THE USE OF ISDN SIGNALING FOR REAL-TIME APPLICATIONS AT HOMES AND SMALL BUSINESSES

By

NISHEETH PRAKASH

B.Tech (Computer Science & Engineering), Govind Ballabh Pant University – Pantnagar, UttarPradesh, India - 1993

Thesis

Submitted in partial fulfillment of the requirements for the Degree of Master of Science (Computer Science)

Acadia University
Fall Convocation 1998

Table of Contents

List of Figures

ABSTRACT

Integrated Services Digital Networks (ISDN) is an end-to-end digital telecommunications network providing the capability to transmit voice, data, facsimile, telemetry, signaling, and slow motion video. In this thesis, the use of the narrowband ISDN has been explored for the real-time applications at homes and small businesses. We have classified some of the real-time applications at homes and small businesses into different categories and have investigated LAP-D protocol (Q.921) and Call Control Protocol (Q.931) to support the real-time applications. The user-to-user signaling connection over the D-Channel has been explored to develop a framework for these real-time applications. High level scenarios and the protocol message and bit level scenarios have been designed for these real-time applications at homes and small businesses. A testing model has been developed on the SUN SPARC station in Solaris environment and few of the applications have been successfully tested using our model.

List of Definitions of Abbreviations

ISDN	Integrated Services Digital Networks
B-Channel	Bearer Channel
D-Channel	Delta Channel
PCM	Pulse Code Modulation
Kbps	Kilo bits per seconds
PBX	Private Branch Exchange
LAN	Local Area Network
LAP-D	Link Access Protocol for Delta Channel
LAP-B	Link Access Protocol for Bearer Channel
NT1	Network Termination 1
NT2	Network Termination 2
TE1	Terminal Equipment type 1
TE2	Terminal Equipment type 2
TA	Terminal Adapter
DLCI	Data Link Connection Identifier
HDLC	High Level Data Link Control
TEI	Terminal Endpoint Identifier
SAPI	Service Access Point Identifier
C/R bit	Control/Response bit
P/F bit	Poll/Final bit
SABME	Set Asynchronous Balanced Mode Extended
DISC	Disconnect

RR	Receive Ready
RNR	Receive Not Ready
FRMR	Frame Reject
FCS	Frame Check Sequence
UA	Unnumbered Acknowledgement
PRI	Primary Rate Interface
BRI	Basic Rate Interface
DME	Data Monitoring End
DGE	Data Generating End
TCP/IP	Transport Control Protocol / Internet Protocol
DLPI	Data Link Provider Interface
REJ	Reject
DLS	Data Link Service

Acknowledgement

I would like to thank Dr. Ali Elkateeb, my thesis supervisor, for his dedication, patience and the valuable time he contributed towards this thesis. He was a constant source of inspiration and encouragement at times when I came across several problems and difficulties in the research.

I would like to thank the Jodrey School of Computer Science and the Research and Graduate Studies Department that considered me eligible for the Acadia Graduate Fellowship and the Teaching Assistantship during my stay at Acadia as a Graduate student.

I would like to thank TARA (Telecom Applications and Research Alliance - Halifax) for awarding me the TARA scholarship for my research work and at times providing with the useful pointers in the industry to clarify certain doubts about the industry standards being followed in ISDN. I would also like to thank Mr. Bill McMullin of Info-Interactive – Halifax, for being an industry affiliate in this research.

I would like to thank Dr. Andre Trudel for lending me his Sparc work station to do testing in the research work and also helping us out in his capacity as a Director of the School and the Graduate Coordinator.

Finally I would like to thank my parents and dedicate this thesis to them, who encouraged me to go for the Masters degree. Without their support this thesis would not be possible.

Chapter 1

Introduction

The Integrated Services Digital Networks (ISDN) is a public end-to-end digital telecommunications network providing the capability to transmit voice, data, facsimile, telemetry, signaling, and slow motion video. The main feature of the ISDN concept is the support of a wide range of voice and non-voice applications in the same network. ISDN provides a range of services using a limited set of connection types and multipurpose user-network interface arrangements. ISDN supports a variety of applications, including both the switched and non-switched connections. The switched connections in an ISDN include both the circuit switched and packet switched connections.

The number of applications that can be supported by telecommunication networks is endless. Some of the many applications that are supported by ISDN and mentioned in [BellISDNapp96],[Kess93] and [Nitz90] are as follows:

1) Enhanced Phone Services - ISDN number is issued for life. Wherever you move in the world, the number moves with you. Automatic callback, selective call forwarding and selective call blocking are other enhanced services that ISDN provides.
2) High Speed Data Transfer - ISDN can provide a data transfer rate equivalent to that of T1 (1.544 Mbps).
3) TeleConferencing - Teleconferencing is a method by which images of individuals, at different locations in the world can simultaneously transmit voice and data to each other. The bandwidth provided by ISDN can support teleconferencing.

4) TeleFinancing - Services such as telebanking, teleaccounting will be provided at homes through ISDN.
5) TeleMedicine - Doctors will provide on-line outpatient monitoring at homes and surgeons will transmit 3-D pictures to others while they are performing surgery.
6) TeleCommuting - TeleCommuting will allow individuals to work at home, without going to the office.
7) TeleControl - is the ability to automatically control remote device at different locations.
8) TelePolling - Remote Surveillance of security cameras located in different buildings from a central control room is one example of TelePolling.
9) TeleMonitoring - Monitoring of utility meters, burglar alarms or fire detectors from a remote location is also possible through ISDN.

This thesis addresses a few of the above mentioned applications like TeleControl, TelePolling and TeleMonitoring that can be supported through ISDN. Keeping in view the bandwidth requirements of the above applications, the ISDN call control protocol and LAP-D protocol have been analyzed in detail to support these applications. The high level scenarios have been designed for these applications. The protocol message and bit level scenarios have also been designed. An implementation model has been developed to show how the ISDN protocols can be used to support the above applications. The model has been developed on Sun Sparc work station in Solaris environment. A few applications, such as TeleMonitoring and TeleControl have been tested using the model.

The organization of this thesis is as follows. Chapter 2 gives an overview of ISDN in general, ISDN interfaces, LAP-D protocol specified in [ITUQ921] (ISDN User-Network Interface Layer-Data Link Layer Specification) and Q.931 call control protocol specified in [ITUQ931] (ISDN User-Network Interface Layer 3 Specification for Basic Call Control). Chapter 3 consists of the high level scenarios for the applications which have been considered for detailed analysis. Chapter 4 lists the message and bit level scenarios at the protocols level (LAP-D & Call Control). Chapter 5 describes in detail the design and implementation of the protocol model which has been chosen for the applications. Chapter 6 describes the testing model that is designed and implemented to support the above applications, using the protocol model described in Chapter 5. Chapter 7 is the conclusion and the future works. Finally the Appendix contains the source code for the applications.

CHAPTER 2

Overview of ISDN, its Interfaces and Protocols

2.1 Introduction to ISDN

ISDN had been proposed in early 80's to support a large variety of services that the existing network did not support. ISDN refers to simultaneously carrying of digitized voice and a variety of data traffic on the digital transmission links and the digital exchanges. The following sections describe the main features of ISDN such as the basic principles of ISDN, ISDN channels and transmission structures as specified in [Stall95], ISDN protocol architecture, and the state machines of the protocols. The features of the protocols that have been explored to support the real-time applications at homes and small businesses are also described in detail.

2.1.1 The Basic Principles of ISDN

The basic principles of ISDN can be summarized as follows:

a) Support of voice and non-voice applications - ISDN supports a variety of services related to voice communications (telephone calls) and non-voice communications (digital data exchange).

b) Support of switched and non-switched applications - ISDN supports both circuit switching and packet switching.

c) Reliance on 64-kbps connections - The basic rate with which ISDN supports the circuit and packet switched connections is 64Kbps, as it is a standardized rate for digitized voice.

d) Layered protocol architecture – The protocol model developed for user access to ISDN can be mapped directly to the OSI model. At the data link layer level, ISDN supports the LAP-D (Link Access Protocol for Delta Channel which carries signaling information) and LAP-B (Link Access Protocol for Bearer Channel which carries digitized voice or digital data). At the layer 3 level, X.25 can be used for packet switching services that ISDN provides, and for call establishment and release Q.931 call control protocol is used at the layer 3.

e) Variety of configurations - Several physical configurations are possible for implementing ISDN, as shown in Figure 2.1.

