in bytes) of the string. No delay is possible.

GET$PACKET$BUFFER BUFFER$PTR:

A procedure of type address called to obtain a free packet buffer. On exit a non-zero BUFFER$PTR indicates that the buffer starting at that pointer was granted to the caller. A zero BUFFER$PTR indicates that no buffers were available at the time of the call. No delay is possible when using this call.

GET$FREE BUFFER$PTR:

Same as above except that the caller is delayed should there be no buffers available at the time of the call.

PUT$FREE (BUFFER$PTR):

An untyped procedure called to release the packet buffer pointed to by BUFFER$PTR. No delay is possible.

ASSEMBLE$PACKET (PKT$PTR, PKT$TYPE):

An untyped procedure called to attach a header to the packet buffer pointed to by PKT$PTR. PKT$TYPE specifies which type of header should be attached, and is coded as shown below. No delay is possible.

PKT$TYPE CODE	PKT$TYPE
0	restart request
1	call request
2	call accepted
3	data
4	RR packet

DISPLAY (MSSG$AT,MSSG$LENGTH):

An untyped procedure called to get a string of length MSSG$LENGTH (of type address) located at MSSG$AT displayed. Each character in the string will be output to the video display driver in sequential fashion on a 'poll for device ready' basis. Hence the calling process will be tied down during the time it takes to display the string. No other delay is possible.

Figure 3.10 DISPLAY-RELATED PROCEDURES

	Skinny X.25		Super Skinny Method A	
	Code	Data	Code	Data
KISR + Keyboard Process	1470	63	1470	63
SCREEN ACCESS + DISPLAY + CNTL MSSG	760	0	760	0
BUFF MNGR	100	0	100	0
PKT HDR + FLO CNTL	900	25		
PKT HDR + EXT QUEUE			370	20
Processes CNTL, RX (and TX for SSA)	700	15	435	21
All else (HDLC, startup, kernel, etc.)	7335	2257	7335	2257
Buffers		variable		variable
	11265	2360	10470	2361

