X.25 PACKET LEVEL COMMUNICATION PROTOCOLS: AN INTERPRETATION AND PROPOSED IMPLEMENTATION FROM DTE VIEWPOINT

Bijendra N. Jain*

Department of Computer Science
University of Texas at Austin
Austin, Texas 78712

TR 187 November 1981

---

*The author is on leave from Department of Electrical Engineering, Indian Institute of Technology, Delhi 110016, India. His work in India was supported in part by Department of Science and Technology (Government of India) Grant No. 13(12)/79-SERC. This report is also available as a technical report from Department of Electrical Engineering, Indian Institute of Technology, Delhi, India.

ABSTRACT

This report is an attempt to interpret CCITT X.25 Level 3 (Packet Level) procedures to be used across the interface between a Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) operating in a packet mode on public data networks. The procedures enable the establishment and operation of a number of virtual calls and/or permanent virtual circuits between the DTE and DCE through the use of logical channels.

The interpretation of the interface description is given in the form of a set of procedures written in PASCAL that also suggest its implementation on a DTE.

CHAPTER 1

INTRODUCTION

1.1 X.25 Communication Protocols

The past decade has seen the development of a growing number of public data networks that support computer communication, either in the form of experimental or commercial networks based on packet switching (U.S.A. : Telenet, Arpanet, Canada : Datapac, France : RCP, U.K. : EPSS, Japan : DDX, etc.). In their early stages each network developed its own procedures for data terminals to access network facilities. In an attempt to facilitate connection of the variety of data terminals (host computers and smart terminals) to message siwtched data networks, CCITT has over the past few years developed a number of standards known as 'recommendations'. Recommendation X.25, pertaining to an interface between a data terminal and a network node, is one of the most widely discussed and accepted in recent years.

The CCITT recommendations X.25 /1/ attempt to standardize the interchange of data between a user terminal or a host computer (technically known as Data Terminal Equipment (DTE)) and the device, known as Data Circuit Terminating Equipment (DCE), through which the DTE accesses the network. It thus specifies the functional characteristics presented by the network to a DTE connected to it, and visa-versa. These are logically divided into three levels (see Figure 1.1):

Level 1: the physical level,

Level 2: the frame level,

Level 3: the packet level.

The physical level interface requires the use of full-duplex, point-to-point synchronous circuit operating at 1200 bps to 48000 bps in accordance with X.21 or X.21 bis recommendations /1/. This thus provides a two-way simultaneous path between the DTE and the network node, viz. DCE.

The frame level procedures provide a means of error-free, in-sequence communication of information or packets generated and interpreted by the higher packet level interface. The frame level specifies the use of High-level Data Link Control (HDLC) compatible data link control procedures:

the formats used to enclose level 3 generated packets resulting in I frames;

the use of supervisory control frames to acknowledge or reject an incoming I frame, or to indicate the status of the device as a receiver; and

the control frames - so called Unnumbered frames - used to set up, disconnect or reset the link.

1.2 Packet Level Interface

The packet level interface provides for the simultaneous establishment and operation of a number of logical communication channels between a DTE and the DCE. These communication channels are referred to as permanent virtual circuits or (temporary) virtual calls, depending upon whether the logical connection is established on a permanent basis in agreement with the network administration at the time of subscription to service, or on a temporary basis as and when required. Procedures have been laid down for initializing the permanent virtual circuits (PVC), establishing, clearing and initializing virtual calls (VC), and for the transfer of user data on PVCs and VCs. Once a virtual call has been established, its operation is no different from that of a PVC.

A PVC or a VC, while providing a logical communication channel between the (DTE,DCE) pair, in fact provides an independent communication between a local DTE and some remote DTE specified at the time of subscription to service (in case of PVCs) or at the time of establishing the call (in case of VCs). This is achieved by establishing a similar logical PVC or VC at the remote DTE-DCE interface. The responsibility of establishing and maintaining a

one-to-one corresponding between the local PVC/VC and the remote PVC/VC rests with the network. Figure 1.2 provides a simple example of how a given DTE simultaneously communicates with other DTEs through the use of PVCs/VCs. Figure 1.3 illustrates how one such logical communication between two DTEs is established by associating a logical channel number with each of local and remote PVC or VC. The correspondence between the local and the remote PVC or VC is maintained by the network by, for example, establishing an intranet PVC or VC with its own logical channel number. Thus, as in Figure 1.3, user data generated by higher level software at DTE1, and transmitted over PVC/VC numbered x to DCE1 (upon being communicated over the corresponding intranet PVC/VC numbered a between DCE1 and DCE2) would be delivered by DCE2 to DTE2 on PVC/VC numbered y.

The packet level software has interfaces with the Link level access software (Level 2) as also the higher level software which issues commands for setting up (or disconnecting) VCs and for transmission and reception of user data. The Link level interface provides a reliable (viz., error-free and in-sequence) communication link between the DTE and the DCE over which packets pertaining to all the PVCs and VCs of the packet level interface are communicated as Information frames (see Figure 1.4).

1.3 Report Outline

This document reports our attempts to understand and interpret the above packet-level procedures with a view to implement the relevant software on a data terminal. The interpretations, written in the form of PASCAL procedures, are based upon the X.25 Recommendation approved by CCITT in March 1976 /1/. The revisions made in 1977 and 1980 (See /2/, /3/) shall be incorporated in a subsequent report. The implementation of these procedures from the viewpoint of the DCE shall have to consider its interface with the intranet protocol software. This shall be the subject of a report currently under preparation.

In Chapter 2 we discuss, qualitatively, the procedures to set up,

disconnect and operate a VC or a PVC. We also discuss the procedures for interchange of user data under flow control. Aspects of the procedures that are secondary to the operation of a PVC or VC have been omitted for simplicity. Chapter 3 discusses the implementation of these procedures (written in PASCAL and listed in the Appendix) and their interface with the higher level system software and with the Link level interface software.

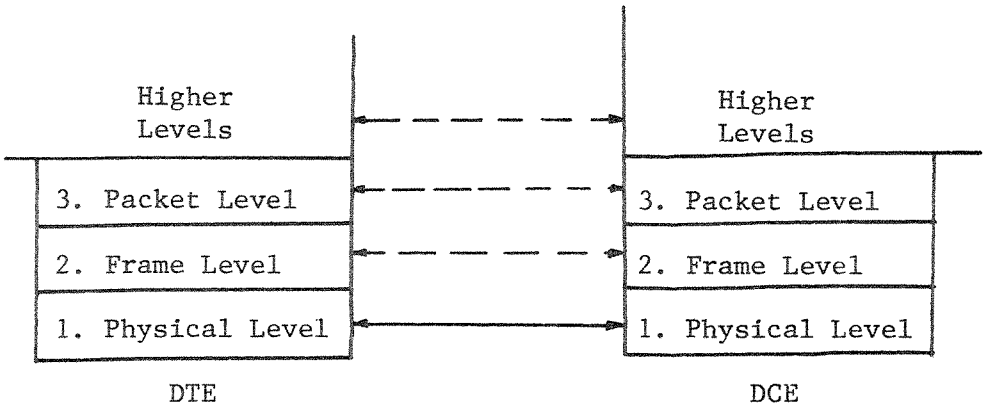


Figure 1.1: X.25 protocol layers

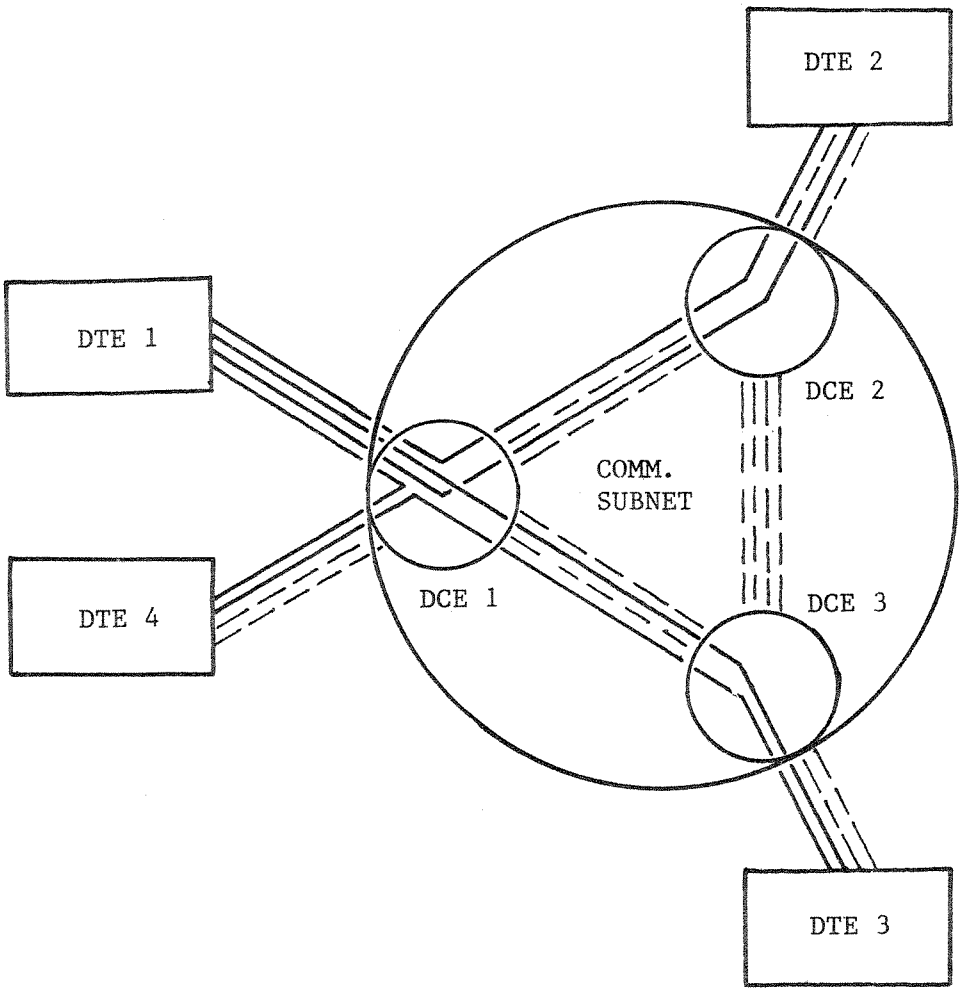
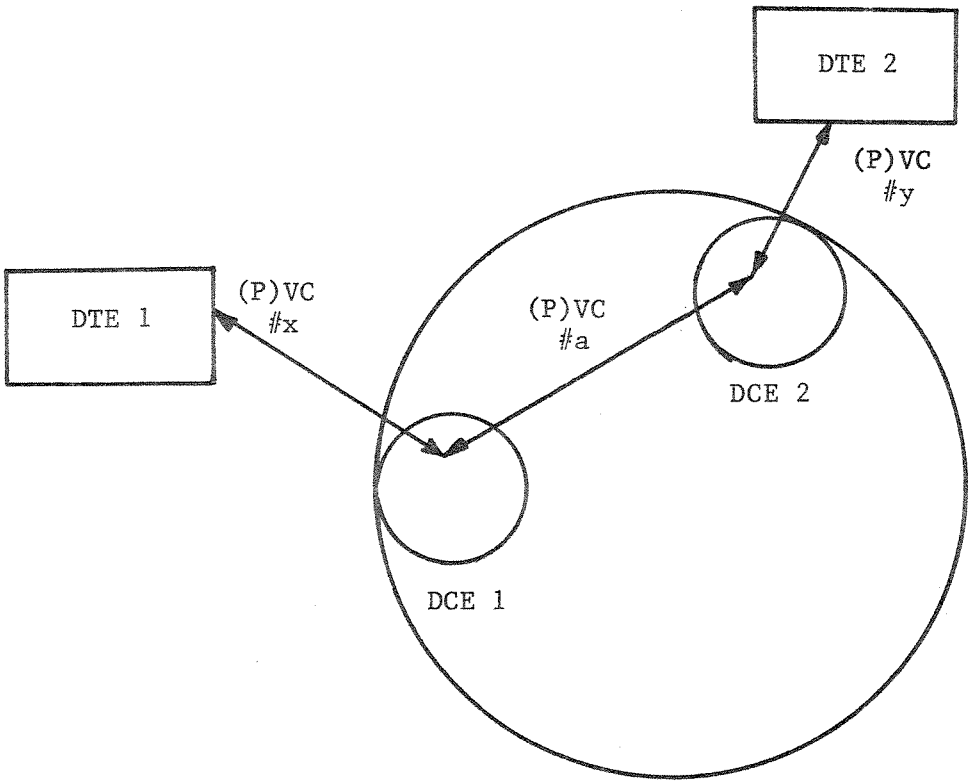


Figure 1.2: A DTE may establish multiple communication circuits/calls with one or more remote DTEs.