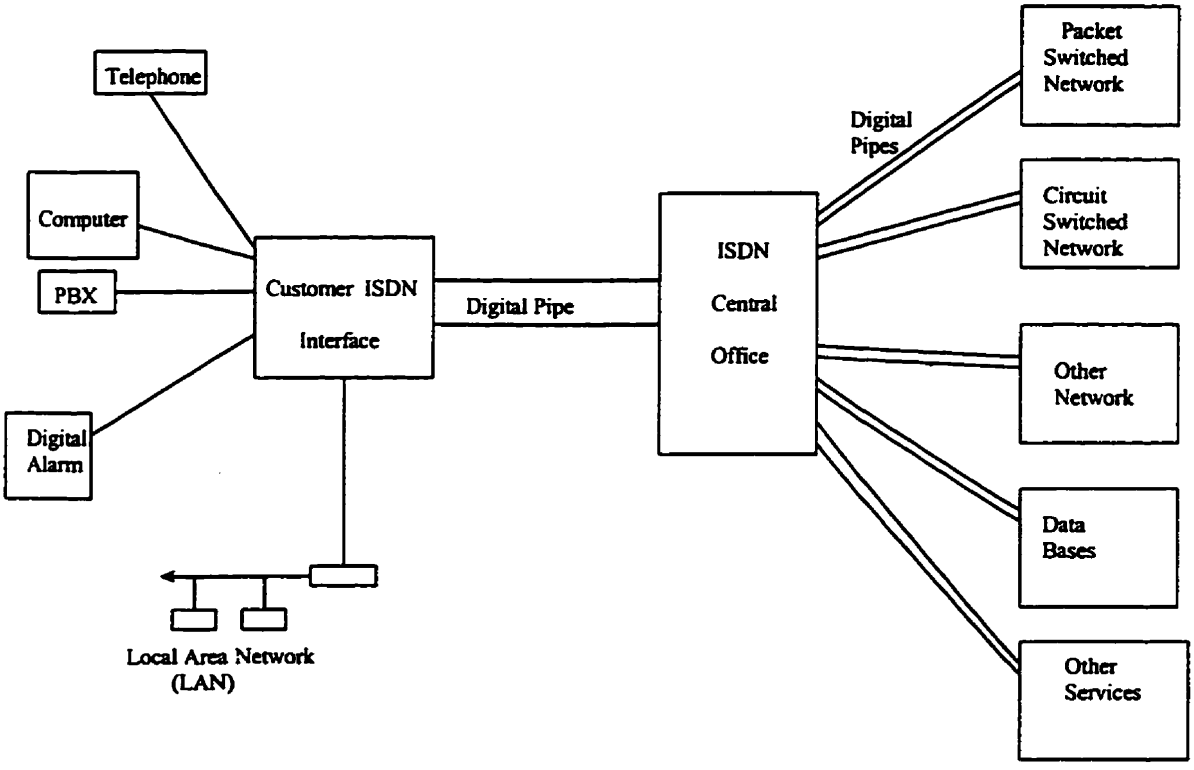


Figure 2.1: Conceptual View of ISDN Connection Features

2.1.2 ISDN Channels

The digital pipe between the central office and ISDN subscriber is used to carry a number of communication channels, namely the B-Channels and the D-channel.

a) **B-Channel** - (also called bearer channel) has a capacity of 64Kbps and is used to carry digital data and PCM encoded digital voice. Three kinds of connections are supported: circuit switched, packet switched and semi-permanent (leased line).

b) **D-Channel** - (also called delta channel) has a capacity of 16Kbps (Kilo bits per seconds) and is used to carry signaling information to control circuit switched calls on the associated B Channels. The channel can also be used for packet switching.

2.1.3 ISDN Transmission Structures

The above channel types are grouped into two different transmission structures, which are offered to the user as a package:

a) **Basic Rate Access** - It provides two full duplex B Channels and one full duplex D-Channel (2B+D = 144Kbps). Framing and synchronization overhead bits, that are added t o each frame makes it 192Kbps. Basic access is suited to the most individual users, residential subscribers and small offices and allows simultaneous use of several voice and data services like fax, telephone and Internet.

b) **Primary Rate Access** - In US, Canada and Japan, it provides 23 full duplex B-Channels and one 64Kbps D-Channel (1.544Mbps). In Europe it provides 30 full duplex B Channels and one 64Kbps D-Channel (2.048Mbps). It is intended for users with higher capacity requirements such as offices with digital PBXs or a LAN. In primary rate access the D-Channel has a higher capacity (64 Kbps) than the basic rate access (16Kbps) as it carries signaling information of 23 or 30 B-Channels.

2.2 ISDN Interfaces, Functional Devices and Reference Points [Stall95]

2.2.1 ISDN Functional Devices

The various ISDN functional devices as shown in figure 2.2 are as follows:

Network Termination 1 (NT1) is the termination of the physical connection between the user site and the local exchange. NT1 performs line performance monitoring and timing, used for power transfer and multiplexes the B-Channel and the D-Channel.

Network Termination 2 (NT2) is an intelligent device that may include up through OSI layer 3 functionality. NT2 can perform switching, multiplexing and concentration functions. Example of NT2 is a digital PBX, LAN etc.

Terminal Equipment type 1 (TE1) refers to the devices that support the standard ISDN interface like digital telephone, standard voice/data terminals etc.

Terminal Equipment type 2 (TE2) refers to existing non-ISDN equipment. Examples are equipment with a physical interface, such as RS-232C, and host computers with an X.25 interface. Such equipment needs a Terminal Adapter (TA) which converts the non-ISDN protocols into ISDN protocols.

2.2.2 ISDN Reference Points

R, S and T (Figure 2.2) are the reference points used to separate the group of functions. The R reference point separates the non-ISDN device from Terminal Adapter (TA). The S reference point separates Terminal Equipment (TE1) or Terminal Adapter (TA) and network termination equipment (NT1 or NT2). The T reference point separates customer site switching equipment (NT2) and the local termination (NT1).

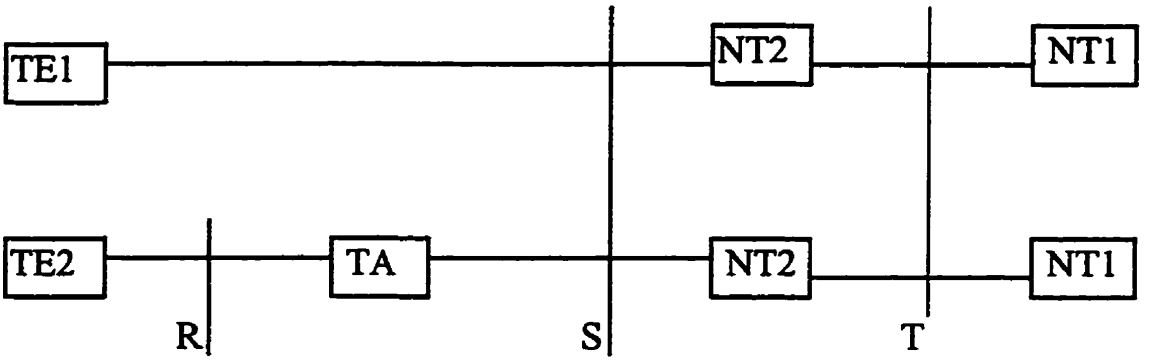


Figure 2.2: ISDN Reference points and Functional Groupings

2.3 ISDN Protocols

2.3.1 The Layer 2 - Q.921 / LAP-D Protocol –

The Layer 2, also called Link Access Protocol for delta channel is specified in [ITUQ921] (LAP-D or Q.921 as shown in Figure 2.3). It describes the high level data link procedures.