Figure 3.11 Memory Requirements Protocols of the Two Method A Protocols

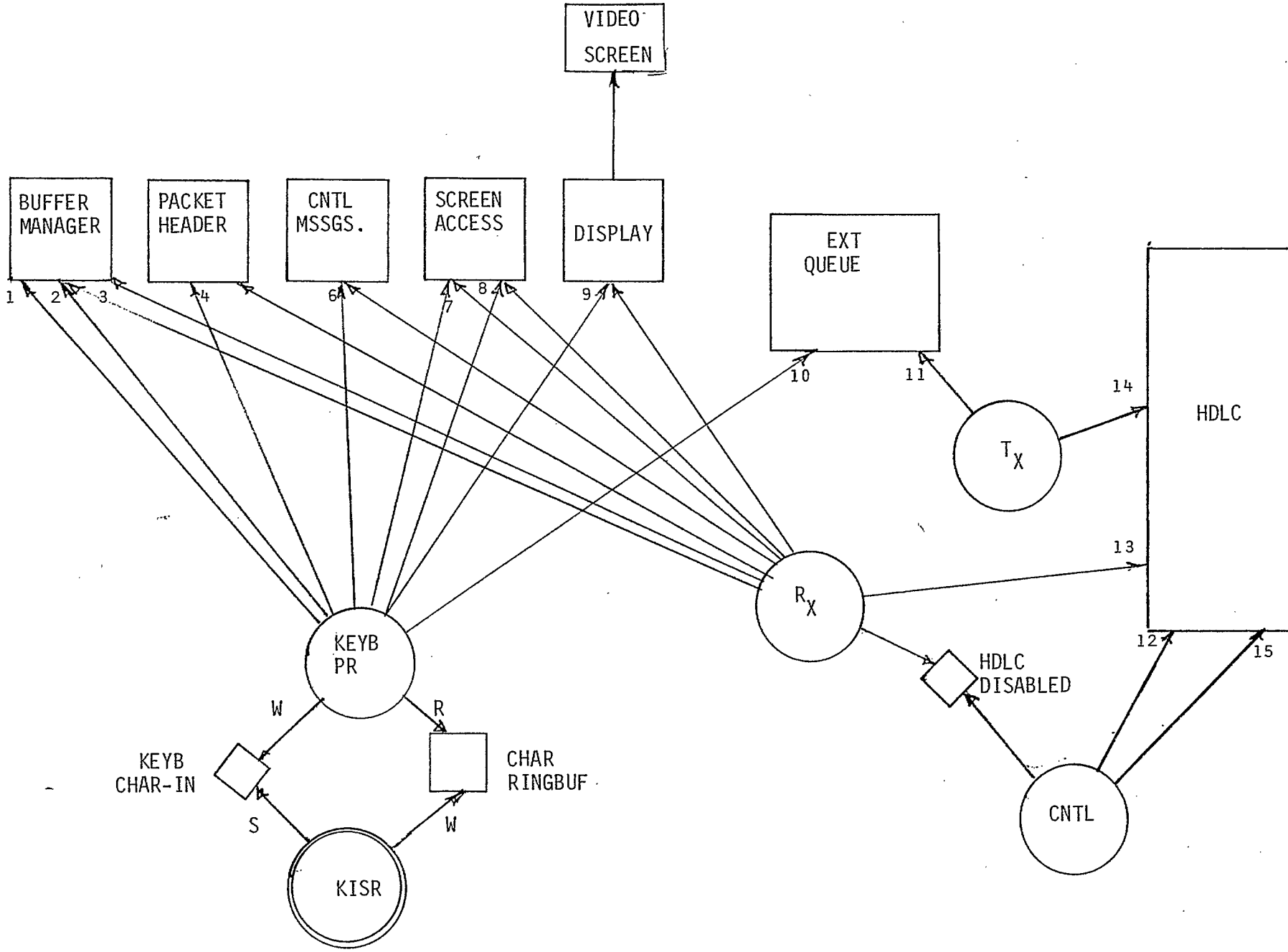


Figure 3.12 a) Super Skinny Method A Access Graph

ACCESS GRAPH CALLS:

```
1)      GET$PKT$BUFFER <-- PKT$PTR;
2)      PUT$FREE (PKT$PTR);
3)      GET$FREE <-- PKT$PTR
4)      ASSEMBLE$PACKET (PKT$PTR,PKT$TYPE);
5)      IDENTIFY$PACKET (PKT$PTR) <-- PKT$TYPE;
6)      LOCATE$CNTL$MSSG (MSSG$ID,MSSG$AT$PTR,MSSG$LENGTH$PTR);
7)      REQUEST$SCREEN$ACCESS;
8)      RELEASE$SCREEN$ACCESS;
9)      DISPLAY (MSSG$AT,MSSG$LENGTH);
10)     SEND$DATA$PKT (PKT$PTR);
11)     GET$PKT <-- PKT$PTR;
12)     RE$INIT (STATUS$AT);
13)     FP$GET (PKT$PTR,STATUS$AT) <-- PKT$PTR;
14)     PF$PUT (PKT$PTR,STATUS$AT)
15)     STATUS$WAIT (STATUS$MATCH);
```

Figure 3.12 b) Legend for Super Skinny Method A Access Graph

SEND$DATA$PKT (PKT$PTR):

An untyped procedure called to obtain transmission of a data packet pointed to by PKT$PTR. The packet will temporarily be queued in EXT QUEUE until the TX process picks it up to forward it to the HDLC monitor. No delay is possible.

GET$PKT ⟵ (PKT$PTR):

An untyped procedure called by the TX process to obtain the next packet to be forwarded to HDLC for transmission. A potential delay exists until such a packet arrives from the Keyboard process.

Figure 3.13 Procedures of Super Skinny Method A Which Differ from Skinny X.25

4. METHOD C: CALL CONTROL BELOW DATA LINK CONTROL

4.1 Review

As defined in the companion report [1], Method C procedures are those in which call control is located below the data link control. Notification of major changes in call status - desire to establish a call and desire to clear a call - is achieved by voltage changes on specific pins of the interface. Selection and call progress signals are exchanged in their own format, and only after the call is established does the DLC attempt to initialize the link (to the node or to the DTE at the other end of the circuit). Clearly X.21 is an example of Method C, as is the BASIC FDTE described in another companion report [2].

Method C interfaces have two compelling advantages. First, they are suitable for both circuit and packet network interfaces, since they do not rely on special frames to clear a call. Second, they are independent of the particular DLC adopted. The customer is not locked into, for example, HDLC in a point-to-point and balanced version; existing BSC and SDLC terminals, as well as DEC computers speaking DDCMP, can access networks with Method C interfaces. Packet networks, of course, must have agreed to support the appropriate DLC.

4.2 Infoswitch X.21

Infoswitch is an integrated network which offers both circuit and packet mode services, termed InfoExchange Service (IES) and InfoCall Service (ICS), respectively. IES and ICS are both accessed through the same X.21-like call control procedure. In fact, the customer can select the service, and hence the tariff structure, he wishes at call setup time, by the addition of a single dialing digit.

In particular, the call control method provided by Infoswitch is based on X.21 bis, the adaptation of X.21 to V-series modems. The actual interface is RS 232 with the interpretation of some pins altered from X.21 bis. Another difference is that Infoswitch provides more flexibility in the envelope format of selection and call progress signals, and provides a richer variety of call progress signals than X.21 bis.

InfoExchange Service is relatively straight forward. After call setup, the calling and called DTEs are connected by a transparent TDM real circuit. Any data, bit or byte oriented, from synchronous terminal or fax reader, (as examples) are passed through the network untouched. Charges are proportional to call duration.

InfoCall Service is a transparent packet switched service [3]. After call setup, data blocks entered in a previously agreed-upon format are carried individually through the network and delivered untouched to the destination DTE. Sequence is preserved although the interblock spacing may be changed. Interframe time fill is not transmitted. Data link control (sequence numbering, error detection and retransmission, etc.) is therefore

conducted on an end to end basis, with the network taking no part. Charges are based strictly on volume, not on connect time, provided the volume is over a minimum level. The unusual feature of InfoCall is that the network cannot exert flow control on a DTE to limit the influx of packets to the network.

4.3 An X.21 Interface Package

This section describes the structure of a software package which provides processes ("tasks") access to the integrated services of Infoswitch. It incorporates:

- call control for IES/ICS
- framing and transparency in the transmission and reception interrupt service routines
- CRC generation and validation
- buffer transmission and reception primitives

It does not include:

- retransmission error control
- flow control; there is a possibility of data loss if the reception procedure is not called often enough

Although this interface does not provide flow control or retransmission error control, it is possible to incorporate these services in an HDLC module above the interface. The composite structure will be described in the next section.

As shown in Figure 4.1, the interface is constructed as two modules and a set of semaphores. The call module is a set of procedures which provide higher levels of software with the standard five communication primitives: place call, wait for call, hangup, send pkt and rcve pkt. This module is not a monitor in the usual sense; it simply coordinates line module accesses and semaphore waits. The exclusion required is provided by the line module, which is a set of interrupt-protected data structures and procedures, forming a critical section. All state variables of the call and the transmission and reception ISRs are maintained in this

module. Because of its intimate association with the line termination devices, the line module can be executed only by processes running on the CPU owning the USART.