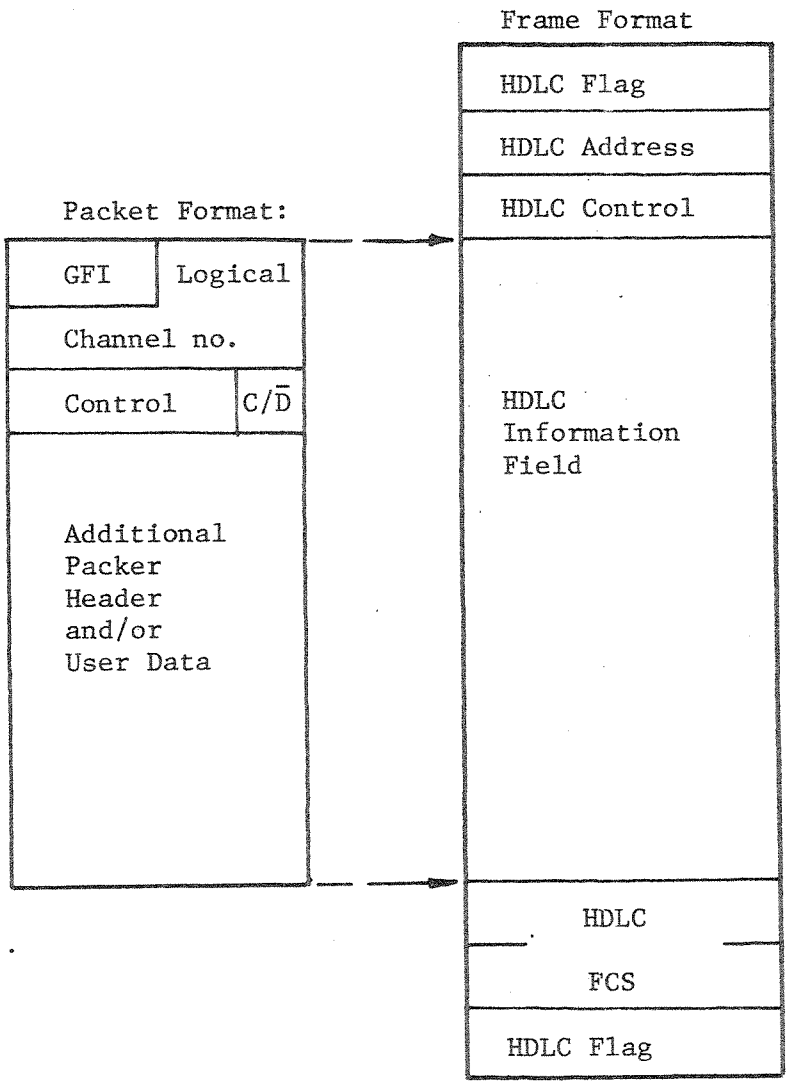


Correspondence maintained by Network

DCE 1: (DCE2,a) = f(DTE1,x)
(DTE1,x) = g(DCE2,a)

DCE 2: (DCE1,a) = f(DTE2,y)
(CTE2,y) = g(DCE1,a)

Figure 1.3: Illustration of a logical communication channel between two DTEs.



GFI: General Format Identifier
C/D̄: Control/Data

Figure 1.4: X.25 Level 2 frame and Level 3 Packet formats.

CHAPTER 2

PACKET LEVEL PROCEDURES

This chapter discusses the packet level procedures pertaining to the reinitilization of all permanent virtual circuits (PVCs) and virtual calls (VCs), i.e., the restart mechanism, the establishment and disconnection of VCs, viz, call set up and call clearing phases, the mechanism to reset a PVC or a VC, the transfer of interrupt data over a PVC or a VC, and the interchange of user data. But first we discuss the numbering of PVCs and VCs.

2.1 Logical Channel Number

As mentioned earlier in the last chapter, a PVC or a VC while establishing a circuit or a call between a DTE and the local DCE, in fact provides an independent channel between this DTE and some remote DTE. In other words packets pertaining to the DTE pair

local DTE, remote DTE

are communicated over a corresponding PVC or a VC which exists between

local DTE, local DCE

as also over the corresponding PVC/VC between

remote DCE, remote DTE

at the other end. The network protocols are responsible for communication between

local DCE, remote DCE

Each PVC or a VC has associated to it a Logical Channel Number (LCN). This number, instead of the address of the local and remote DTEs, is used to identify packets on the local interface and similarly at the remote DTE-DCE interface. An LCN is assigned to a VC at the time of requesting its establishment from amongst those available. Towards this, the range and number of LCNs that can be used by a DTE are pre-specified, just as are the LCNs for permanent VCs. The LCN appears in every packet, and is coded as 12 binary bits.

2.2 Call Establishment and Call Clearing

The establishment of a virtual call is initiated primarily by a DTE - either a local DTE or a remote DTE. In the latter case the local DCE acts on behalf of the remote DTE. The DTE initiating the process is called the 'calling DTE', while the other DTE is referred to as the 'called DTE'. If a logical channel number (LCN) is available (or equivalently if a VC is in a state of being 'Ready'), then it is assigned to the VC being established. The DTE transmits a Call Request to the local DCE, with the LCN and the DTE addresses specifically mentioned, and awaits a response from the local DCE. The network, in fact the remote DCE to which the remote DTE is connected then transmits to the remote DTE an Incoming Call with an available LCN relevant to the remote DCE - remote DTE interface. If the call is accepted by the remote DTE then it responds with a Call Accepted packet (with the LCN of the remote interface), thereby establishing a VC at the remote interface. The local DCE on receiving an intimation from the remote DCE then responds with a Call Connected packet this time mentioning, therein, the LCN of the local interface (see Figure 2.1). During this end-to-end VC establishment the network establishes the correspondence (may be indirectly through an intranet VC established between the local DCE and the remote DCE) between the local VC LCN and the LCN of the VC at the remote end. Subsequently, when user data or other control packets are to be communicated these are identified by the LCN of the VC relevant to the interface alone.

Figure 2.2 gives the state transitions during Call set-up as interpretted by/for a DTE. There are two differences between this and those given in /1, Figure 15(a)/. The first pertains to the Call Collision state (p5). The LCNs available for establishing VCs on a particular DCE/DTE interface are shared between the DTE and the DCE. Thus, it is quite likely that due to communication delays both the DTE and the DCE use the same available LCN in their Call Request and Incoming Call packets. This leads to collision (see Figure 2.3). The

recommendations stipulate that the DTE shall, in effect, ignore any such Incoming Call packet, while the DCE cancels its earlier Incoming Call packet and retransmits it, with a new LCN, if available. Thus, the DTE never enters the Call Collision state, as shown in Figure 2.2.

The second difference is in the manner in which the DTE accepts an Incoming Call. As shown in Figure 2.2 the DTE responds with a Call Accepted packet every time it receives an Incoming Call. If, however, the call is not acceptable then a call clearing phase is initiated. (It must be mentioned that the state diagrams given in Figure 15-17 in /1/ are from the DCE viewpoint while we are concerned with their implementation on a DTE).

Given the above scheme, it is transparent to the network DCEs whether there is, in fact, a VC set up between a given DTE pair. Thus, it is possible to have multiple connections being set up between a given (local DTE, remote DTE) pair. These multiple connections shall, however, use distinct VCs on each of the local and remote DTE/DCE interfaces.

At the time of end-to-end call set-up there is provision for the DTE to request (and possible negotiate with) either the local DCE or the remote DTE certain available facilities or parameters thereof. The recommendation on this are incomplete and are for further study by the CCITT group. For this reason their interpretation and implementation is being deferred. For the moment every Call Request and Incoming Call packets shall have a Facility Length of 0. Call User Data, as indicated in the Call Request Packet shall transparently be communicated to the remote DTE via the Incoming Call on the remote interface. The maximum length of such data is 16 bytes.

For any of the many reasons listed in Table 7/X.25 of the recommendation /1/ a virtual call can be cleared. This procedure, again, has an end-to-end relevance, in that the clearing of the VC on the local interface is incomplete till the corresponding intranet VC (between the local DCE and remote DCE) as also the VC on the remote interface are cleared. This is illustrated in Figure 2.4.

Figure 2.5 illustrates how clearling takes place when the two DTEs (more or less) simultaneously initiate the clear procedure - more or less in the sense that each of the devices is unaware of the other having initiated the process. (Note that, in contrast, when two DTEs initiate a call set up procedure, more or less, simultaneously then two distinct end-to-end calls would be set up). Figure 2.6 gives the state transition diagram pertaining to the implementation of the clear procedure from the DTE viewpoint. Here again it is assumed that the DTE shall positively respond with a Clear Confirmation packet whenever it receives a Clear Indication from the corresponding DCE.

2.3 The Restart Procedure

The restart procedure is used to simultaneously initialize the PVCs and VCs, that is to clear all virtual calls, and to reset all permanent virtual circuits. This may primarily be invoked when the DTE or the DCE detects a local procedural error. The DTE may transmit a DTE Restart Request to the corresponding DCE, and enters the state (r1) where it awaits either a Restart Confirmation or a similar request (DCE Restart Indication) from the local DCE. The local DCE would then attempt to clear (or reset) all the intranet calls (or circuits) that pertain to the PVCs and VCs on the DTE-DCE interface. If successful, it would have thereby selectively cleared (or reset) the VCs (or PVCs) at remote interfaces that correspond to the VCs (and PVCs) being restarted. Thereupon, the DCE confirms the restart of all VCs and PVCs by transmitting to the DTE (which initiated the process) a DCE Restart Confirmation packet. The DTE (as also the DCE) is then ready to establish new VCs or to transfer Data and Interrupt packets on permanent VCs, i.e. the logical channels are in the state of being "Ready" or in the case of PVCs, "Flow Control Ready". Figure 2.7 illustrates the restart procedure, while Figure 2.8 gives the corresponding state transition diagram from the viewpoint of a DTE. Also shown in these figures is the restarting of PVCs and VCs by a DCE.

It must be emphasized that the Restart procedure is local to the DTE-DCE interface. However, when a particular DTE-DCE interface is restarted, the other interfaces are effected but only to the extent that a few selected VCs (or PVCs) are cleared (or reset).

2.4 The Reset Procedure

This procedure is equally applicable to VCs and PVCs, and is used to reinitialize a VC or a PVC for data transfer. It is invoked whenever procedural errors are detected while transferring user Data or Interrupt packets over a channel, already in the data transfer state. The procedure is end-to-end, in that, if a VC is reset on an interface then the corresponding intranet VC between the local DCE and remote DCE and the VC at the remote interface are also reset. Further, the VC or the PVC is reset in both directions simultaneously.

Figure 2.9 gives the DTE state transitions during a reset operation. Figure 2.10 illustrates the resetting of a PVC or a VC by either a DTE or a DCE.

2.5 Data Transfer

User data generated by a higher user-level software is transmitted across the DTE-DCE interface as a Data packet on the appropriate PVC/VC. It becomes the responsibility of the network to transport it across to the appropriate remote DCE, and to deliver it to the destination DTE on the relevant PVC/VC. The rate of flow of data packet on each of the PVC or VC is controlled by an acknowledgement process existing across the DTE-DCE interface. For every data packet transmitted by a DTE or a DCE across the interface, an acknowledgement is expected. This acknowledgement may specifically have to be generated by the receiving device or may be implied in some data or control packet transmitted in the reverse direction. For this purpose data packets are sequentially numbered 0 to 7 (modulo 8). This number is referred to as the packet send sequences number, P(S).

The flow control of data packets in a given direction is achieved by prohibiting a device from transmitting a data packet if already there are W number of unacknowledged data packets. (W is referred to as the size of the window, and its value, common to all PVCs and VCs, is determined at the time of subscription to the network services, but in no case can it exceed 7). In other words the first data packet not authorized to cross the interface would have a P(S) = P(S) of the data packet last acknowledged + W + 1 (modulo 8).