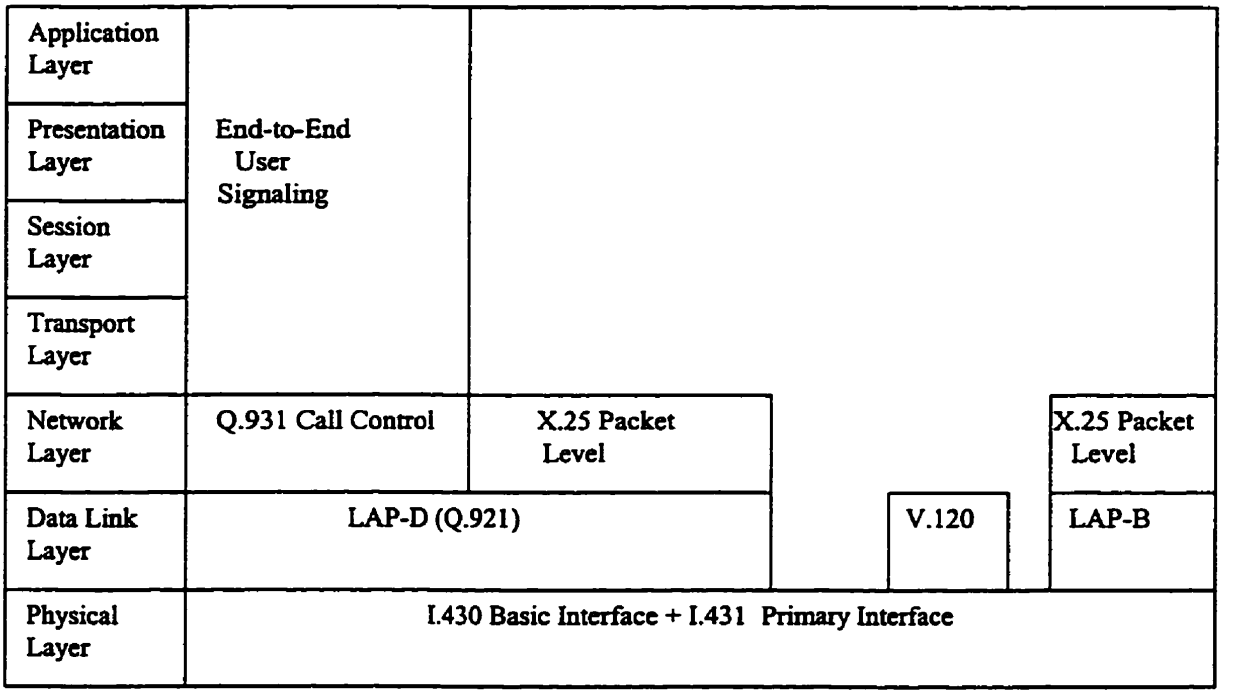


Figure 2.3: ISDN Protocols at User-Network Interface

2.3.1.1 Layer 2 Functions

The major functions of the layer 2 also called as LAP-D (Q.921) as specified in [ITUQ921] are as follows:

a) the provision of one or more data link connections on a D-channel. Discrimination between the data link connections is by means of a data link connection identifier (DLCI) contained in each frame

b) frame delimiting, alignment and transparency, allowing recognition of a sequence of bits transmitted over a D-channel as a frame

c) sequence control, to maintain the sequential order of frames across a data link connection

d) detection of transmission, format and operational errors on a data link connection

e) recovery from detected transmission, format and operations errors

f) notification to the management entity of unrecoverable errors

g) flow control

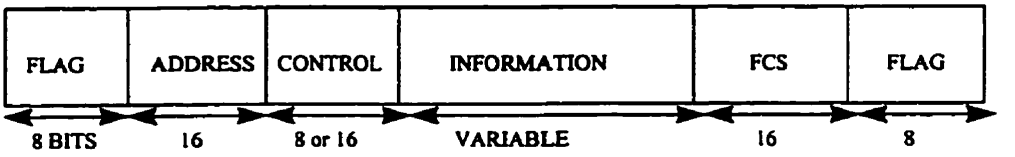
h) Layer 3 Call Control information is carried in the information field of the LAP-D frame and is delivered to the peer Layer 3 entity

2.3.1.2 LAP-D Frame Format and Frame Types

The LAP-D protocol is modeled after the LAP-B protocol used in X.25 and on HDLC (High Level Data Link Control). Both user information and parameters are transmitted in the form of frames. The various fields of the LAP-D frame are as follows:

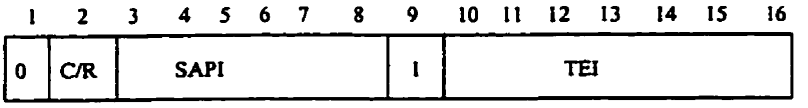
Flag Field - The flag field delimits the frame at both ends with the unique pattern 011,111,10. In order to avoid the occurrence of the similar pattern inside the frame, the

technique known as bit stuffing is used. Between the transmission of the starting and the ending flags, the transmitter will always insert an extra 0 bit after each occurrence of five consecutive 1's in the signaling unit.



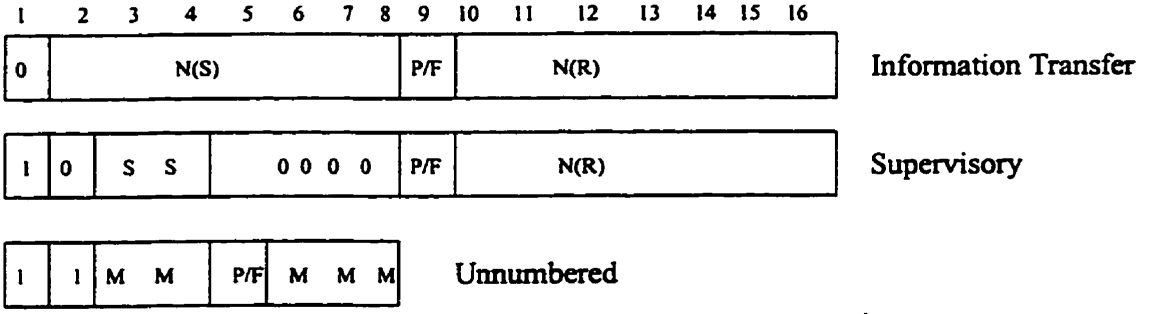
a) LAP-D FRAME FORMAT

The FLAG FIELD delimits the frame at both ends with the unique pattern 01111110



b) ADDRESS FIELD FORMAT

C/R = Command/Response
SAPI = Service Access Point Identifier
TEI = Terminal End Point Identifier



c) CONTROL FIELD FORMAT

N(S): Transmitter send sequence number
N(R): Transmitter receive sequence number
S : Supervisory function bit
M : Modifier function bit
P/F :Poll/Final bit

Figure 2.4: LAP-D Frame Format

Address Field - A LAP-D frame has a two part address field, consisting of a terminal end point identifier (TEI) and a service access point identifier (SAPI). Since multiple user devices share the same physical interface at a subscriber site, each device is given a

unique TEI. The service access point identifier (SAPI) identifies a layer 3 user of LAPD and thus corresponds to a layer 3 protocol entity within the user device. SAPI and TEI together are called DLCI (Data Link Connection Identifier).

Typical SAPI values are:

0 - Call Control procedures for managing B-Channel circuits

16 - packet mode communication on the D-Channel using X.25 as layer 3

63 - used for exchange of layer 2 management information

1 - packet mode communication using Q.931.

The address field also includes the command/response (C/R) bit. All LAP-D messages are categorized as either commands or responses, and this bit is used to indicate which type of message is contained in the frame.