The call module generally resembles the X.25 interface, especially in the three call control primitives (PLACE$CALL, WAIT$FOR$CALL, and HANGUP), but there are some unavoidable differences:

- Only one virtual call can exit at a time. For this reason, procedures SEND$PKT, RCVE$PKT and HANGUP need not be reentrant.
- Infocall does not supply the identity of the caller along with the indication of an incoming call, so WAITFORCALL does not return this information. If it is required, terminals can observe a convention that the first packet transmitted by the caller is identification and subconnection information.
- Status messages are not the same as in X.25, since different information about call progress is returned from the network.
- There can be missing packets in the stream obtained from RCVE$PKT, depending on the frequency of calls to this procedure and noise level on the local (node to DTE) lines. Resolution of these problems is for higher levels of software.

The services provided by the InfoSwitch package to higher levels of software are defined by the five procedures described below:

PLACE$CALL(STATION$AT, STATUS$AT)

A reentrant procedure which causes a call to be placed to another DTE. STATION$AT is the address of a user-supplied block which specifies the called network address. STATUS$AT is the address of the user-supplied status block. Normally there is an indefinite

delay until the line becomes available. Following availability, addressing actions take place to bring the virtual circuit to the data transfer state. These actions are monitored by internal timers, expiry of which causes the link to be declared unfit. If the link becomes unfit at any time, for any reason, there is an immediate exit.

WAITFORCALL (STATUS$AT)

A reentrant procedure which causes acceptance of the next incoming call. There is an indefinite delay until network notification of an incoming call. The call is then accepted and the status is returned. If the link becomes unfit at any time there is an immediate exit.

HANGUP (STATUS$AT)

A procedure which is the normal termination to every successfully established call, regardless of which party initiated the call. There is no delay other than that in the return of unused buffers to the pool.

SEND$PKT (PTR,STATUS$AT)

A procedure which places the specified buffer pointer on the transmit queue and initiates transmission, if necessary. There is a delay if buffers are queued too far in advance of the one being transmitted. This is <u>not</u> a form of flow control. If end to end error control is implemented, the CRC bytes must already have been appended to the buffer. There is an immediate exit if the link becomes unfit at any time.

RCVE$PKT (TIME, STATUS$AT)

A procedure of type ADDRESS used to obtain a filled buffer from the reception ISR; a pointer to the buffer is returned as the value of the procedure. There is a delay if no filled buffer is available. The byte TIME specifies the number of clock ISR activations the process is willing to wait. There is an immediate exit if the link becomes unfit at any time.

As might be expected, the software required for this simple call control is small. The line module takes up 2049 bytes. Of this figure 1.2 K bytes are associated with framing and transparency functions. Since these would be largely eliminated (as would the 900 byte CRC module) by use of the current communication controller chips, only about 800 bytes of the line module can be associated with call control. The call module itself is only 635 bytes. Our conclusion, therefore, is that call control for X.21, even in a package designed for processes instead of human beings, costs less than 1.5 K bytes of PL/M code. This is significantly less than that required by the X.25 package described in Chapter 2, and could be reduced further by coding in assembler.

4.4 Software Organization of the First Experimental Method C Terminal

The application used as a test bed is the same as that used in the Method A terminals of Chapter 3: conversational DTEs operating in free-running dialogue mode. Many of the non-communications structures are therefore similar to those of the Method A terminals. The operator interface, for example, consists of keyboard with function keys (listed in Figure 4.2), and a screen running in teletype mode. The right to print a single message on the screen is obtained on a first come, first served basis by the local operator and the process receiving incoming messages.

The major differences are that call setup, both incoming and outgoing, is completed before the HDLC module is turned on, and that call setup and clear down procedures are independent of and do not conflict with HDLC procedures.

Figure 4.3 shows the organization of the software for use of Infocall and InfoExchange services. The undue complexity of the structure results from the late discovery of a duration constraint on one of the interface states associated with call clearing,one which was not recorded in CNCP interface specifications. Compensation for this constraint at the process level was felt to be a faster "fix" than modifying the line module, where changes really should be introduced.

After system startup, transport processes IF (interrupt to frame) and FI (frame to interrupt) are asleep in the HDLC monitor, as is the CNTL process, who waits for HDLC to be turned on. The LOGGER waits for an incoming call and the KEYB process waits for user input. TX waits in

MAILBOX and RX waits on the HDLC$UP semaphore.

If a call comes in, the LOGGER accepts it, turns on HDLC and notifies the operator by a message on the screen. The CNTL process emerges, wakes up RX, then returns to HDLC to wait for the link to be deadwaited. The call is now in progress, and actions are virtually identical to those of the Super Skinny Method A terminal (Chapter 3).

If an outgoing call is to be placed in response to the operator pressing the appropriate function key, KEYB accesses the PLACE$CALL in the CALL MODULE and notifies the operator of the result.

If the network clears the call, IF or FI turn HDLC off, causing CNTL to emerge, call HANGUP and notify the operator. On the other hand, if the call is cleared in response to a function key pressed by the operator, then KEYB turns HDLC off, causing CNTL to call HANGUP.

This interface was used for both the circuit and transparent packet services of Infoswitch. The only detectable difference was a slightly increased delay in the case of packet mode transmission. If nothing else, it demonstrates that X.21 can be used as a packet net access method.