In order to acknowledge an incoming data packet the receiving device transmits across the interface a number P(R), referred to as the packet receive sequence number. This number, P(R), conveys information on the status of the device as a receiver of data packets. A P(R), when received, is interpretted as an acknowledgement of data packets sequentially numbered upto but not including P(R). But before acknowledging the receipt of an incoming data packet the receiving device must determine whether the data packet is acceptable and, if so, whether an acknowledgement can be initiated. A data packet is valid and acceptable provided the receiver is 'not busy' and its P(S) is within the range (read this modulo 8)

$$\text{last transmitted } P(R) \leq P(S) < \text{last transmitted } P(R) + W.$$

It is important to note that the receiver does not expect the sequentially numbered data packets to arrive in the same sequence. (The motivation for providing this flexibility is not understandable, especially in view of the assumed, in fact, implied reliability of the Level 2 interface). If earlier a data packet with the same number had been received but not acknowledged then the current reception of the data packet is treated as being redundant. An acknowledgement can then be initiated provided the received data packet has a P(S) equal to the last trasmitted P(R). In other words, if the received data packet has a P(S) other than the one anticipated then it cannot be acknowledged until all data packets expected to arrive earlier (and sequentially numbered accordingly) have been received correctly. As an example let the last

transmitted P(R)=2, and W=4. A P(S) is valid provided $2 \leq P(S) \leq 5$. If a data packet with P(S) = 3 arrives then it is accepted, but an acknowledgment cannot be initiated. Subsequently, if again a data packet with P(S) = 3 arrives then it is treated as being redundant. If a P(S) = 2 arrives then an scknowledgement for data packets numbered 2 and 3 is initiated in the form of a P(R) = 4.

Acknowledgements are generally piggybacked onto an outgoing data packet in the form of a P(R) written into the control byte, viz. the packet type identifier. Alternatively, if no such transmission if pending or possible then a flow control DTE/DCE Receive Ready packet is transmitted. This packet can also be used to clear a busy condition of the device as a receiver, if it exists. An indication of the device becoming busy can be given by transmitting across the interface a DTE/DCE Receive Not Ready packet which also carries the packet receive sequence number P(R).

If a data packet with an invalid P(S) is received then the VC or PVC is reset.

The format for a data packet identifies two special bits, known as the Data Qualifier bit, Q, and the More Data bit, M. The Q bit may be used to multiplex data packets pertaining to two distinct streams onto one logical VC or PVC, or alternatively to identify a string of data packets belonging to one complete sequence or as being logically related to each other. The more data bit, M has a similar interpretation, except that it indicates that the packet with the M bit set to 1 and the one following it are part of the same message. It, thereby, implies that a packet with the M Bit set to 1 may be recombined with the packet sequentially numbered next. This provision enables varying limits on the length of the data field. That is, two DTEs may have different allowable data field lengths, and the allowable length at the interfaces may differ from that permissible within the network.

2.6 Interrupt Packets

Independent of the facility to transfer user data on each of the VCs or PVCs, there is provision to transfer in each direction short Interrupt packets that contains a user generated data byte. The receipt of each Interrupt packet is acknowledged by the receiver transmitting in the reverse direction a DTE/DCE Interrupt Confirmation packet. The number of Interrupt packets for which an acknowledgement is outstanding at any time is at most 1. Thus if an Interrupt packet is received by a device while it is processing an Interrupt received earlier then is construed as a procedural error and the VC/PVC is reset in both directions. Further, if a device were to receive an Interrupt Confirmation packet when no such Interrupt packet is pending acknowledgement then again the VC/PVC is reset.

It is particularly relevant to note at this point that if a packet pertaining to the interrupt facility were to be duplicated, for example, by the level 2 link level procedures then either there is loss of synchronization between the DTE and the DCE or the relevant VC/PVC is reset.

2.7 Optional Facilities

At the packet level interface many optional facilities exist. These may or may not be built into the network. Additionally, even if these have been built in as part of network characteristics, their availability to a particular DTE is determined either at the time of subscription of (even) at the time of establishment of a VC. These include:

Reverse Charging

Flow Control Parameters Selection

One-way Logical Channel

Packet Retransmission

In view of the fact that the recommendation on these are incomplete, discussion on these optional facilities, and their implementation is being deferred.

The more recent revisions to these recommendations envisage a mechanism for requesting and transmitting an end-to-end acknowledgement of user data packets (see /2/). Towards this purpose one of bits of the general format identifier is set to request an end-to-end acknowledgement for the particular data packet. This bit is referred as the D bit. (It may be pointed out that the handling of the D bit is more of a problem of the DCE than of DTE.)

These revisions also now include two new capabilities suited to communication of small amount of data. The first provides a datagram service to independent messages, while the other, referred to as the fast-select, provides for transfer of 128 bytes of user data at the time of VC establishment. These revisions shall be taken up when more details on these new services become available.

The Appendix provides a listing of the PASCAL procedures using which the procedures for initializing, setting up, clearing, data transfer, or resetting a VC or a PVC as described in Sections 2.2 - 2.6 may be implemented. In the next chapter we discuss these procedures briefly.

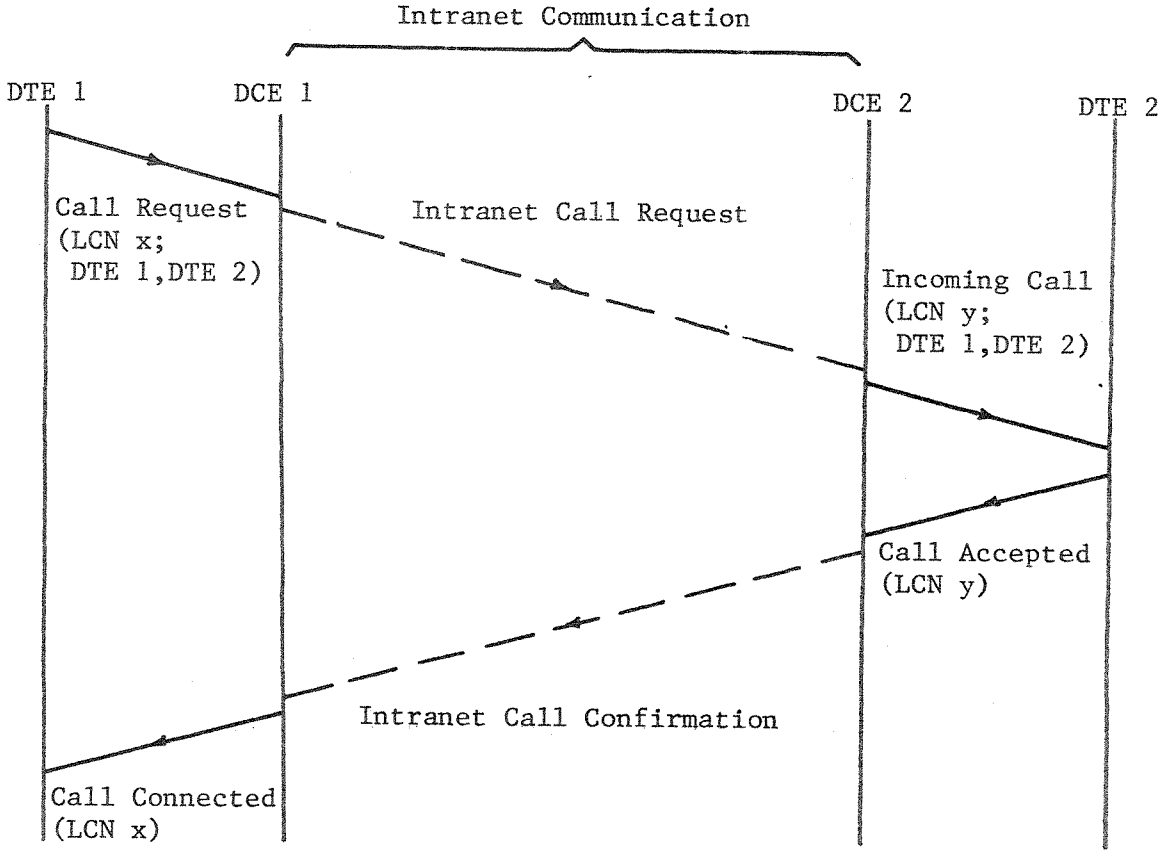


Figure 2.1: End-to-End Call Set-up.

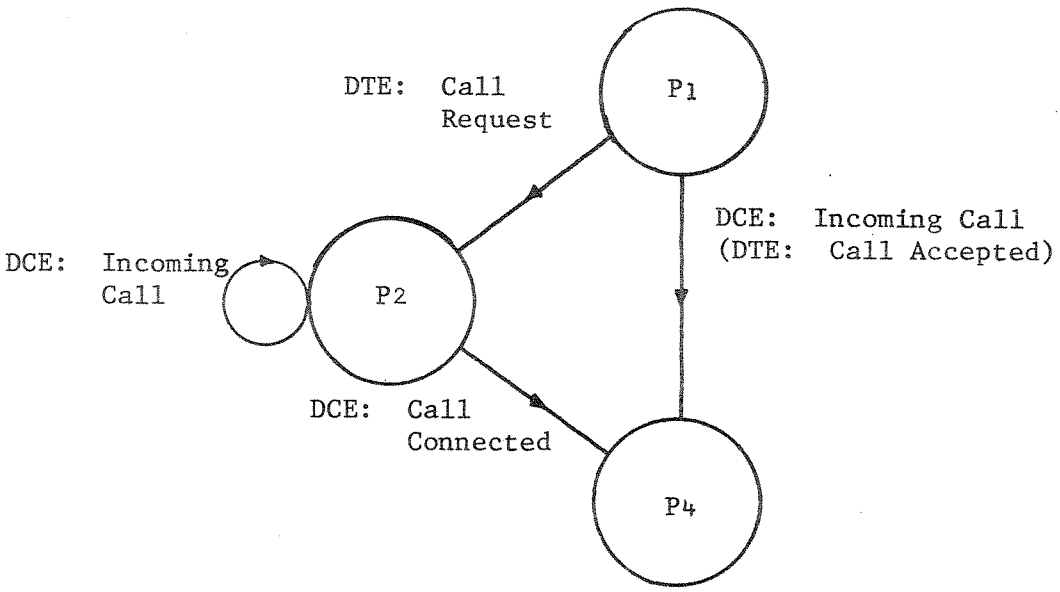


Figure 2.2: DTE State transitions during Call Set-up.

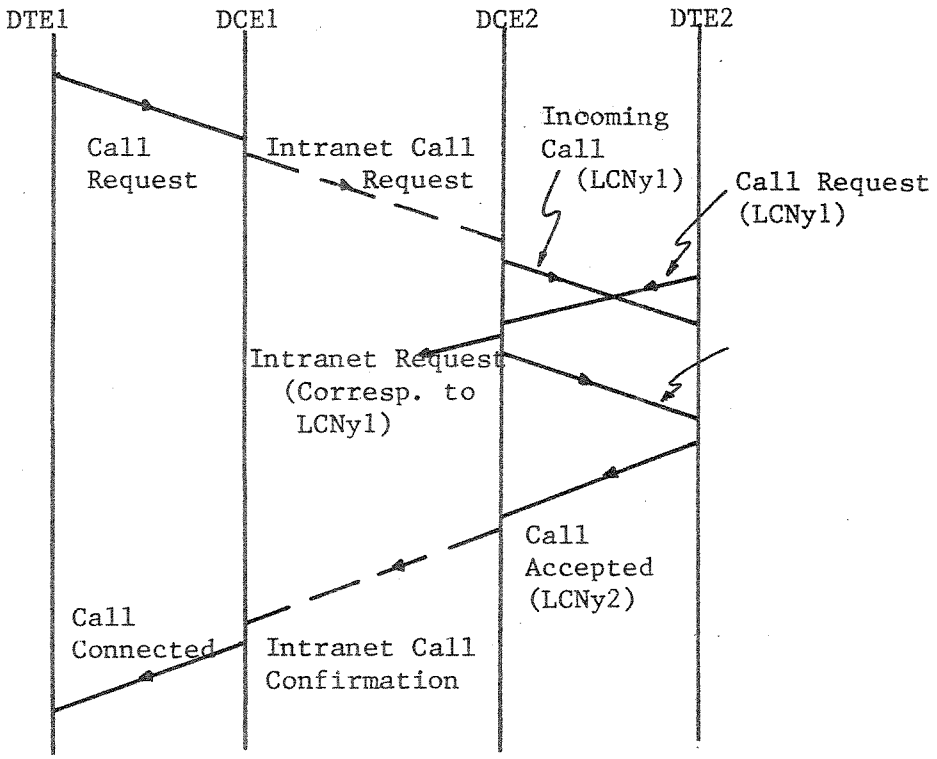


Figure 2.3: Call Set up with Collision.