Control Field - LAP-D defines three types of frames.

a) Information Frames (I- frames) carry the data to be transmitted for the user

b) Supervisory Frames (3 different S - frames :- RR receive ready , RNR receive not ready, REJ reject) provide the Automatic Repeat Request flow control mechanism

c) Unnumbered Frames (SABME - request logical connection, DM Disconnected Mode - unable to establish or maintain connection, DISC Disconnect - terminate logical connection, UA Unnumbered Acknowledgment - Acknowledge SABME or DISC, FRMR frame reject - reports receipt of unacceptable frame)

The Control field identifies each of these different types of frames. The control field format contains the poll/final (P/F). In Command frames it is referred to as P bit and is set to 1 to solicit a response from the peer LAP-D entity. In response frames, it is

referred to as F bit and is set to 1 to indicate the response frame transmitted as a result of a soliciting command.

Information Field - This field is present only in I-frames and some unnumbered frames. The field can contain any sequence of bits and must consist of an integral number of octets (group of eight bits). The maximum number of octets is 260.

Frame-Check Sequence Field - It is an error detecting code calculated from the remaining bits of the frame, exclusive of flags.

2.3.1.3 LAP-D Message Flow and Operation

a) **Connection Establishment** - A request for service from the customer results in layer 3 requesting a service from Q.921 by sending a DL_ESTABLISH primitive. In its response the LAP-D entity sends SABME (containing the SAPI & TEI of the Q.931 entity to which the connection is requested) to the peer LAP-D entity, and starts a timer T200 to wait for an acknowledgment. The peer LAP-D entity passes up this connection request to the appropriate layer 3 entity. The layer 3 entity responds with DL-CONNECT_CON to indicate its acceptance of the connection. The LAP-D entity on receiving the connection acceptance, will respond with Unnumbered Acknowledgment (UA) to the LAP-D entity which requested the connection. If the LAP-D entity receives the UA before the timer T200 expires, it enters a state where they can exchange Information Frames. This state is called Multiple Frame Establishment state. If the timer T200 expires before receiving UA, then the LAP-D entity retries to establish connection by sending SABME again. Only three retrials are allowed (N200 parameter).

b) **Data Transfer** - Once the two communicating LAP-D entities enter a multiple frame establishment state, the corresponding Layer 3 entities can begin sending the user data in the LAP-D Information frames starting with sequence number 0. N(S) and N(R) fields of the I-frame are sequence numbers that support flow control and error control. Receive Ready (RR) supervisory frame is used to acknowledge the last I-frames received and Receive Not Ready (RNR) supervisory frame is used to ask the peer entity to suspend transmission of I-frames. Reject (REJ) supervisory frame indicates that the last I-frame received has been rejected and that the retransmission of all I-frames beginning with number N(R) is required.

c) **Disconnect** - In case of fault or on a request from the Layer 3 user , the LAP-D entity can initiate the connection disconnect by sending a DISC frame to the peer LAP-D entity. The remote LAP-D entity responds with an Unnumbered Acknowledgment and also informs its layer 3 user that the connection has been terminated.

d) **Frame Reject (FRMR)** - Frame Reject is used to indicate that an improper frame has arrived (ex - a frame with undefined control field, S-Frame or U-Frame of incorrect size). The effect of the FRMR is to abort the connection and start re-establishment of the data link.

2.3.1.4 LAP-D (Q.921) State Transition:

Figure 2.5 shows the states that the Q.921 LAP-D protocol can be in. Initially the state machine is in TEI unassigned state and the LAP-D entity sends a physical interface (ISDN card) activation request. On receiving the physical activation acknowledgment, the LAP-D entity sends the TEI assignment request to the network. On getting the TEI from the network the LAP-D entity enters the Multiple frame establishment state and

stays in that state as long as layer 3 peer-to-peer communication requires a reliable communication channel. After that LAP-D entity moves to the TEI assigned state.

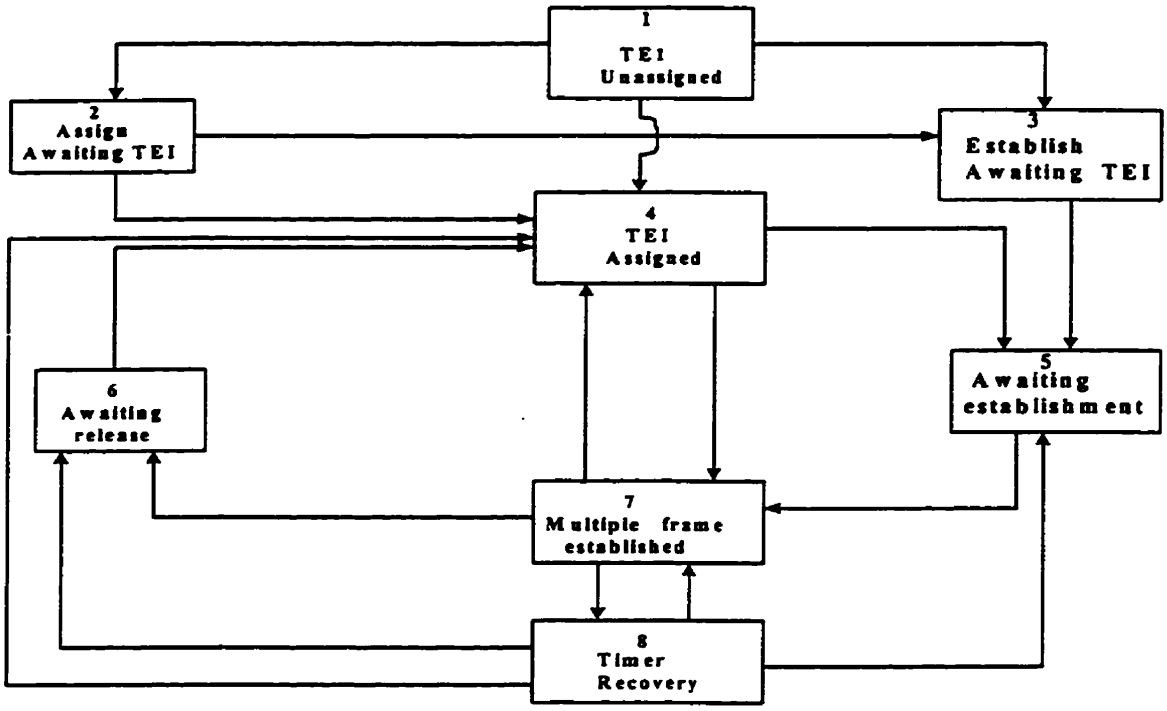


Figure 2.5: LAP-D State Transition Diagram

2.3.2 ISDN Network Layer Protocol / Call Control Protocol / Q.931

The ISDN network layer protocol (Q.931) is a D-Channel protocol used to establish, maintain, and terminate network connections on the B channel.

2.3.2.1 Layer 3 Functions

The basic set of functions to be performed at the network layer, for call control are:

a) Interacting with the data link layer (LAPD) to transmit and receive messages.

b) Generation and interpretation of layer 3 messages.

c) Administration of timers and logical entities used in call control messages.

d) Administration of the access resources, including B-Channels and packet layer logical channels used in X.25.

e) Includes mechanisms for providing network connections making use of data link connections.

f) Provides mechanisms to convey user to network and network to user information with or without the establishment of circuit switched connection on the B-Channel.

2.3.2.2 Layer 3 Message Format

All the layer 3 (Q.931) messages are built up by a certain message body. The layer 3 message format is described in the Figure 2.6.

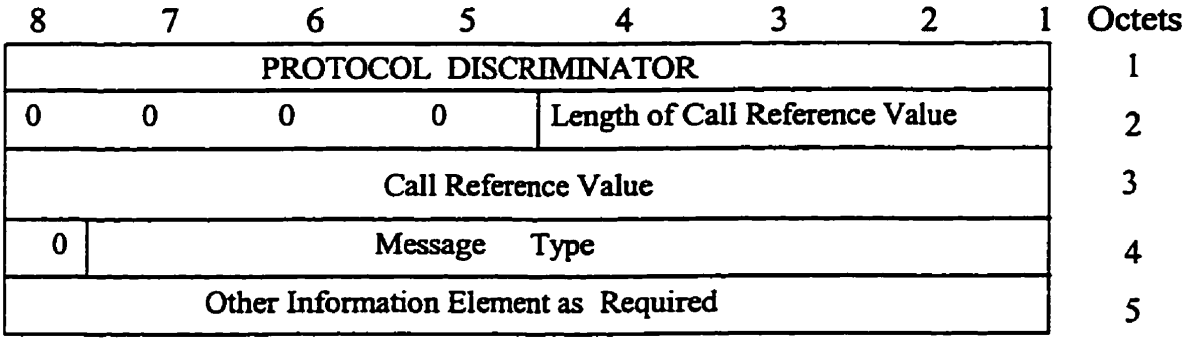


Figure 2.6: Layer 3 Message Format

The various fields of the layer 3 message format are described as follows:

Protocol Discriminator - It is used to distinguish messages for the user-network call control messages from other messages. Its value for Q.931 call control messages is 0001000.