4.5 The Second Experimental Method C Terminal

The previous section defined a software structure in which HDLC sat above a bare X.21 interface. Although these terminals ran satisfactorily in that they could call each other and exchange messages in free-running dialogue mode, they had inherent limitations. As explained in Chapter 5 of another report [1], bare X.21 does not allow the calling DTE to identify itself; it also leads to non-uniformities in call control procedures when placing internetwork calls. One solution, also described in that report, is an end-to-end convention immediately following X.21 call setup whereby the calling DTE transmits the equivalent of an X.25 call request/incoming call packet, and the called DTE responds with a call accepted/connected packet (or clears the X.21 call). After this exchange the two DTEs initialize the end-to-end DLC by a set mode command.

The second experimental Method C terminals differed from the first only in the inclusion of a mechanism for accomplishing the exchange described above. Figure 4.4 shows the transcript of a typical call, in which it can be seen that the operator is automatically provided with the identity of the calling terminal, a service not provided by bare X.21. Specifically the KEYB process, after successfully emerging from the PLACE$CALL procedure of the CALL module, prepares an X.25 call request packet containing both DTE addresses, transmits it over the X.21 connection in procedure SEND$PKT, and waits for a call connected reply in procedure RCVE$PKT. (Both these procedures are in the CALL module). Having completed the exchange, the KEYB process calls the HDLC module procedure TURN$ON. The CNTL process will notify

the operator once the DLC is set up that the call is ready for data transfer.

As for reception of incoming calls, if the LOGGER process emerges from procedure WAITFORCALL of the CALL module, it then calls RCVE$PKT to obtain the incoming call packet transmitted by the calling DTE and displays the address of that DTE to the local operator. LOGGER then sends a call accepted packet by SEND$PKT and turns on HDLC. As before, the CNTL process notifies the operator when the DLC is set up.

Evidently, the memory required to support this call control protocol is that of the X.21 interface described in the previous section, plus that required for preparation and interpretation of the call request and call connected packets, plus that required for the extra message to the operator. The increment is under 300 bytes, for a total of about 1.8 K bytes associated with call control. It should be noted that the inclusion of the exchange of X.25 frames after X.21 call setup is not difficult, once the machinery of section 4.4 is in place. Our experience was one man day.