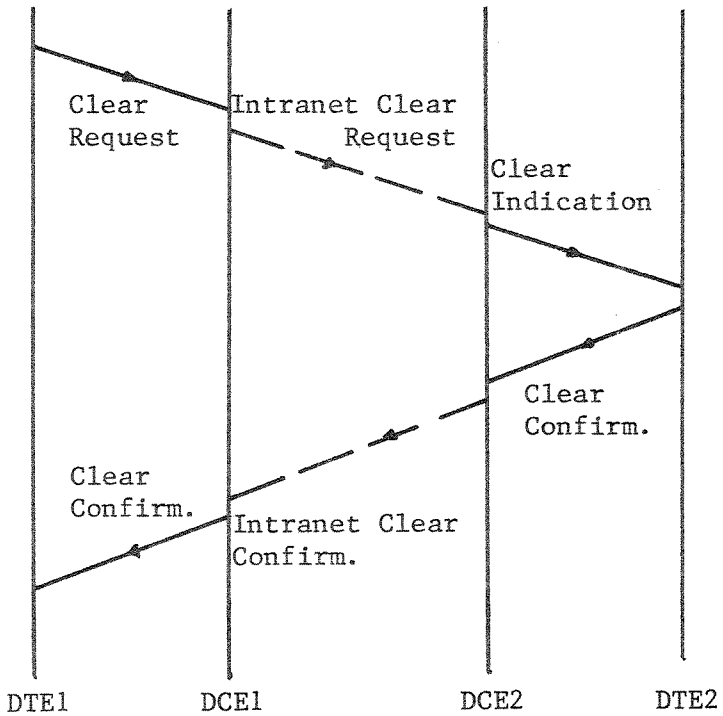


Figure 2.4: End-to-End Clearing of a VC.

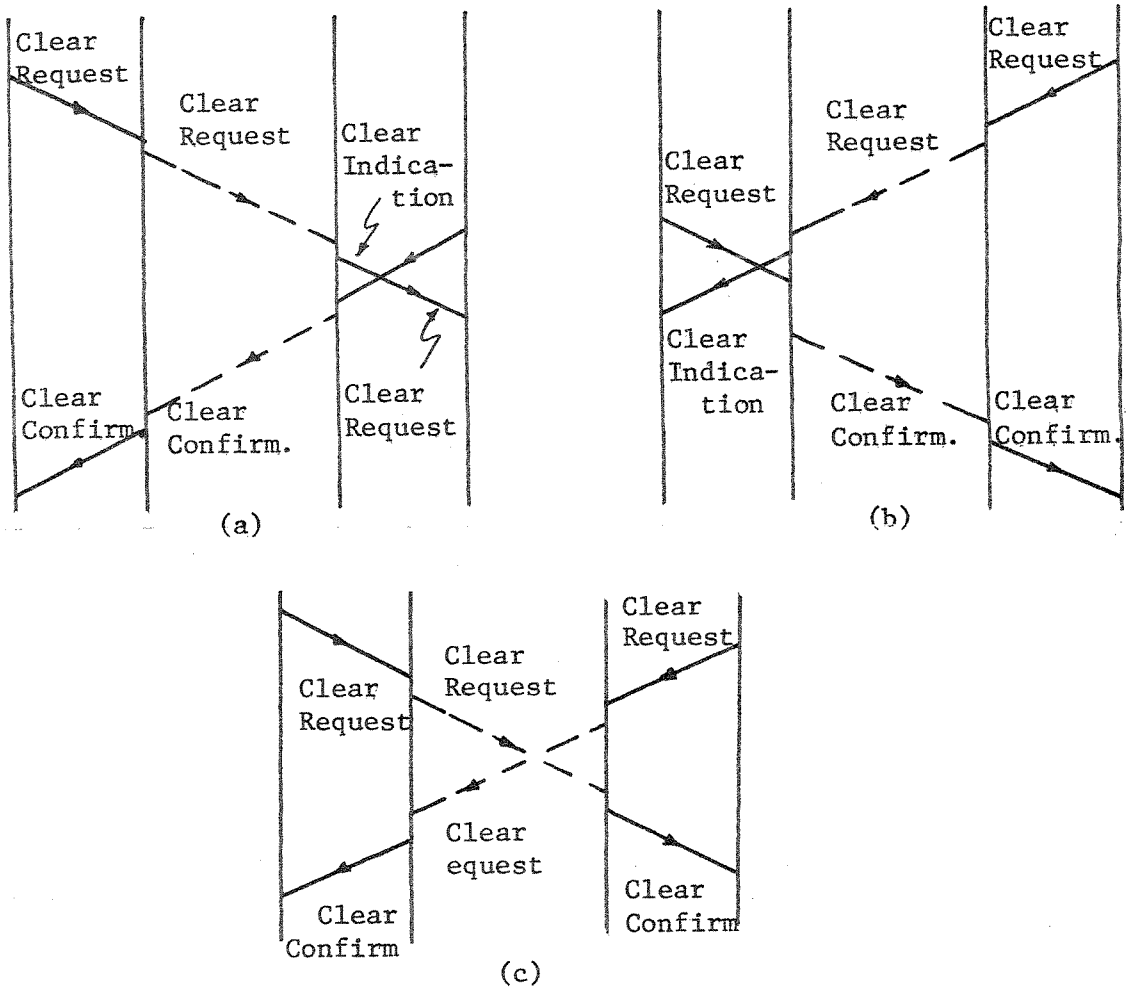


Figure 2.5: Clearing of a VC initiated by both DTEs simultaneously.

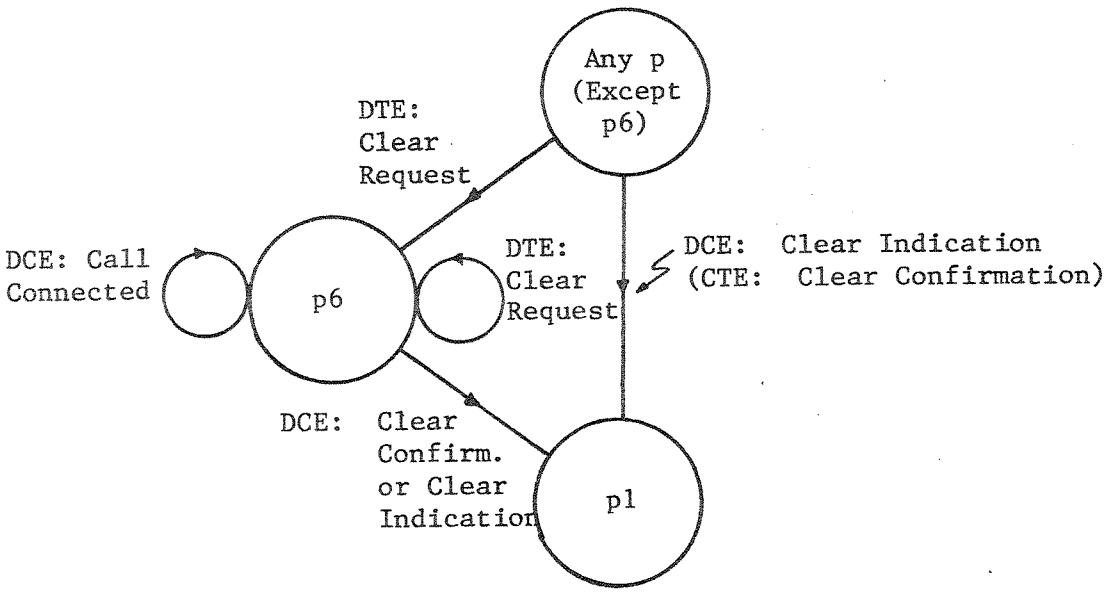
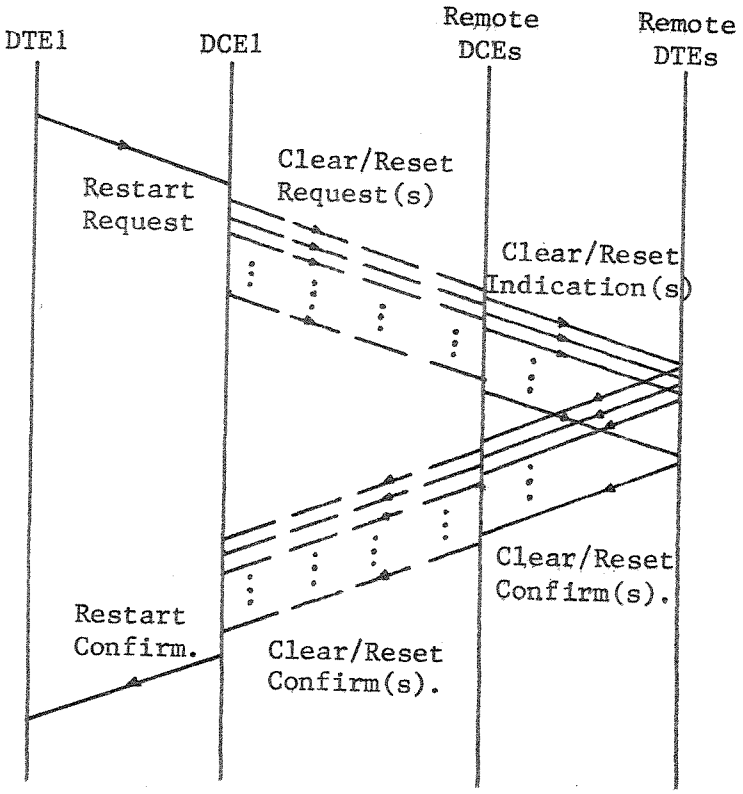
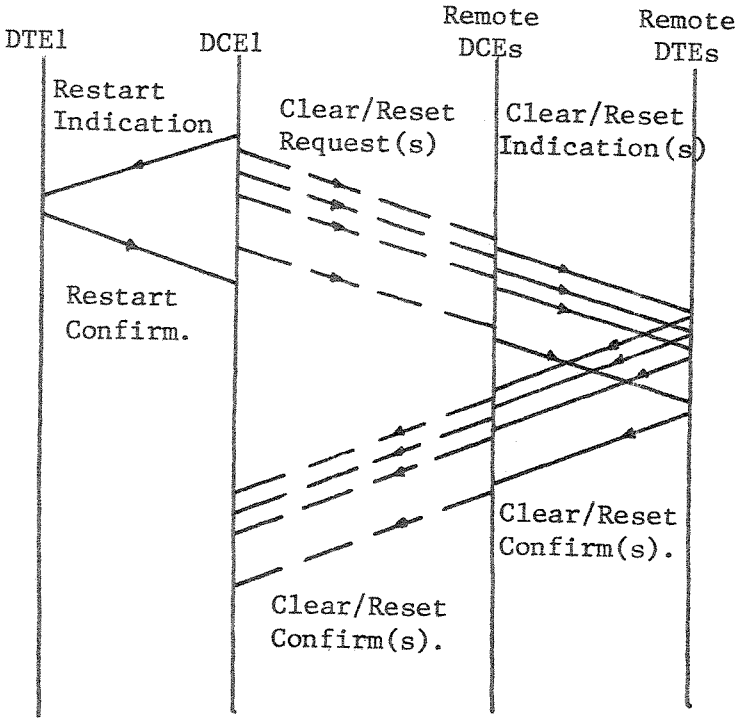


Figure 2.6: DTE State transitions during Clearing of a VC.



(a) DTE Initiated



(b) DCE Initiated

Figure 2.7: The Restart Procedure Illustrated.