Call Reference - The purpose of call reference is to identify the call or facilitate the registration/cancellation request at the local user-network interface to which the particular message applies. The call reference has only local significance, rather than having an end-to-end significance across ISDNs. The length subfield identifier specifies

the length of the remainder of the fields in octets. Its value is one for the basic rate interface and two for the primary rate interface. The call reference value is the number assigned to a particular call, it uniquely identifies the call and is used by future messages to specify a connection.

Message Type - The purpose of this field is to identify the function of the message being sent. This field has a separate bit pattern for each of the different messages. As an example, Alerting message is represented by 00000001 and Call Proceeding by 00000010. Q.931 messages can be grouped on the basis of the applications they support. The various applications are:

a) Circuit Mode Connection Control - refers to the functions needed to setup, maintain, and clear a circuit switched connection on the B-Channel.

b) Packet Mode Access Connection Control - refers to the functions needed to setup a circuit switched connection to an ISDN packet switching node, which connects the user to the packet switching network.

c) User-to-User Signaling not associated with circuit-switched calls - allows two users to communicate without setting up a circuit switched connection on the B-Channel. A temporary signaling connection is established and cleared in a manner similar to the control of circuit switched connection. Signaling takes place over the D-Channel and thus does not consume the B-Channel resources.

Other Information Element - Two categories of information elements are defined

a) Single Octet Information Element: represented as in figure 2.7

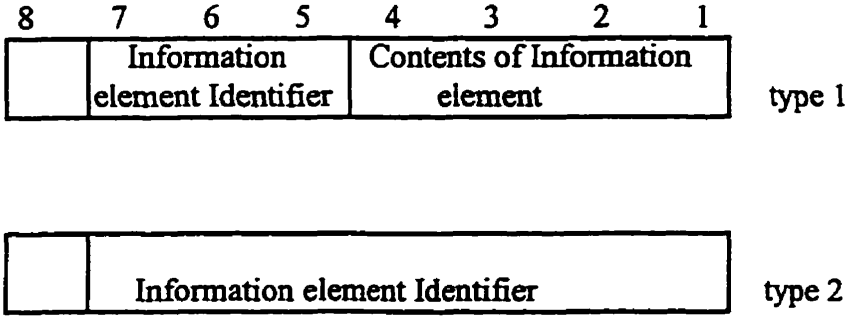


Figure 2.7: Two types of Single Octet Information Element

b) Variable Length Information Element: represented as in figure 2.8

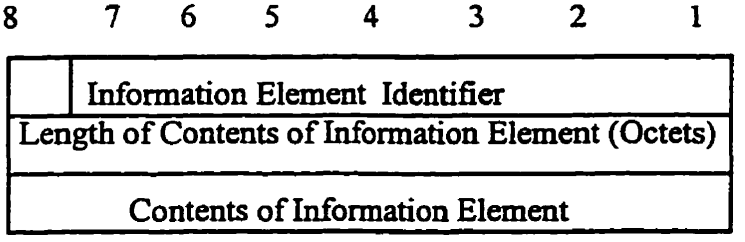


Figure 2.8: Variable Length Information Element

2.3.2.3 Q.931 message flow and Operation

The Q.931 message flow and operation has been described in the [ITUQ931] – ISDN User- Network Interface Layer 3 Specification for Basic Call Control (Q.931).

The Q.931 Call establishment module forwards the SETUP message to the LAP-D entity, which forwards that message on the D-Channel, to the network. The Q.931 Calling Module enters the Call Initiated State. The SETUP message contains the address of the Called Party. The network forwards the SETUP message to the Called Party and replies with the Call Proceeding to the Calling Q.931 module. The Called

Party and replies with the Call Proceeding to the Calling Q.931 module. The Called Party on receiving the SETUP enters the Call Present State. The Calling Q.931 module on receiving the Call Proceeding, enters the Outgoing call Proceeding state and is waiting for Alert message.

a) **Call Establishment at the Calling Party End -**

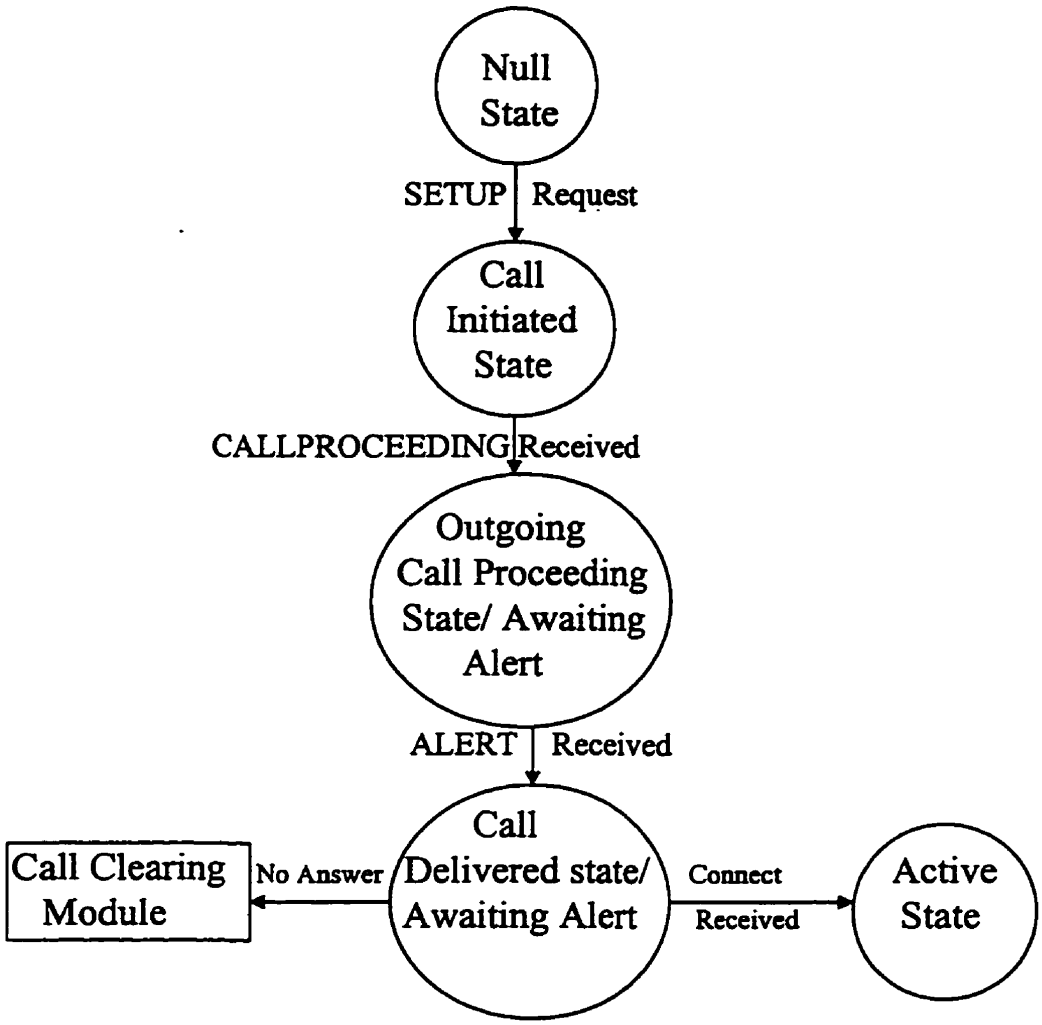


Figure 2.9: Q.931 State Transition Diagram and Message Flow during Call Establishment at Calling Party End