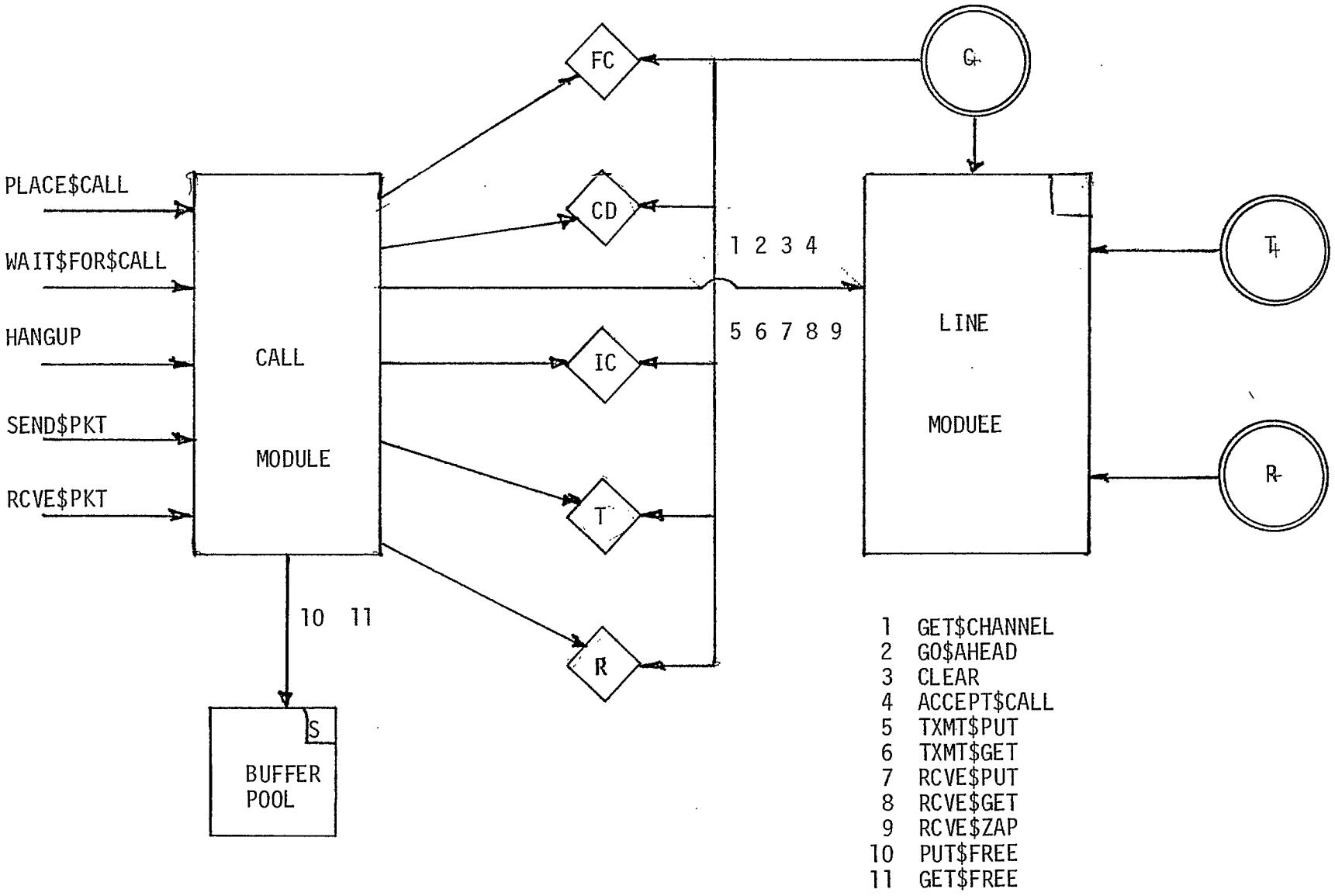


Figure 4.1 General Organization of Micro ICS Interface

FUNCTION KEY	INTERFACE ACTION	OPERATOR MESSAGE
PLACE CALL		PLACE CALL TO: XXXXXXXXXX
	HDLC set up after outgoing or incoming X.21 call setup	CALL CONNECTED
	identification of calling DTE after incoming X.21 call setup (second Method C DTE only).	CALL FROM XXXXXXXXXX
PREPARE MSSG		CURRENT MESSAGE IS:
SEND MSSG		MESSAGE SENT
	first packet of incoming multipkt message (chained above HDLC)	MESSAGE RECEIVED:
CLEAR CALL	network clearing of X.21 call or system initialization	CALL CLEARED SYSTEM READY

Figure 4.2 Method C Function Keys, Interface Actions and Operator Messages

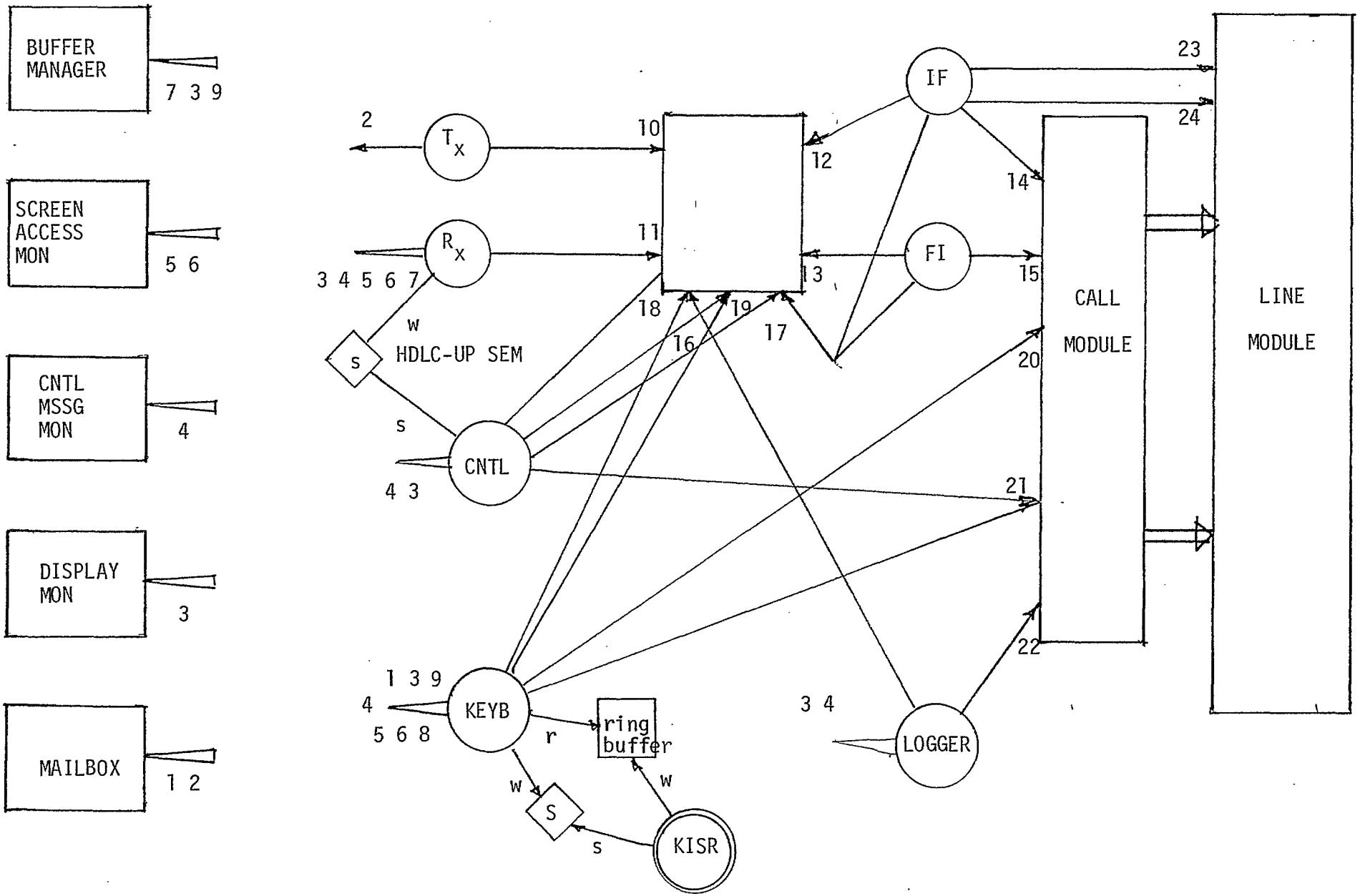


Figure 4.3 a) Access Diagram of Infoswitch Interface

```
1     SEND$DATA$PKT (PKTPTR);
2     GET$PKT <-- PKTPTR;
3     DISPLAY (STRINGAT, STRINGLENGTH);
4     LOCATE$CNTL$MSSG (MSSG$ID, MSSGATPTR, MSSGLENGTHPTR);
5     REQUEST$SCREEN$ACCESS;
6     RELEASE$SCREEN$ACCESS;
7     GET$FREE <-- PKTPTR;
8     PUT$FREE (PKTPTR);
9     GET$PKT$BUFFER <-- PKTPTR;
10    PF$PUT (PKT$PTR, STATUS$AT);
11    FP$GET (PKT$PTR, STATUS$AT) <-- PKTPTR;
12    IF$PUT (PKTPTR) <-- PKTPTR;
13    FI$GET <-- PKTPTR;
14    IF$GET (PKTPTR) <-- PKTPTR;
15    SEND$PKT (PTR, .STATUS);
16    TURN$ON;
17    TURN$OFF;
18    GIVE$EMPTY (PKTPTR);
19    STATUS$WAIT (STATUS$MATCH, STATUS$AT);
20    PLACE$CALL (STATION$AT, STATUS$AT);
21    HANGUP (STATUSAT);
22    WAIT$FOR$CALL;
23    RCVE$PUT (PTR);
24    GET$RCVE$BUF <-- PKT$PTR;
```