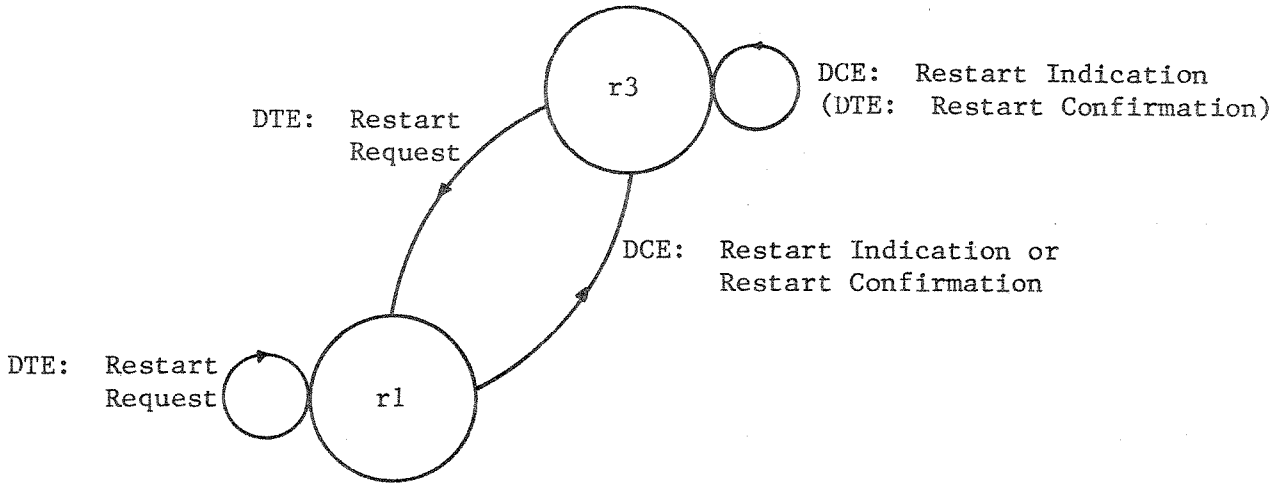


Figure 2.8: DTE State transitions during Restart.

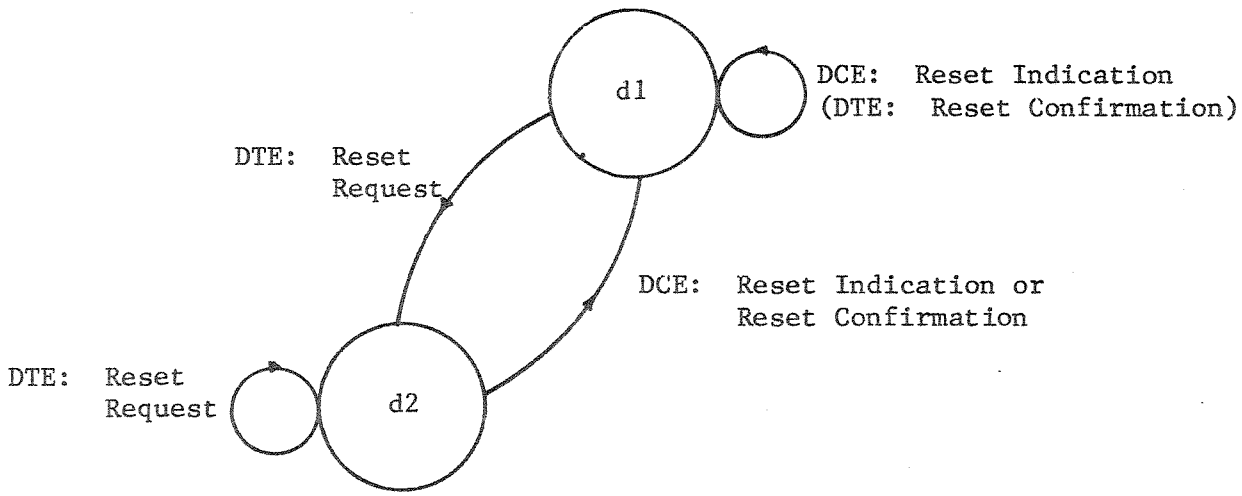
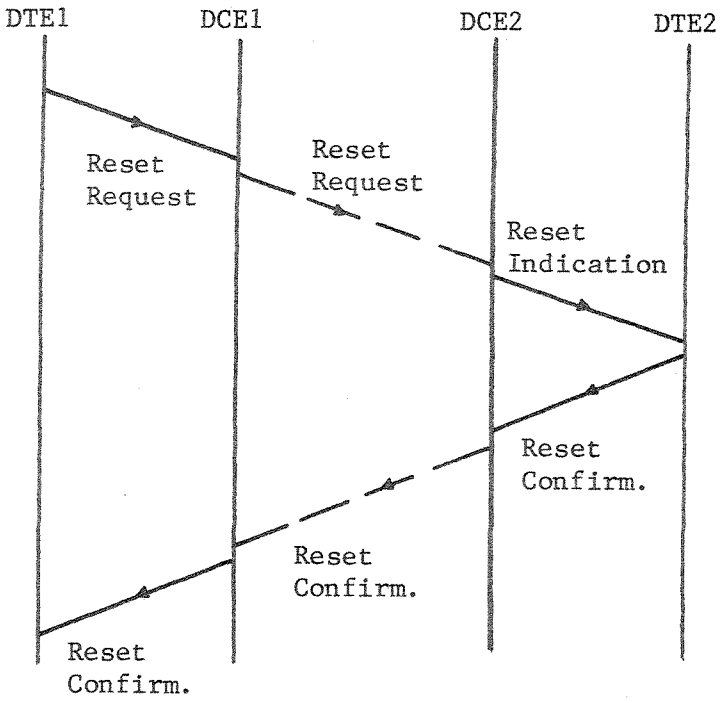
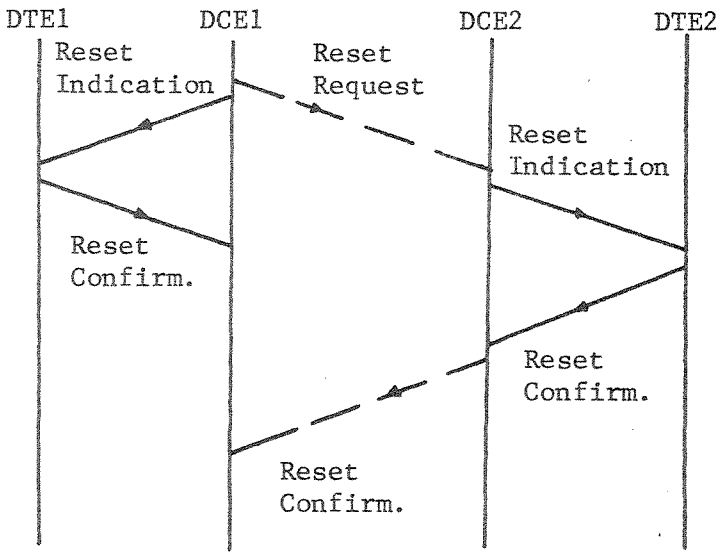


Figure 2.9: DTE State transitions during Reset.



(a) DTE Initiated



(b) DCE Initiated

Figure 2.10: The Reset Procedure.

CHAPTER 3

A PROPOSED IMPLEMENTATION

In this chapter we discuss an implementation of the X.25 Packet Level procedures. The attempt here is to give a more definite interpretation to these recommendations rather than to write a code that is directly implementable. PASCAL as a programming language is not well suited for real time programming. But because of its structured constructs it makes specification of an algorithm simple, readable, and easy to modify. The actual implementation would be in some real-time programming language or in an assembly language, capable of handling interrupts. In the Appendix we give a collection of procedures written in PASCAL that are invoked as it becomes necessary. No attempt is made, therein, to show the nesting of these procedures. There is a large set of global variables that are accessed/modified by the various procedures. Further, a number of procedures pertaining to the lower frame level X.25 recommendations are assumed to be available. Additionally, certain basic procedures/functions that pertain to queue/stack handling, or to accessing specified bits of packet are assumed.

3.1 Elementary Procedures

In the following we list the elementary procedures that are assumed to be available.

Given a buffer address the following functions provide the specific components of a packet contained in the buffer:

```
Compute  Packet type (k : Buffer Address) : byte;
Compute  Channel no. (k : Buffer Address) : Channel no;
Compute  Calling DTE Address (k : Buffer Address) : DTE Address;
Compute  Called DTE Address (k : Buffer Address) : DTE Address;
Compute  Q(k : Buffer Address) : bit;
Compute  M(k : Buffer Address) : bit;
Compute  User Data (k : Buffer Address) : Data string;
```

Compute Interrupt Data (k : Buffer Address) : byte;

Compute PR(k : Buffer Address) : 0..7;

Compute PS(k : Buffer Address) : 0..7.

Similarly, the following procedures write the corresponding packet components into a buffer:

Write GFI(k : Buffer Address);

Write Packet type (k : Buffer Address ; t : byte);

Write Channel no (k : Buffer Address ; x : Channel no);

Write Calling DTE Address (k : Buffer Address ; i : DTE Address);

Write Called DTE Address (k : Buffer Address ; j : DTE Address);

Write Q(k : Buffer Address ; Q : bit);

Write M(k : Buffer Address ; M : bit);

Write Use Data (k : Buffer Address ; User Data : Data string);

Write Interrupt Data (k : Buffer Address ; Interrupt data : byte);

Write PR(k : Buffer Address ; PR : 0..7);

Write PS(k : Buffer Address ; PS : 0..7).

Write Clearing Cause (k : Buffer Address ; y : byte);

Write Resetting Cause (k : Buffer Address ; y : byte);

Write Diagnostic Code (k : Buffer Address ; y : byte):

Write Restarting Code (k : Buffer Address ; y : byte).

A list of all available logical channels needs to be maintained. This list is initialized to include all numbers ranging from Max Perm VC no + 1 to MaxVC no by invoking procedure Initialize Available Channels. Further the

procedure Get Channel (var x : Channel no);

procedure Return Channel (x : Channel no) are used to obtain the number of an available logical channel, and to return to the list a logical channel number as soon as the corresponding VC becomes 'Ready'. Since the X.25 Recommendations do not suggest the order in which the logical channels are to be used

(as far as a DTE is concerned) the list could be implemented in any manner. (in the above MaxPerm VCno and Max no are the largest numbers corresponding to a PVC or a VC, respectively.)

The following functions/procedures enable specific operations to be performed on a given queue:

```
function   Head of (Q : Queue Pointer) : Buffer Address;
function   Tail of (Q : Queue Pointer) : Buffer Address;
function   Is Empty (Q : Queue Pointer) : Boolean;
function   Length (Q : Queue Pointer) : integer;
procedure  Add First (frame : Buffer Address ; Q : Queue Pointer);
procedure  Add Last (frame : Buffer Address ; Q : Queue Pointer);
procedure  Behead (Q : Queue Pointer);
procedure  BeTail (Q : Queue Pointer);
procedure  Merge (Q1, Q2 : Queue Pointer);
procedure  Initialize Q (Q : Queue Pointer).
```

While the operations of these routines are obvious, it needs to be mentioned that the Merge procedure achieves:

```
while length (Q1) > 0
   do begin  Add First (Tail of (Q1) , Q2);
             BeTail (Q1)
      end
```

A statement used often in the procedures listed in the Appendix is

```
Incase   C1 : S1 ; C2 : S2 ; ... ; Cn : Sn ; end
```

or

```
Incase   C1 : S1 ; C2 : S2 ; ... ; Cn : Sn ; else SO end.
```

This statement is not available in PASCAL, and differs substantially from its case statement. Its meaning is

```
if C1 then S1 else if C2 then S2 else ... if Cn then Sn;
if C1 then S1 else if C2 then S2 else ... if Cn then Sn else SO,
```

respectively, where Ci is a Boolean valued expression and Si is any statement. The Incase statement has been introduced to improve readability.