b) Call Establishment at Called Party End

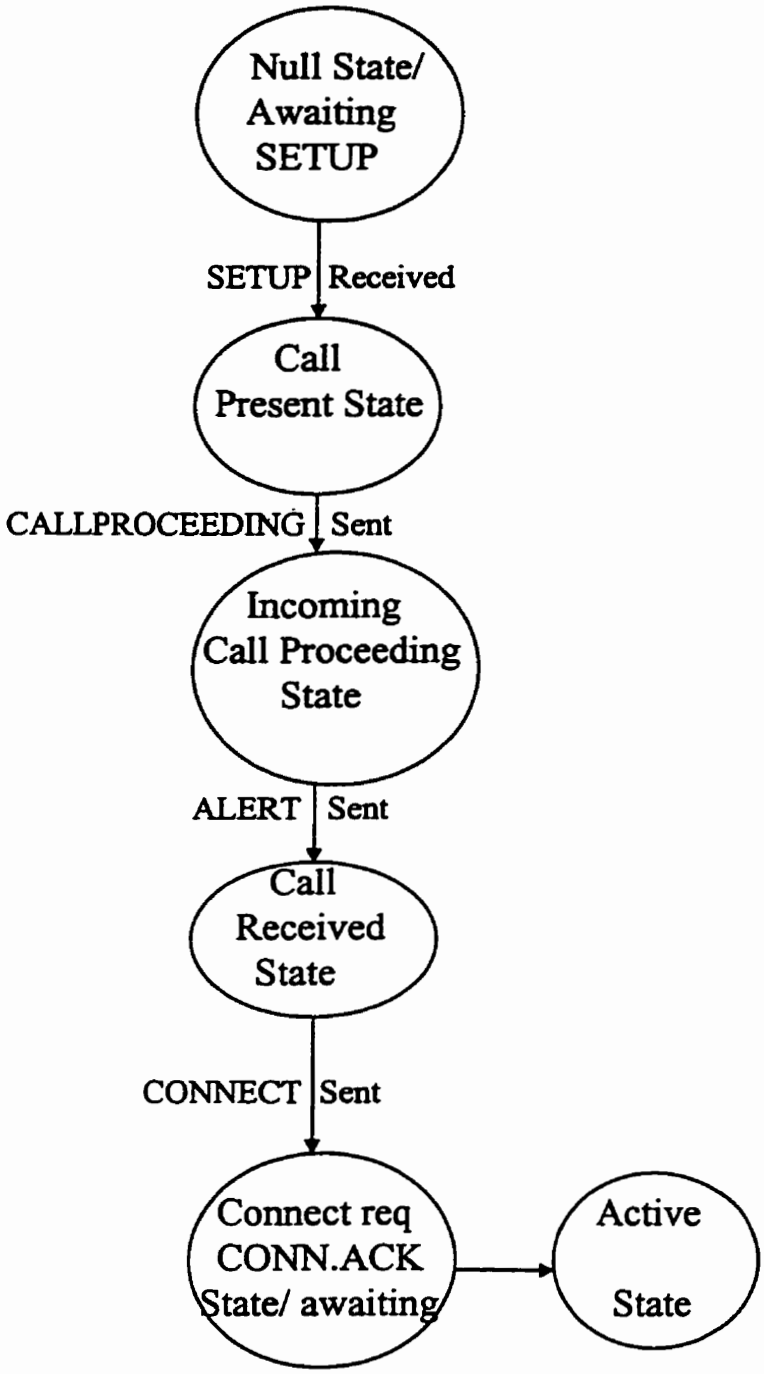


Figure 2.10: Q.931 State Transition Diagram and message flow during Call Establishment at Called Party End

The Called Party Q.931 module on receiving the SETUP message responds with a Call Proceeding message to the network, and comes to Incoming Call Proceeding state. Once the phone at the Called Party end starts ringing, the Q.931 Called Module sends the ALERT message to the network and comes to the Call Received state. The network forwards the ALERT message to the Calling Party. The Calling Q.931 module on receiving the ALERT, enters Call Delivered State and waits for the Connect message. When the user at the Called end picks up the phone, the Called Q.931 module sends the Connect message to the network and waits for the CONNECT ACK from the network. The network forwards the Connect message to the Calling module, and the Calling Module enters the Active state in which the user information is being transferred. The Called Module on receiving the CONNECT ACK enters the Active State.

The Call Clearing Module takes care of terminating the call. The termination of the call can be initiated by the user or the network or in case of fault situation. The user initiates by sending the DISCONNECT message to the network. The user disconnects the B-Channel and waits for the RELEASE message from the network. If the user receives the RELEASE message from the network, it releases the B-Channel and sends the RELEASE COMPLETE message to the network and enters the NULL state.

c) **Call Clearing**

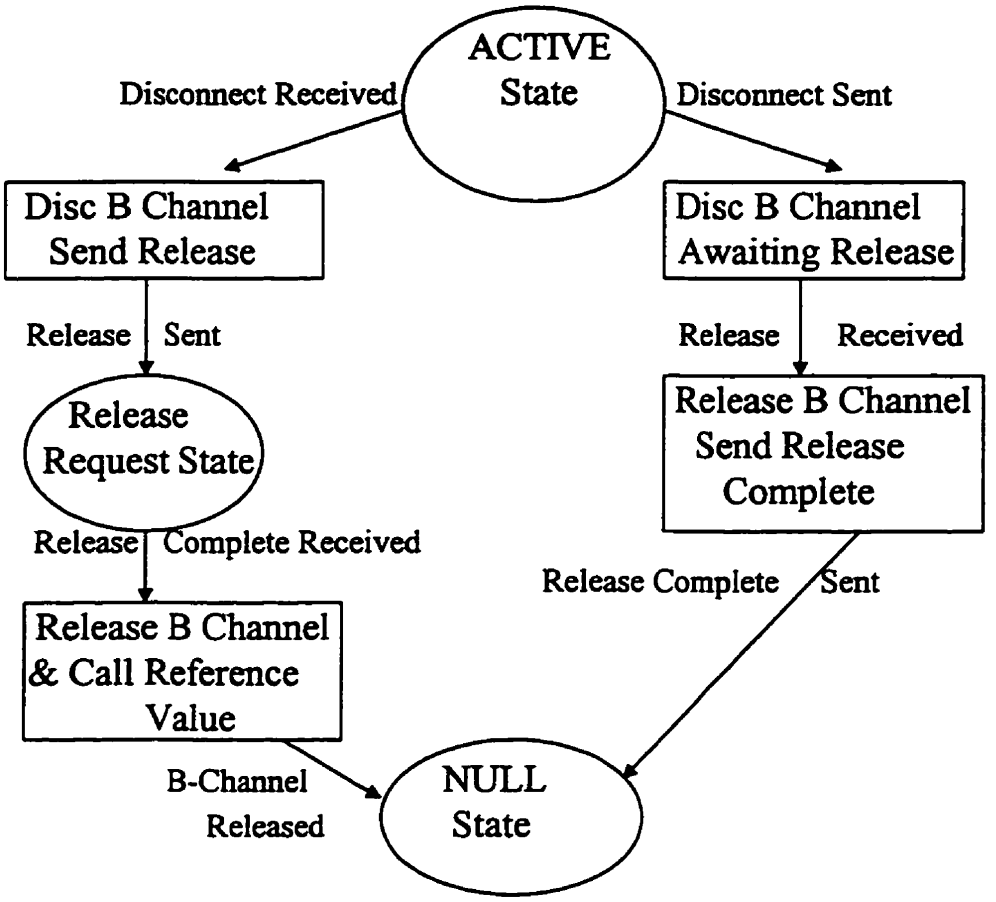


Figure 2.11: Q.931 State Transition Diagram during Call Clearing

If the DISCONNECT message is received from the network, the user disconnects the B-Channel, sends the RELEASE message to the network and enters the release request state. If the network sends the RELEASE COMPLETE message, the Q.931 entity at the user end releases the B-Channel and the Call Reference value.