Figure 4.3 b) Legend for Infoswitch Access Graph

CALLING DTE	CALLED DTE	COMMENTS
CALLS CLEARED *SYSTEM READY*	*CALLS CLEARED* *SYSTEM READY*	UPON SYSTEM INITIALIZATION
**PLACE CALL TO:* 5015011 CR	*CALL FROM* 5015012	CALLING DTE PLACES CALL X.21 CALL IS CONNECTED (LINK NOT AVAILABLE UNTIL HDLC IS UP)
CALL CONNECTED	*CALL CONNECTED*	HDLC IS UP BOTH DTE'S LINK IS AVAILABLE FOR USE THROUGH HDLC
**CURRENT MESSAGE IS:* HELLO HUGH!	**CURRENT MESSAGE IS:* HELLO AGATHA!	BOTH OPERATORS PRESS THE "PREPARE MSSG" KEY AND BUILD A MESSAGE
MESSAGE SENT *MESSAGE RECEIVED:* HELLO AGATHA!	*MESSAGE SENT* *MESSAGE RECEIVED:* HELLO HUGH!	BOTH OPERATORS PRESS THE "SEND MSSG" KEY
**CALLS CLEARED* *SYSTEM READY*	**CALLS CLEARED* *SYSTEM READY*	EITHER ONE, OR BOTH OPERATORS (SIMULTANEOUSLY) PRESS THE CLEAR CALL KEY
		OR, ERROR WAS DETECTED BY DTE SOFTWARE OR BY THE NETWORK

* Asterisk denotes operator use of function keys.

Figure 4.4 Conversation on Second Method C FDTE

5. EXPERIMENTAL INTERNETWORKING OF CIRCUIT AND PACKET FDTES

5.1 Objectives

We have defined and constructed both packet mode and circuit mode FDTEs. One of the major objectives of this study was to ensure that any such FDTEs be able to interwork with each other and with X.25 DTEs, and that tandem circuit/packet connections be possible. It was felt that the best way to gain confidence in these designs was to actually construct an experimental internetwork switch which would allow calls between packet and circuit networks.

Accordingly, an experimental gateway node was constructed (Figure 5.1) which connected the circuit mode service (IES) of Infoswitch wtih a Datapac-equivalent X.25 network. The local net interface was at the host level, in Sunshine's terminology [5]: Infoswitch presented the gateway with bare X.21; the Datapac-equivalent network presented itself as X.25. The job of the gateway, of course, was to bridge the interface with call control and data transfer processes.

The objectives were relatively clear. We wished to demonstrate the feasibility of internetworking the circuit mode FDTE with a packet mode X.25 DTE. Internetworking was to mean both call control and data transfer functions. In doing so, we hoped to turn up incompatibilities or awkwardnesses not suggested by the earlier FDTE construction (Chapter 4) or analysis [1, Chapter 5]. Any changes to the FDTE resulting in increased complexity were to be determined.

5.2 Configuration and Assumptions

The DTE on the packet network shown in Figure 5.1 was a Skinny X.25 terminal (Chapter 3); the reader will recall that it is Datapac-compatible. The Super Skinny Method A terminal (Chapter 3) was considered for the packet network DTE, but was rejected for two reasons: first, it would have made no difference technically, since the flow control window is interpreted locally; and second, it was felt that interworking with a true X.25 terminal would be more convincing. The circuit network DTE was an unmodified Method C DTE of the second type constructed (Chapter 4), one which exchanges X.25-like call request and call connected packets with the remote DTE after X.21 call establishment and before DLC initialization.

The gateway was a multiprocessor in which each network had its own CPU for servicing interrupts. Other processes associated with packet transport and call control ran on whichever CPU seemed appropriate for a given function. These processes viewed the circuit and packet networks through interfaces based on the second Method C and Skinny X.25 respectively. More details of the gateway will not be given, since it was entirely disposable, ad hoc software.

A certain suspension of disbelief is required in viewing the configuration as a viable internetwork service. For one thing, it assumes that both networks are actively cooperating by employing a global (or at least mutual) numbering scheme, so that extra-network calls are detected and routed to the appropriate gateway. Another point of unreality is that the Datapac-equivalent network is accessed by the gateway through X.25. In fact, the X.75 internetwork protocol would

be the appropriate interface if correct addresses are to be retained in the call request/incoming call packet. The present arrangement, for example, will cause the packet net DTE to be informed that an incoming call is from the gateway, rather than from the DTE on the X.21 network.

5.3 Results

The experimental internetwork arrangement essentially confirmed the earlier analysis and design. No change was required to either the Skinny X.25 DTE or the X.21 FDTE. The gateway, however, was required to perform some tricks, as outlined below.