3.2 Level 3 - Level 2 Interface

The Level 2 software may monitor the following variables which define the state of Level 2:

```
var   Level 2 Enabled,
      Link Setup In Progress,
      Data Tx In Progress,
      Link Disc In Progress  : Boolean
```

and may issue commands to bring the level 2 into the active channel state, to initiate Link Setup or Link Disconnection procedures, or to transmit or receive a packet. It does so by activating one of the following:

```
procedure  Enable Level 2
procedure  Enable Packet Interchange
procedure  Disable Packet Interchange
procedure  Transmit Packet (k : Buffer Address)
procedure  Receive Packet
```

The level 2 in turn responds on executing the command (irrespective of the outcome) by activating the appropriate

```
procedure  Return Enable Level 2
procedure  Return Enable Packet Interchange
procedure  Return Disable Packet Interchange
procedure  Packet Transmitted (k : Buffer Address)
procedure  Packet Received (k : Buffer Address)
procedure  Packet Not Transmitted (k : Buffer Address)
procedure  Packet Not Received.
```

In order to bring the link into the active channel state

```
procedure  Enable Level 2
```

is activated which initiates the sequence of initializing the level 2 software at both the ends of the X.25 interface. Normally, a non-zero finite, though random, amount of time clapses before the attempt is complete. It may not be

successful if the physical level interface cannot be established. In either case the level 2 activates

procedure Return Enable Level 2

with the outcome of the attempt recorded in

var Level 2 Enabled : Boolean

An one consequence of enabling Level 2 software, the pool of available buffers is initialized. These buffers are used to receive incoming frames and to store outgoing frames. These are also shared by the Level 3 software through simple routines to obtain a free buffer or to return a buffer:

procedure Get Pkt Buffer (var k : Buffer Address);

procedure Return Pkt Buffer (k : Buffer Address).

As stated earlier, the level 2 software at the other end of the interface is also initialized. It implies that the local level2 may be enabled as a consequence of a similar command executed at the other end. In which case, again

Level 2 Enabled := true

and the procedure Return Enable Level 2 is activated.

Once level 2 has been enabled level 3 software may issue commands for setting up of disconnecting the link (asynchronous response mode) by invoking the appropriate

procedure Enable Packet Interchange;

procedure Disable Packet Interchange.

(Of course, the link may be set up or disconnected at the behest of the device at the other end of the interface.) Substantial time may elapse before the link is set up or disconnected, and again the attempt may not succeed. In either case the appropriate

procedure Return Enable Packet Interchange;

procedure Return Disable Packet Interchange

is activated with the outcome of the attempt available in

var Data Tx In Progress : Boolean.

The relevant Return procedure is also executed in case packet interchange is

enabled/disabled by the device at the other end of the interface.

Once information transfer phase has been established Level 3 can use the packet communication capability provided by level 2 by invoking either of

procedure Transmit Packet (k : Buffer Address),

procedure Receive Packet.

It may be quite some time before the indicated packet is transmitted, and similarly before a packet is received.

Transmit Packet is used to instruct the level 2 to transmit at the earliest opportunity the packet contained in the buffer identified by the argument. It is assumed that the packet contents are uniquely determinable by the buffer address. When an acknowledgement to a packet, transmitted earlier as an I frame, is received the level 2 software activates

procedure Packet Transmitted (k : Buffer Address)

defined and maintained by level 3. It must be noted that level 2 does not move packet data from one buffer to another while handling its transmission or while awaiting acknowledgement.

Through activation of the procedure Receive Packet the level 3 software authorizes level 2 to receive and accept an additional packet (as an I frame). Whenever an I frame is accepted the level 2 passes it on to level 3 by activating the level 3

procedure Packet Received (k : Buffer Address).

All unacknowledged packets, as also excess receive authorizations are returned to level 3 as soon as the link is disconnected through the repeated activation of level 3

procedure Packet Not Transmitted (k : Buffer Address);

procedure Packet Not Received;

but before the procedure Return Disable Packet Interchange procedure is called.

For greater details on the interface and the implementation of Level 2 see the report /4/. (Therein certain modifications will have to be made in view of

the fact that Level 2 and Level 3 software share the same Buffer Pool. As one example the procedure named Receive Packet will have to be rewritten, and all occurrences of the procedure named Packet Not Received modified.)

3.3 Level 3 - Higher Level Interface

The software utilizing the communication capability provided by X.25 level 3 software shall be referred to as the transport level software. It can monitor the following variables defined and maintained by Level 3:

```
var  Level 3 Enabled : Boolean;
     State of VCs : (r1, r3);
     Prev State : array [1..Max VC no] of (p1, p2, p4, p6)
     State: array [1..Max VC no] of record p:(p1, p2, p4, p6);
                                           d:(d1, d2)
                                    end
     Interrupt Out Buffer Full,
     Interrupt in Buffer Full : array [1..Max VC no] of Boolean.
```

These variables define the state of the packet level software or that of the individual PVC/VC. Further, the transport level software may issue commands to

.initialize level 3 software,

.restart all PVCs and VCs,

.establish a VC between itself and another DTE,

.clear a VC,

.transmit user data on a PVC/VC,

.transmit interrupt data byte on a PVC/VC,

.authorize the reception of user data on a VC/PVC, or to

.accept an incoming interrupt data byte on a VC/PVC.

This it does by invoking, respectively, the following Level 3 procedures. (Indicated within parenthesis are the conditions to be satisfied before the particular procedure can be activated.)

```
procedure  Enable Level 3 {not Level 3 Enabled};

procedure  Restart VCs {Level 3 Enabled and State of VCs = r3};

procedure  Establish VC (var x : Channel no ; i, j, : DTE Address)
  {Level 3 Enabled and (State of VCs = r3)};

procedure  Clear VC (x : Channel no)
  {Level 3 Enabled and (State of VCs = r3)};

procedure  Transmit Data Message (x : Channel no; Q, M : bit; User Data : Data String)
  {Level 3 Enabled and (State of VCs = r3) and (State [x].p = p4)};

procedure  Transmit Interrupt Message (x : Channel no ; Interrupt Data : byte )
  {Level 3 Enabled and (State of VCs = r3) and
   (State [x].p = p4) and not Interrupt Out Buffer Full [x]};

procedure  Receive Data Message (x : Channel no)
  {Level 3 Enabled and (State of VCs = r3) and
   (State [x].p = p4)};
```

The packet level software, on completing the attempt to execute these commands responds by setting the above mentioned variables appropriately, and by invoking one of the following transport level procedures:

```
procedure  Return Enable Level 3;

procedure  VCs Restarted;

procedure  VC Established (x : Channel no ; i, j, : DTE Address);

procedure  VC Cleared (x : Channel no);

procedure  Data Message Transmitted (x : Channel no ; Q. M : bit ;
                                     User Data : Data String);

procedure  Data Message Not Transmitted (x : Channel no ; Q, M : bit;
                                         User Date : Data String);

procedure  Interrupt Message Transmitted (x : Channel no; Interrupt Data : byte);

procedure  Interrupt Message Not Transmitted (x : Channel no; Interrupt Data : byte);

procedure  Data Message Received (x : Channel no ; Q, M : bit ; User Data : Data String);
```

When the packet level software receives an Interrupt message then it invokes the transport level

procedure Interrupt Message Received (x : Channel no ; Interrupt Data : byte)

to which the transport level software responds by activating

procedure Interrupt Message Accepted (x : Channel no)

provided it accepted the Interrupt data.

A description of the action's taken by the packet level software upon executing the above commands (successfully or otherwise) follows.

Provided the Level 3 is not enabled activation of the

procedure Enable Level 3

results in initializing the level 3 software. This is done by first ensuring that Level 2 is enabled and that interchange of Level 3 generated packets is possible via Level 2 (i.e. Level 2 is in Information transfer phase). Among the actions taken:

.the queue of incoming packets, Pkt In Q, which are yet to be processed
is made empty;

.the number of packets Level 2 is authorized to receive is initialized to,
say, 8;

.all PVCs are moved to the data transfer state, viz. (p4, d1);

.all VCs are in a state of being Ready (p1); and

.the restart of all PVCs/VCs is not in progress,
i.e., State of VCs = r3 - a newly defined state.

Once Level 3 is enabled the transport level

procedure Return Enable Level 3

is activated with

Level 3 Enabled := true.

Alternatively, if Level 3 cannot be enabled then, too, the Return Enable Level 3 is activated but with Level 3 Enabled = false.

If the local DCE were to initiate the enabling of Level 3, then, irrespective of the outcome, the procedure Return Enable Level 3 would be activated with the variable Level 3 Enabled appropriately set.

Restarting PVCs/VCs

The procedure Restart VCs when activated, initiates the process of restarting all VCs and PVCs. Provided the attempt is successful the transport level procedure named VCs Restarted is activated. It leaves all PVCs in the 'data transfer state' and all the VCs 'Ready'. In effect, all PVCs are reset while all VCs are cleared. The variable State of VCs is again set to r3. Similarly, when the DCE attempts to restart all the PVCs and VCs, the DTE responds with a confirmation, thereof, and informs the transport level by activating the procedure VCs Restarted.

Establishing a VC

While requiring to establish (may be redundantly) a VC between the DTE and some remote DTE, the transport level software activates the procedure named Establish VC with the local DTE address, remote DTE address specified as its argument. Level 3 immediately assigns a logical VC number and provides it in x(which, note, is variable (address) substituted). Subsequently, when the VC is end-to-end established the level 3 activates the transport level procedure VC Established (x: Channel no, i, j : DTE Address).

It is relevant to note here that correspondance between the VC channel number and the local and remote DTE addresses is maintained by the transport-level software, but is 'read-only' accessible to the level 3 software. This correspondence may be maintained by defining, for example,
var Local DTE, remote DTE : array[1..MaxVCno] of DTE Address.

If the VC cannot be established then it is expected that DCE would clear the VC, and in that case the procedure VC Cleared would be activated.

If a remote DTE, or on its behalf the local DCE, were to attempt establishing a call then this DTE (level 3) always responds positively, and the Level 3

software activates the VC Established procedure with the VC channel number, local DTE address, and remote DTE address as its arguments, as before. If the transport level does not favor this establishment of a VC, it could instruct the packet Level 3 to clear the VC.

Clearing a VC

The transport level software may attempt the clearing of a VC by simply activating

procedure Clear VC (x : Channel no).

The packet level software, upon receiving a confirmation to the that effect from the remote DTE (in fact, the local DCE acting on behalf of the remote DTE), informs the transport level by activating

procedure VC Cleared (x : Channel no)

The logical channel so cleared is put in a state of 'ready'. If a request to clear a VC is received from the remote DTE through a DCE initiated Clear Indication packet this DTE packet level software always responds with a confirmation to it and informs the transport level, as before. But before the clear operation is put into effect the packet layer returns all untransmitted user-data and interrupt messages by repeatedly (if necessary) activating

procedure Data Message Not Transmitted (x : Channel no ; Q, M : bit;
User Data ; Data String);

procedure Interrupt Message Not Transmitted (x : Channel no; Interrupt Data ; byte);

Transmitting Interrupt Data

The packet layer may be instructed to transmit a user interrupt message, provided the relevant logical channel has been established and there is no outstanding interrupt message. This can be done by activating the

procedure Transmit Interrupt Message (x : Channel no ; Interrupt Data : byte).

The packet level software transmits the Interrupt data at the earliest opportunity. Upon receiving a confirmation from the remote DTE the packet level activates:

procedure Interrupt Message Transmitted (x : Channel no)

thereby indicating that it is now ready to transmit yet another Interrupt message.

Receiving Interrupt Data

Upon receiving an interrupt message from the remote DTE (local DCE) the packet level software informs the transport level software of its receipt, and awaits its acceptance. The transport level accepts the incoming Interrupt message by activating the packet level

procedure Interrupt Message Accepted (x : Channel no)

The receipt of the Interrupt is now confirmed with the packet level transmitting a DTE Interrupt Confirmation packet to the local DCE.

Transmitting User Data

The transmission of a user data packet can be initiated by activating procedure Transmit Data Message (x : Channel no ; Q, M : bit ; User Data : Data String). This is so, provided the relevant logical channel has been established. The packet level software then transmits the data packet at the earliest opportunity. This happens as soon as the VC is not being reset, corresponding to the VC the DCE is known to be not busy, and the number of outstanding data packets is less than the number of allowable unacknowledged packets.

An acknowledgement to an earlier transmitted data packet is received in the form of a PR. Whenever one or more data packets are so acknowledged the packet level informs the transport level by repeatedly (if necessarily) activating

procedure Data Message Transmitted (x : Channel no ; Q, M : bit ;
User Data : Data String).

Receiving Data Messages

The transport level software may authorize the packet level to receive an additional user-data packet by activating

procedure Receive Data Message (x : Channel no)

When a data packet is received from the local DCE then, privided it is a fresh

data packet and its P(S) is within the permissible window, this data packet is made available to the transport level through the activation of

procedure Data Message Received (x : Channel no ; Q, M : bit ;
User Data : Data String).