2.3.3 User-to-User Signaling

The User-to-User Signaling allows two users to communicate without setting up a circuit switched connection as specified in the [ITUQ957] – Stage 3 description for additional information transfer supplementary services using DSS1: User-to-User Signaling. The User-to-User Signaling can be implicit or explicit.

2.3.3.1 Implicit User-to-User Signaling

In the SETUP, CONNECT, DISCONNECT, RELEASE, ALERTING and RELEASE COMPLETE messages there is a USER-to-USER information field. The purpose of this field is to convey information between ISDN users. This information is not interpreted by the network, but is rather carried transparently and delivered to the remote user(s). The User-to-User information element has network dependent maximum size of 35 or 131 octets. The user information is structured according to user needs. This is called Implicit User-to-User signaling as these messages are primarily used to establish circuit mode connection, but implicitly deliver information from one user to the other, keeping the network transparent.

2.3.3.2 Explicit User-to-User Signaling

Explicit User-to-User Signaling allows the two users to communicate by means of User-to-User signaling on the D-Channel without setting up a circuit switched connection on the B-Channel. Temporary signaling connection is established and cleared on the D-Channel, in a manner similar to the control of circuit switched connection on the B-Channel.

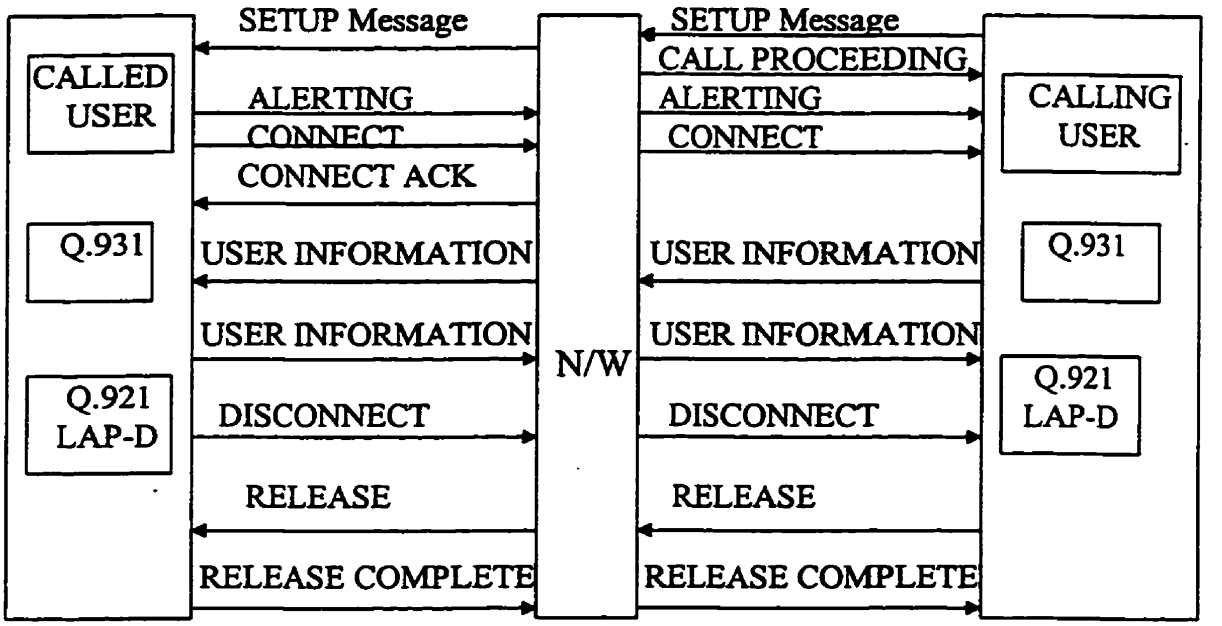


Figure 2.12 : Message flow in Explicit User-to-User Signaling Temporary Connection on the D-Channel

The message flow for temporary connection establishment is as shown in figure 2.12. The calling user sends a SETUP message identifying, within Bearer capability and channel identification information elements, a temporary signaling connection to be established on the D-Channel, with SAPI = 0. The bearer capability information element indicates unrestricted digital information in the information transfer capability field, packet mode in the transfer mode field, and D-Channel in the Channel indicator field. The called user accepts the temporary signaling connection request by sending a CONNECT message towards the calling user. After the called user has received a CONNECT ACKNOWLEDGE and calling user receives CONNECT, the temporary signaling connection is set up. Once the temporary signaling connection is established,

both users can transfer information between themselves by transferring User Information messages across user-network interface. The network provides for the transfer of such messages from the called user to the calling user side and vice versa. The User information messages includes the Call reference, the Protocol discriminator, and the User-to-User information elements. This provides entire 16Kbps bandwidth of the D-Channel for the transfer of User-User Information messages.

CHAPTER 3

High Level Scenarios for Real-Time Applications at Homes and Small Businesses

3.1 Classification of Real-Time Applications

There is a number of applications that can be supported by ISDN. We will be concentrating on the real-time applications that are suitable for home and small businesses. We have broadly classified these applications under three main headings:

3.1.1 TeleMonitoring

The term TeleMonitoring is the ability to monitor the data generated in real-time at remote locations. The data generated can be emergency data such as in the case of smoke detector and burglar alarms. TeleMonitoring allows us to monitor this data on the emergency basis. In case of utility meters (power, gas and water) the data is generated as and when the power, gas and the water is consumed. Such data needs to be monitored by the utility meter reading company on a periodic basis or on a demand basis. TeleMonitoring allows both the periodic and on demand monitoring of such data.

3.1.2 TeleControl

The term TeleControl is the ability to automatically control a device from a remote location. As an example, a person before leaving the work place, could call his home and turn on/off the air conditioner or the heater.

3.1.3 TelePolling

The term TelePolling is the ability to poll several devices from a distance. The remote video surveillance of different security cameras installed in different buildings, from a central control room is one of some examples. ISDN allows temporary connections to be established to the cameras at remote sites. The fast call setup time of ISDN allows a central operator to switch between the various cameras monitoring the sites. The high speed digital service and the high bandwidth provided by ISDN, makes it possible to transmit the image captured by the surveillance camera to the central office with a reliable quality of service.

3.2 High Level Scenarios for the Above Applications

3.2.1 TeleMonitoring

As mentioned in the section 3.1, there are three types of TeleMonitoring applications.

1) On Demand Monitoring - Utility meter reading company requests the meter readings from a particular customer's meter is one of the examples. This may help the company to study the power consumption behavior of a particular customer (such as an industry) in different hours of the day, and can generate power as per requirements.
2) Periodic Monitoring - All the customer's meters transmit the meter readings to the meter reading company on a monthly or periodic basis is an example of periodic monitoring. This will eliminate the need of meter reader to go to the site of the meter in order to collect the meter readings.
3) Monitoring on Emergency Basis - Devices such as burglar alarm, fire detector or carbon monoxide detector, go on/off as and when the emergency arises. The monitoring of such devices from a remote location such as fire department or the police department is one of the examples.

In the following section more detailed scenarios will be discussed to explain the functionality of these services over the ISDN.

3.2.1.1 On Demand Monitoring

Here we consider the example of On Demand Monitoring of the meter readings by the Utility Meter Reading Company. The On Demand Monitoring has to be initiated by the Utility Meter Reading Company, as the customer computers transmit the meter readings only on the periodic basis.