During data transfer, flow control was exerted on what was, in effect, an end-to-end basis, though in fact there were four anchor point pairs: (X.25 DTE, DCE), (DCE, DCE), (DCE, gateway), (gateway FDTE). The gateway process carrying packets from the circuit interface to the packet interface would, if delayed due to X.25 flow control, cause a frame level RNR (receive not ready) to be sent over the circuit link simply because of buffer starvation. Similarly, if the process carrying packets from the packet network to the circuit network were delayed because of K HDLC I frames outstanding, then the X.25 window would fill and no window rotation would be generated to be carried back to the sending DTE through the packet network. One effect, however, of our amateur gateway was that the flow control windows added over the two networks: if the Datapac window W=2 and the circuit net HDLC maximum frame advance K=3 then the total number of frames outstanding between source and destination would be 5. This is a minor problem and can be corrected by modifying the X.25 interface so that procedure RCVE$PKT does not automatically generate a window rotation, and instead making this action the result of a new procedure BUMP$CREDIT. In any case, design of a gateway was not the principal interest of the experiment, so the modification was not performed.

As for call control, there were obvious problems attributable to the fact that the networks were unaware of, and hence not cooperating with the experiment. As a result anomalous addresses appeared in the incoming call packets. Specifically, consider a call originating at the packet DTE for the circuit DTE. The actual sequence was as follows: the packet DTE emits a call request packet with the called address as the gateway; the gateway receives it as an incoming call packet and assumes that it is intended for the specific circuit mode DTE used in the experiment; the gateway modifies the "called DTE address" field and places the X.21 call to the destination, and follows the call setup with the call request packet; the called DTE after X.21 setup receives the correct incoming call packet. The other direction of call placement was similarly awkward, with the following sequence: the calling (circuit net) DTE places an X.21 call to the gateway, then sends the call request packet with the packet DTE address in the "called" field (note different "called" addresses in the two stages of call placement); the gateway does not have to assume a destination for calls in this direction, but must insert its own address in the "calling" field to conform with X.25 when placing the packet net call; the destination DTE therefore receives an incoming call packet indicating only the gateway as the source.

Now for what we require of the networks in order to make the arrangement work properly. First, a mutual DTE numbering scheme is necessary as is the cooperation of the networks in routing a call to the proper gateway when a foreign called DTE address is recognized. The effects would be that a calling packet net DTE could specify the correct destination address in its call request packet, rather than the gateway address, and that the circuit mode DTE could use the same destination address in both

stages of call control. The second requirement is that the packet net present the gateway with an X.75, rather than X.25, interface. The effect would be that the call request/incoming call packets crossing this interface would not be required to have one of the addresses be that of the gateway; the called packet DTE would be given the correct identity of the calling DTE, on one hand, and the gateway would be given the correct address of the called DTE, on the other. With this level of network cooperation, there is no obstacle to convenient internetworking between dissimilar services.

Finally, we should note again that, while most of the technical issues are easily resolved, there remains a minor problem with tariffs. When the circuit net DTE calls a packet DTE through the structure described above, charges on the circuit portion of the path begin to accumulate from the time of establishment of the call to the gateway. The time to set up the packet portion of the call is therefore chargeable, even in the case of a packet DTE with slow reaction time, or a refused call. This problem would be eliminated by adoption of the BASIC FDTE interface [2].