The number of authorization to receive data packets on the particular PVC/VC is reduced by 1.

From the procedures listed in Appendix it may be noted that the condition of the DTE being busy on a particular logical channel is tied up to the number of data packets it is authorized to receive. As soon as this number falls to zero the DTE is said to become busy.

CHAPTER 4

CONCLUSIONS AND ADDITIONAL COMMENTS

4.1 Some Comments

We have presented here an interpretation of the X.25 Packet level protocols in the form of typical histories, state transition diagrams, and finally as a collection of PASCAL procedures. While these procedures would be greatly helpful in carrying out an implementation of the packet level software, these do not constitute an implementation.

To keep the presentation simple, we have avoided discussion of optional facilities. Discussion on the recent X.25 revisions pertaining to the end-to-end acknowledgement (the D bit) and on the datagram/fast select services has been postponed to a subsequent report. Furthermore, whenever a reset, clear or a restart operation is initiated the transmitted control message does not include the reasons why such a procedure is being initiated. Nor does the receiver interpret any of the fields pertaining to the restart/clear/reset cause or code.

It is important to emphasize again that the interpretation of the packet level procedure is given from the DTE viewpoint. The interpretation for the DCE's viewpoint is similar. However, its implementation on a DCE would be somewhat more complex, in that its interface with other packet level softwares relevant to other DTE's and DCE's would have to be considered, simultaneously.

4.2 Duplication of Packets

The Link Access Procedures (level 2) provide the basic support for the packet level software to be able to establish (or clear) virtual circuits or to transfer user interrupt and data messages over a PVC or a VC. In considering the effectiveness of the packet level software to provide these services one would have to consider the support service provided by Level 2. The primary function of Level 2 is to provide error-free insequence communication of packets. However, in attempting to do so it may deliver (or receive) redundant copies of

packets. This duplication of packets may occur whenever the link level communication is reset. For then, the trasmitter of Information (I) frames is unaware of the status of the receiver; and upon the link being reset it resumes transmission with the first unacknowledged I frame. Level 2 receiver has no way of determining whether this frame contains a fresh packet of one that was already received and passed over to the packet layer. It is for the packet layer to determine, if it can, whether a received packet is a duplicate copy or a fresh one. From the viewpoint of the transmit mechanism in a packet layer, it cannot be sure if duplicate copies of its packets are being delivered by the link level transmitter.

The way packet level protocols is defined/designed, the corresponding receive software does not attempt to detect duplication of packets, except in certain obvious cases.

We shall take up the duplication of control messages pertaining to packet level protocol, first. But, it is pertinent to immediately point out that due to procedural errors at the other end, a DTE (or DCE) may receive unexpected control messages. The primary concern of the designers of packets level protocol has been to handle such unwarranted transmissions. It, therefore, recommends initiation of corrective action (aimed at achieving synchronism) whenever an unexpected control message arrives. The receiver software does not attempt to determine whether an unanticipated control message is a consequence of some procedural error (at the transmitter) or that it, in fact, is a duplicate copy of a message received earlier. It would have been more efficient if such a determination were possible. Thus, in all cases unintended duplications (by the level 2) when received are interpretted as a consequence of proecdural errors at the packet level transmitter end. (The protocol does not allow for purposeful retransmission of a CALL REQUEST whereas a CLEAR REQUEST (as also DTE initiated Reset and Restart requests) may be retransmitted and re-received.)

Consider now the communication of Interrupt packets on a logical channel

and the associated packets. From the DTE's viewpoint, it expects only one DCE Interrupt at a time. If two copies of the same DCE Interrupt packet are received then the link is reset. Similarly the DTE expects only one DCE Interrupt Confirmation in response to its DTE Interrupt packet. As before, if a DCE Interrupt Confirmation were to be redundantly received, then the VC or PVC is reset, as the receipt of the second confirmation packet would be treated as a procedural error. This situation can be remedied by sequentially numbering the Interrupt and the corresponding Confirmation packets using, for example, a modulo 2 numbering scheme. But notice what may happen when the window size at the link level is large (this implies that the length of the sequence of messages redundantly received may be large). If the confirmation packets to Interrupt packets numbered 0 and 1 are duplicated then they may be erroneously interpretted as acknowledgements to subsequently transmitted fresh interrupt packets numbered 0 and 1.

Note that the maximum number of outstanding (unacknowledged) Information (I) frames at the link level is finite, and that duplication of packet is somewhat orderly. For example, two packets numbered P1 and P2 may be received redundantly as

P1, P2

P1, P1, P2

P1, P2, P2

or P1, P2, P1, P2,

but certainly not as, for example

P1, P2, P2, P1, P2.

Thus, it does seem that it should be possible to design a protocol that would enable detection of duplicate Interrupt packets. The maximum number of outstanding Interrupt packets, k_p, the maximum number of outstanding information frames, k_1, at the link level, and the (modulo n) sequential numbering scheme for Interrupt scheme would have to satisfy certain constraints. It appears that if

$k_p \leq n$,

$k_1 < n$,

and confirmation packets are initiated for every Interrupt packet then duplication of Interrupt and confirmation packets by the link level can be detected.

A similar scheme for detecting duplicate User Data packets and the corresponding acknowledgement packets can be devised. The constraints are likely to be different as the acknowledgement scheme here is different: An acknowledgement to a User data packet may be delayed, and an acknowledgement numbered i acknowledges User Data packets numbered i-1, i-2, and so on. It appears that the constraints on k_1, w, and n would be much stiffer.

The above discussion should not be taken to mean that packet level provides a totally duplication-free PVC or VC. For, even at the packet level there is a mechanism to reset a PVC or VC; and user packets may be redundantly delivered to the transport layer at the other end. Further, when a VC is cleared there is no way for the transmitter of user data or interrupt packets to determine the state of receiver in respect of user messages received. It shall be the responsibility of transport layer to detect duplicate user messages that originate in the packet layer, just as the packet layer could detect duplication of packets by the link layer.