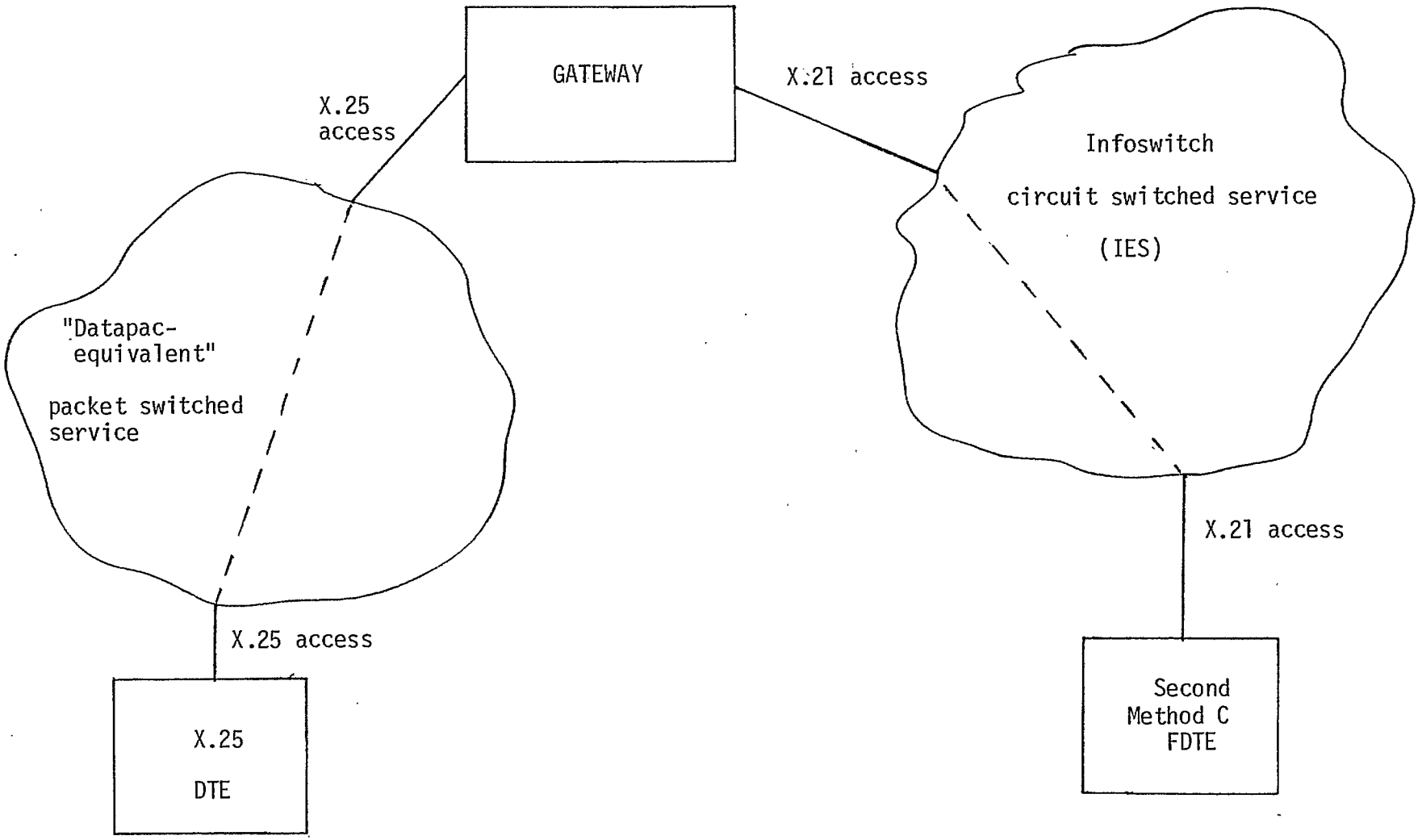


Figure 5.1 Experimental Internetwork Configuration

6. CONCLUSIONS

Our conclusions must begin with a small disclaimer; although we can be precise about the function and size of the various FDTE interfaces, only informal estimates of the comparative implementation difficulty are available. This is a result of problems with Infoswitch. It was the first time they had offered this access method to a customer, and it did not work according to their specifications. Eventually they remedied the situation, after several reversals on sync and framing characters and CRC algorithm. In the interval, several man weeks were lost trying to track Infoswitch, and the estimates of difficulty became hopelessly confused.

In proceeding to firmer conclusions, we consider the first objective of simplified access to packet networks. Earlier analysis [1] demonstrated that Method A protocols were most desirable in this context, and among Method A protocols X.25 itself deserves examination. A full multichannel version suitable for automatic operation on a multiprogrammed multiprocessor required almost 10 K bytes in addition to HDLC. This package was far too sophisticated for the application area of interest to the FDTE controversy: the single channel, synchronous interactive terminal. Accordingly a single channel X.25 which takes ruthless advantage of the environment was built. After the design the implementation proved to be exceedingly simple, taking less than 1 K byte of space in addition to HDLC, and the result was compatible with Datapac.

Another Method A protocol, in which the redundant packet level flow control had been removed, was also built. It reduced the memory requirements

from those of the special X.25 above by almost 800 bytes but this figure represents only 6% of the total code in the system. However it was not Datapac compatible: every out of band (flow control exempt) packet, such as call control or interrupt, had to be preceded by a frame level reset, with associated data loss. Whether the loss is a problem depends on the application. Our first conclusion therefore, is that if simple access to packet networks is the only criterion, there is little reason not to adopt X.25 itself. Stripped-down versions just do not save enough memory or simplify the implementation enough to justify the loss of function.

As for the second objective of uniform procedures for access to circuit and packet networks, it has been shown already [1] that some Method C interface is required. We found that a bare X.21 interface capable of coordinating call control and the sending and receiving of frames took about 1.5 K bytes. This is larger than the Method A interfaces above, in part because the call state FSM was maintained in the interface. It was also found to be somewhat more difficult code to write because of its proximity to the hardware and interrupt service routines. Functionally, X.21 allowed uniform procedures on the Infoswitch circuit and transparent packet services, and there is no reason why more conventional packet networks could not also offer an X.21 interface. It was deficient in one respect; that it gave no indication of the identity of the calling DTE. As discussed elsewhere [1], this feature gives rise to various ideosyncrasies in placing intra and internetwork calls. We implemented one solution for circuit and transparent packet networks in the form of a layer of end-to-end identification exchange at a cost of 350 bytes of easy code. It is questionable whether this can be considered a candidate

"universal interface", though; conventional packet networks supporting this exchange would have to shoulder much of the complexity themselves in taking very special actions at call setup time which depend on the nature of both the calling and the called DTE. Node software would require significant modification and would no longer be "clean". The second conclusion, then, is that X.21 is barely acceptable with respect to our second objective.

In this respect, BASIC FDTE [2] is superior, in that identification of the caller is automatic, as it is in X.25, without a second layer of identification exchange. The third conclusion is that BASIC FDTE is a potential universal interface, though it is not at present supported by any network.

Internetworking of a circuit mode FDTE and a packet mode X.25 DTE showed that the data transfer phase presented no technical difficulties. It also demonstrated clearly the fourth conclusion, that mutual (or global) numbering and network cooperation are essential to the success of convenient internetworking, no matter how elegant the FDTE interface.

REFERENCES

[1] J. K. Cavers, "A Critical Review of Proposals for the Frame Mode Interface to Digital Networks", Report on Contract OSU78-00182, Department of Communications, Ottawa, November, 1978.

[2] J. K. Cavers, "BASIC FDTE: A New Frame Mode Interface to Digital Networks", Report on Contract OSU78-00182, Department of Communications, Ottawa, November, 1978.

[3] G. V. Bochmann, "The Frame Mode DTE Interface", Report on Contract 02SU.36100-7-0304, Department of Communications, Ottawa, October 1977.

[4] IFIP/WG 6.1, "Proposal for a Standard Virtual Terminal Protocol", ISO/TC97/SC 16 N, INWG Protocol 91, February 1978.

[5] C.A. Sunshine, "Interconnection of Computer Networks", Computer Networks, Vol. 1 pp. 175-195, 1977.