APPENDIX

PASCAL PROCEDURES FOR INTERPRETATION

```
procedure  Enable Level 3;
   begin if not Level 2 Enabled
        then Enable Level 2
        else Return Enable Level 2
   end

procedure  Return Enable Level 2;
   begin if Level 2 Enabled
        then Incase
                Data Tx In Progress : Return Enable Packet Interchange;
                Link Setup In Progress :;
                Link Disc In Progress : Return Enable Level 3;
                else : Enable Packet  Interchange
           end
        else Return Enable Level 3
   end

procedure  Return Enable Packet Interchange;
   var i : integer ; k : Buffer Address;
   begin if Data Tx In Progress
        then begin
                Initialize Q(Pkt In Q);
                Pkt Proc Is On := false;
                for i := 1 to 8
                  do Receive Packet;
                for x := 0 to Max Perm VC no
                  do Rx Call Connected (x);
                Initialize Available Channels;
                State of VCs := r3;
                Level 3 Enabled := true;
                Return Enable  Level 3
           end
        else Return Enable Level 3
   end

procedure Packet Received (k : Buffer Address);
   begin if Level 3 Enabled
        then begin
                Receive Packet:
                Add Last (k,Pkt In Q);
                while Length (Pkt In Q)>0
                  do begin k := Head of (Pkt In Q);
                           Behead (Pkt In Q);
                           Rx DCE Packet (k)
                      end
           end
   end

procedure  Packet Not Received;
   begin end;
```

```
procedure  Packet Transmitted (k : Buffer Address);
   begin end;

procedure  Packet Not Transmitted (k : Buffer Address);
   begin end;

procedure  Rx DCE Packet (k : Buffer Address);
   var t : Packet type;
   begin Pkt Proc  Is On := true;
        t := Compute  Packet type (k);
        case State of VCs of
         r1 : case t of
                Restart Indication,
                DCE Restart Confirmation : Rx DCE Restart Confirmation
              end;
         r3 : Incase
                t = Restart Indication : Rx Restart Indication;
                t = DCE Restart Confirmation : Restart VCs;
                t > DCE Restart Confirmation : Rx Pkt In r3
              end
         end;
         Return Pkt Buffer (k);
         Pkt Proc Is On := false
   end

procedure  Rx Pkt In r3 ; {t, k are global}
   var x : Channel no;
   begin x := Compute  Channel no (k);
        case State [x].p of
         p1 : Incase
                t = Incoming Call : Rx Incoming Call (x,k);
                t = Clear Indication : Tx DTE Clear Confirmation;
                else Clear VC(x)
              end;
         p2 : Incase
                t = Incoming Call : ;
                t = Call Connected : Rx Call Connected (x);
                t = Clear Indication : Rx Clear Indication (x);
                else : Clear VC (x)
              end;
         p4 : Incase
                t = Clear Indication : Rx Clear Indication (x);
                t = Incoming Call : Clear VC (x);
                t = Call Connected : Clear VC (x);
                t = DCE Clear Confirmation : Clear VC (x);
                else Rx Pkt In p4
              end;
         p6 : case t of
                Incoming Call : Clear VC (x);
                Clear Indication,
                DCE Clear Confirmation : Rx DCE Clear Confirmation (x);
                Call Connected : if Prev State [x] <> p2 then  Clear VC (x)
              end;
         end
    end
```

```
procedure  Rx Pkt In p4; {t, k, x are global}
   begin case State [x].d of
        d1 : Incase
                t = Reset Indication : Rx Reset Indication (x);
                t = DCE Reset Confirmation : Reset VC (x);
                t > DCE Reset Confirmation : Rx Pkt In d1
             end;
        d2 : case t of
                Reset Indication,
                DCE Reset Confirmation : Rx DCE Reset Confirmation (x)
             end
        end
   end

procedure  Rx Pkt In d1; {t, k, x are global}
   begin case t of
              DCE Data                    : Rx DCE Data (x,k);
              DCE RR                      : Rx DCE RR (x,k);
              DCE RNR                     : Rx DCE RNR (x,k);
              DCE Interrupt               : Rx DCE Interrupt (x,k);
              DCE Interrupt Confirmation : Rx DCE Interrupt Confirmation (x)
        end
   end

procedure  Restart VCs; {Level 3 Enabled and (State of VCs = r3)}
   begin State of VCs := r1;
         Tx Restart Request (00000000)
   end

procedure  Rx DCE Restart Confirmation;
   begin Initialize Channels upon Restart;
         State of VCs := r3;
         VCs Restarted
   end

procedure  Rx Restart Indication;
   begin Initialize Channels upon Restart;
         State of VCs := r3;
         Tx DTE Restart Confirmation;
         VCs Restarted
   end

procedure  Initialize Channels upon Restart;
   begin for x := 0 to Max Perm VC no
           do Rx DCE Reset Confirmation (x);
         for x := Max Perm VC no + 1 to Max VC no
           do Rx DCE Clear Confirmation (x)
   end
```

```
procedure  Establish VC (var x : Channel no;
                         i,j : DTE Address);
           {Level 3 Enabled and (State of VCs = r3)}
   begin Get Channel (x);
         State [x].p := p2;
         Tx Call Request (x,i,j)
   end


procedure  Rx Call Connected (x : Channel no);
   var   i,j : DTE Address;
   begin State [x].p := p4;
         Initialize Vars upon Connect (x);
         i := Local DTE (x);
         j := Remote DTE (x);
         VC established (x,i,j)
   end


procedure  Rx Incoming Call (x : Channel no ; k : Buffer Address);
   var   i,j : DTE Address;
   begin i := Compute  Calling DTE Address (k);
         j := Compute  Called DTE Address (k);
         State [x].p := p4;
         Initialize Vars upon Connect (x);
         Tx Call Accepted (x);
         VC Established (x,j,i)
   end


procedure  Initialize Vars upon Connect (x : Channel no);
   begin State [x].d := dl;
         No of Messages to Receive [x] := 0;
         Initialize Q(Q Un Ack Pkt [x]);
         Initialize Q(Q Un Tx Pkt [x]);
         Interrupt In Buffer Full [x] := false;
         Interrupt Out Buffer Full [x] := false;
         Initialize Vars upon Reset (x)
   end


procedure Initialize Vars upon Reset (x : Channel no)
   begin DTE Pkt Level Busy [x] := false;
         DCE Pkt Level Busy [x] := false;
         Tx State Var [x] := 0;
         Last Recd PR [x] := 0;
         Rx State Var [x] := 0;
         Rx PR Set [x] := [0..W-1]
   end


procedure  Clear VC (x : Channel no);
           {Level 3 Initialized and (State of VCs = r3)}
   begin case State [x].p of
           p1 : VC Cleared (x);
           p2,p4: begin Prev State [x] := State [x].p;
                        State [x].p := p6;
                        Tx Clear Request (x,00000000)
                  end
         end
```

```
procedure  Rx DCE Clear Confirmation (x : Channel no);
   begin Return Un Tx Data and Interrupt Messages (x);
         VC Cleared (x);
         State [x].p := p1;
         Return Channel (x)
   end


procedure Rx Clear Indication (x : Channel no);
   begin Return Un Tx Data and Interrupt Messages (x);
         Tx DTE Clear Confirmation (x);
         VC Cleared (x);
         State [x].p := p1;
         Return Channel (x)
   end


procedure  Return Un Tx Data and Interrupt Messages
           (x : Channel no);
   var   k : Buffer Address ; Q,M:bit ; User Data : Data String;
   begin Merge(Q Un Ack Pkt [x], Q Un Tx Pkt [x])
         while not Is Empty (Q Un Tx Pkt [x])
           do begin k := Tail of (Q Un Tx Pkt [x])
                     Q := Compute  Q(k);
                     M := Compute  M(k);
                     User Data := Compute  User Data (k);
                     Data Message Not Transmitted (x,Q,M,User Data);
                     Betail (Q Un Tx Pkt [x])
              end;
         if Interrupt Out Buffer Full [x]
           then Interrupt Message Not Transmitted (x,Interrupt Out Buffer [x])
   end


procedure  Reset VC (x : Channel no);
           {Level 3 Initialized and (State of VCs = r3)
                                    and (State [x].p = p4)}
   begin if State [x].d = d1
         then begin State [x].d := d2;
                    Tx Reset Request (x,00000000, 00000000)
         end
   end


procedure  Rx DCE Reset Confirmation (x : Channel no);
   begin State [x].d := d1;
         Initialize Vars upon Reset (x);
         VC Reset (x)
   end
```

```
procedure  Rx Reset Indication (x : Channel no);
   begin State [x].d := dl;
         Initialize Vars upon Reset (x);
         Tx DTE Reset Confirmation (x);
         VC Reset (x)
   end


procedure  VC Reset (x : Channel no);
   begin if Interrupt Out Buffer Full [x]
              then Tx DTE Interrupt (x);
            Resume DTE Data Tx (x)
   end


procedure  Transmit Interrupt Message (x : Channel no;
                                       Interrupt Data : byte);
            {Level 3 Initialized and (State of VCs = r3) and
             State [x].p = p4) and not Interrupt Out Buffer Full [x]}
   begin Interrupt Out Buffer Full [x] := true;
         Interrupt Out Buffer [x] := Interrupt Data;
         if State [x].d = dl
           then Tx DTE Interrupt (x)
   end


procedure Rx DCE Interrupt Confirmation (x : Channel no);
   begin if Interrupt Out Buffer Full [x]
               then begin Interrupt Out Buffer Full [x] := false;
                          Interrupt Message Transmitted (x)
                    end
               else Reset VC(x)
   end


procedure  Rx DCE Interrupt (x : Channel no ; k : Buffer Address);
   var   Interrupt Data : byte;
   begin if not Interrupt In Buffer Full [x]
               then begin Interrupt In Buffer Full [x] := true;
                          Interrupt Data := Compute Interrupt Data (k)
                          Interrupt Message Received (x, Interrupt Data)
                    end
               else Reset VC(x)
   end


procedure  Interrupt Message Accepted (x : Channel no);
           {Level 3 Enabled and (State of VCs = r3) and
           (State [x].p = p4 and Interrupt In Buffer Full [x]}
   begin Interrupt In Buffer [x] := false;
         if State [x].d = dl
               then Tx DTE Interrupt Confirmation (x)
   end
```

```
procedure  Transmit Data Message (x : Channel no; Q,M : bit;
                                  User Data : Data String);
           {Level 3 Initialized and (State of VCs = r3) and
           (State [x].p = p4)}
   var   k : Buffer Address ;
   begin Get Pkt Buffer (k);
         Write Q(k,Q)
         Write M(k,M)
         Write User Data (k, User Data)
         Add Last (k, Q Un Tx Pkt [x])
         Resume DTE Data Tx (x)
   end


procedure  Resume DTE Data Tx (x : Channel no);
   var   k : Buffer Address;
   begin if (State [x].d = d1) and not DCE Pkt Level Busy [x]
           then while (Tx State Var [x] < Last Recd PR[x] + W)
                       and (Length (Q Un Tx Pkt [x])>0)
                do begin k := Head of (Q Un Tx Pkt [x]);
                          Tx DTE Data (k,x)
                          Tx State Var [x] := Tx State Var [x] + 1;
                          Behead (Q Un Tx Pkt [x])
                          Add Last (k, Q Un Ack Pkt [x])
                   end
   end


procedure  Rx DCE RR (x : Channel no; k : Buffer Address);
   var   PR : 0..7; PR Check :(Valid, Invalid);
   begin DCE Pkt Level Busy [x] := false;
         PR := Compute  PR (k);
         Process PR (x, PR, PR Check);
         if PR Check = Valid
           then Resume DTE Data Tx (x)
   end

procedure  Rx DCE RNR (x : Channel no: k : Buffer Address);
   var   PR : 0..7; PR Check : (Valid, Invalid);
   begin DCE Pkt Level Busy [x] := true;
         PR := Compute  PR(k);
         Process PR (x, PR, PR Check);
         if PR Check = Valid
           then begin Merge (Q Un Ack Pkt, Q Un Tx Pkt);
                      Tx State Var [x] := Last Recd PR[x]
                end
   end
```

```
procedure  Process PR (x : Channel no; PR : 0..7;
                       var PR Check : (Valid, Invalid);
   var   k : Buffer Address
   begin PR Check := Compute  PR Check (x,PR);
         if PR Check = Invalid
           then Reset VC (x)
           else begin for i := Last Recd PR [x] to PR - 1
                      do begin k := Head of (Q Un Ack Pkt [x])
                                Q := Compute Q(k)
                                M := Compute M(k)
                                User Data := Compute User Data (k)
                                Data Message Transmitted (x, Q, M, User Data)
                                Behead (Q Un Ack Pkt [x])
                                Return Pkt Buffer (k)
                           end;
                      Last Recd PR[x] := PR
                end
   end

function  Compute PR Check (x := Channel no; PR : 0..7) : (Valid, Invalid);
   begin if (PR ≥ Last Recd PR [x]) and (PR ≤ Tx State Var [x])
           then Compute PR Check := Valid
           else Compute PR Check := Invalid
   end

procedure  Receive Data Message (x : Channel no);
           {Level 3 Initialized and(State of VCs = r3)
           and (State [x].p = p4)}
   begin No of Messages to Receive [x] := No of Messages to Receive [x] + 1;
         if DTE Pkt Level Busy [x] and not State [x].d = d1
           then DTE Pkt Level Becomes Not Busy (x)
   end

procedure  Rx DCE Data (x : Channel no; k : Buffer Address);
   var   i, PS, PR : 0..7; Q, M : bit; User Data : Data String;
         PS Check : (Fresh, Duplicate, Invalid);
         PR Check : (Valid, Invalid);
   begin PR := Compute PR (k);
         Process PR (x, PR, PR Check);
         if PR Check = Valid
           then begin Resume DTE Data Tx (x);
                      PS := Compute PS (k);
                      PS Check := Compute PS Check (x,PS);
                      if PS Check = Invalid
                         then Reset VC (x);
                      if (PR Check = Valid) and (PS Check = Fresh)
                        and not DTE Pkt Level Busy [x]
                        then begin Rx PS Set [x] := Rx PS Set [x] - [PS];
                                   if PS = Rx State Var [x]
                                      then begin i := Rx State Var [x];
                                                 while not i in Rx PS Set [x]
                                                   do i := i + 1;
                                                 Rx State Var [x] := i;
                                                 Tx Acknowledge (x)
                                           end;
                                   Q := Compute Q (k);
```

```
                                M := Compute M (k);
                                User Data := Compute User Data (k);
                                Data Message Received (x, Q, M, User Data);
                                No of Messages to Receive [x] :=
                                        No of Messages to Receive [x] - 1
                                if No of Messages to Receive [x] = 0
                                   then DTE Pkt Level Becomes Busy (x)
                            end
            end
   end

function  Computer PS Check (x : Channel no; PS : 0..7) :
                                (Fresh, Duplicate, Invalid);
   begin if (PS > Rx State Var [x]) and (PS < Rx State Var [x] + W)
            then if PS in Rx PS Set [x]
                        then Compute PS Check := Fresh
                        else Compute PS Check := Duplicate
            else Compute PS Check := Invalid
   end


procedure Tx Acknowledge (x : Channel no);
   begin if not (Tx State Var [x] < Last Recd PR [x] + W)
               or not (Length(Q Un Tx Pkt [x]) > 0)
            then Tx DTE RR (x)
   end


procedure  DTE Pkt Level Becomes Busy (x : Channel no);
   begin DTE Pkt Level Busy [x] := true;
         Tx DTE RNR (x)
   end


procedure  DTE Pkt Level Becomes Not Busy (x : Channel no);
   begin DTE Pkt Level Busy [x] := false;
         Tx DTE RR (x)
   end


procedure  Tx Call Request (x : Channel no; i,j : DTE Address);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00001011);
         Write Calling DTE Address (k, i);
         Write Called DTE Address (k, j);
         Transmit Packet (k)
   end


procedure  Tx Call Accepted (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00001111);
         Transmit Packet (k)
```

```
procedure  Tx Clear Request (x : Channel no; y : byte);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00010011);
         Write Clearing Cause (k, y);
         Transmit Packet (k)
   end


procedure  Tx DTE Clear Confirmation (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00010111);
         Transmit Packet (k)
   end


procedure  Tx DTE Data (k : Buffer Address; x : Channel no);
   begin Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00000000);
         Write PS (k, Tx State Var [x]);
         Write PR (k, Rx State Var [x]);
         Transmit Packet (k)
   end


procedure  Tx DTE Interrupt (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Packet type (k, 00100011);
         Write Interrupt Data (k, Interrupt Out Buffer [x]);
         Transmit Packet (k)
   end


procedure  Tx DTE Interrupt Confirmation (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00100111);
         Transmit Packet (k)
   end
```

```
procedure  Tx DTERR (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00000001);
         Write PR (k, Rx State Var [x]);
         Transmit Packet (k)
   end


procedure  Tx DTE RNR (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00000101);
         Write PR (k, Rx State Var [x]);
         Transmit Packet (k)
   end


procedure  Tx Reset Request (x : Channel no; r, d : byte);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00011011);
         Write Resetting Cause (k, r);
         Write Diagnostic Code (k, d);
         Transmit Packet (k)
   end


procedure  Tx DTE Reset Confirmation (x : Channel no);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Channel no (k, x);
         Write Packet type (k, 00011111)
         Transmit Packet (k)
   end


procedure  Tx Restart Request (r : byte);
   var   k : Buffer Address;
   begin Get Pkt Buffer (k):
         Write GFI (k);
         Write Packet type (k, 11111011);
         Write Restarting Code (k, r);
         Transmit Packet (k)
   end
```

```
procedure  Tx DTE Restart Confirmation;
   var   k : Buffer Address;
   begin Get Pkt Buffer (k);
         Write GFI (k);
         Write Packet type (k, 11111111);
         Transmit Packet (k)
   end
```

REFERENCES

1. CCITT, "Public Data Networks", Recommendations of Sixth Plenary Assembly, Orange Book, vol. VIII.2, Internat. Telecom. Union, Geneva, 1977, pp. 70-108.

2. A. Rybczynski, "X.25 Interface and End-to-End Virtual Circuit Service Characteristics", IEEE Trans. Comm., Vol. Com-28, no. 4, 1980, pp. 500-510.

3. H. C. Folts, "X.25 Transaction-Oriented Features-Datagram and Fast Select", IEEE Trans. Comm., Vol. Com-28, no. 4, 1980, pp. 496-500.

4. B. N. Jain, "X.25 Level 2 Communication Protocols: An interpretation and Proposed Implementation", Tech. Rep. No. 8109, Dept. of Electr. Engg., Indian Institute of Technology, Delhi, June 1981.