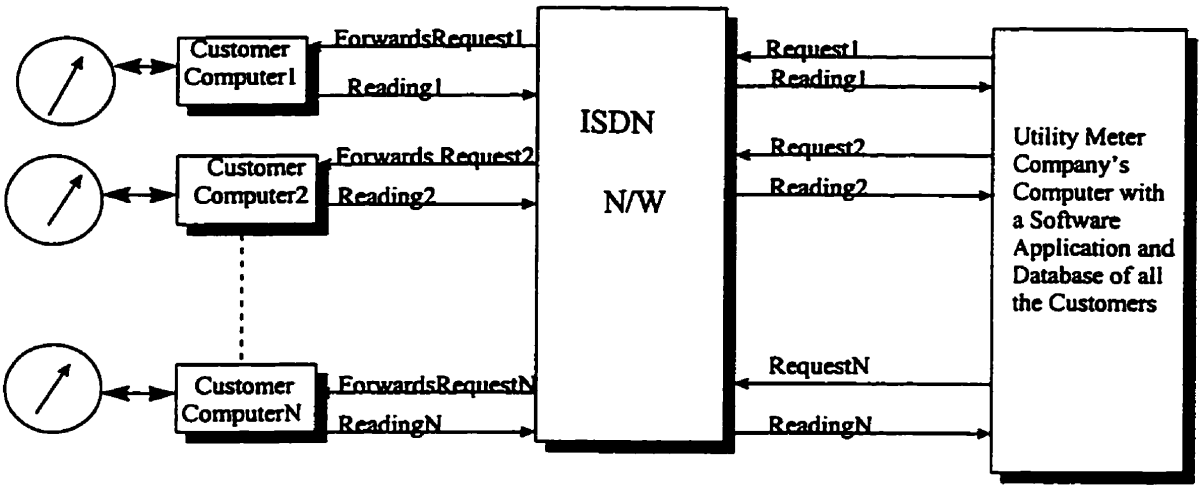


Figure 3.1: On-Demand monitoring of the meter readings by the utility meter reading company

In the above scenario, the computers at the customer's and the utility meter company's premise, have an ISDN interface card which is hooked to the ISDN line. The ISDN line provides 2B+D (Basic Rate Interface). Each computer has an ISDN LAP-D protocol entity on top of the ISDN physical layer (which is the ISDN card) and ISDN call control protocol entity (Q.931) sitting on top of LAP-D protocol. The applications at both ends are interfaced with the respective Q.931 entities. The above scenario can be accomplished by the Implicit User-to-User Signaling on the D-Channel.

The scenario is described in the following sequence of steps:

a) The application residing on the computer of the utility meter reading company maintains a database of their customers phone numbers. The application interacts with the ISDN call control protocol entity (Q.931). When the company needs the meter readings of a particular customer, the application triggers the Q.931 entity to establish an ISDN connection with the customer, via the ISDN switch using the ISDN protocol. As per the ISDN protocol, the connection establishment process involves the message communication between the calling party, ISDN switch and the called party via the D-Channel. The application on the computer of utility meter reading company sends the request on the D-Channel to the switch for a particular customer's meter reading. The request is sent as a part of one of the call control messages used for setting up a temporary connection between the two users, on the D-Channel.

b) The switch forwards the connection establishment message (which contains the request of meter readings also) to the customer's computer, via the D-Channel. The request goes as a part of one of the messages used for setting up a temporary connection, between the two users.

c) The application on the customer's computer is also interfaced with the ISDN call control protocol entity (Q.931). The ISDN call control protocol entity is sitting on the top of the ISDN LAP-D protocol entity. The application is polling on the call control protocol entity, for the particular call setup message, which also contains the request for meter readings. The application on receiving the request, in the setup message, will insert the meter readings in one of the reply messages for the temporary connection setup.

d) The switch on receiving the reply message, forwards the message to the computer of the meter reading company.

e) The application on the computer of the meter reading company is polling the Q.931 call control protocol entity for the reply message, for the connection which it had initiated. Once the Q.931 protocol entity receives the message, the application extracts the meter readings and stores in the database. The above sequence of steps allow the meter reading company to monitor the customer's meter reading as and when needed i.e. on-demand basis.

3.2.1.2 Monitoring On Periodic Basis

In this case the periodic monitoring of the meter readings is done by the switch. The hardware setup of the computers, ISDN card and an ISDN line at the customer and the meter reading company's premises is the same as in the previous scenario. The protocol setup at both the sites is the same. The application at the customer's computer responds to the meter reading request from the switch. The application at the utility meter reading company's computer, maintains the database of the meter readings which it receives from the switch. The difference in the setup is only at the switch. The switch has an application that interacts with the Q.931 entity at the switch and maintains the database of the customers phone numbers and the meter readings. All the Q.931 call control messages which are passed to the LAP-D entity, are sent to the destination LAP-D entity in the information field of the LAP-D frame, via the D-Channel. The destination LAP-D entity will pass the Q.931 call control message to the destination Q.931 call control entity.

In this scenario the application at the utility meter reading company does not have to interact with the customers for the meter readings. It will interact with the application at

the switch which is maintaining the database of the customers meter readings. This scenario will be using Implicit User-to-User signaling for the interaction between the switch and the customers and Explicit User-to-User signaling for interaction between the switch and the Utility meter reading company.

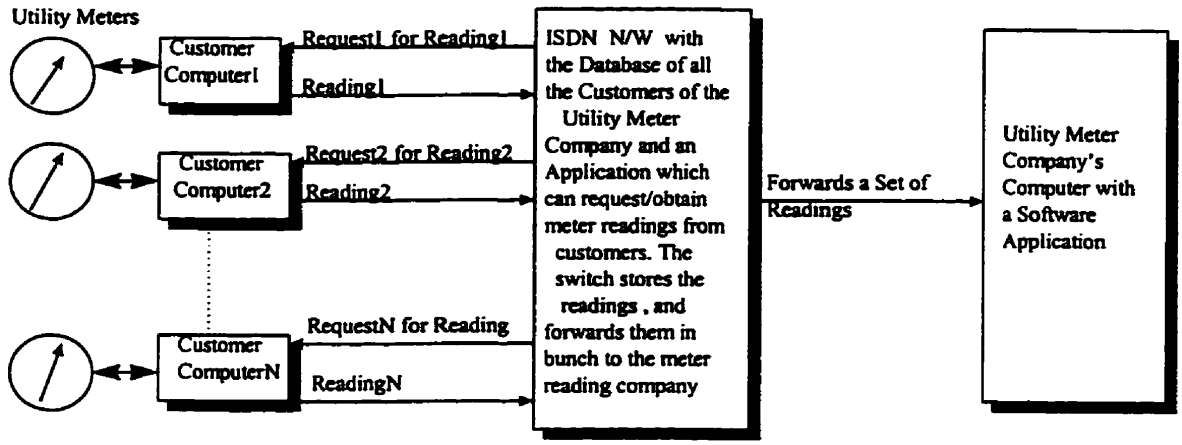


Figure 3.2 : Periodic monitoring of the meter readings by the application on the switch

a) The application at the switch will initiate the connection establishment process with all the customers. The connection establishment process involves the message communication between the calling party (which is the application on the switch), ISDN switch and the called party via the D-Channel. The application on the switch sends the request for the meter readings to each customers computer via the D-Channel. The request is sent as a part of one of the call control messages used for setting up a temporary connection between the two users, on the D-Channel (Implicit User-to-User Signaling).

b) The application on each customer's computer is also interfaced with the ISDN call control protocol (Q.931). The ISDN call control protocol is sitting on top of ISDN LAP-

D protocol. The application is polling on the call control protocol entity, for the particular call set up message, which also contains the request for meter readings (Implicit User-to-User Signaling). The application on receiving the request in the setup message will insert the meter readings in one of the reply messages for the temporary connection setup.

c) The application on the switch is polling the Q.931 call control protocol entity for the reply message, for the particular connection which it had initiated. Once the Q.931 protocol entity on the switch receives the message, the application extracts the meter readings and stores it in the data base.

d) Once the application on the switch receives all the meter readings, it initiates the connection establishment with the application on the utility meter reading company. The application on the switch establishes a temporary connection on the D-Channel, with the application on the utility meter reading company. Once this connection is established, the application on the switch starts sending the meter readings of all the customers in the user to user information messages, on the D-Channel (Explicit User-to-User Signaling). The application on the switch retrieves the meter readings from these User-to-User information messages, and stores them in the company's database. This process could be done every week, bi-weekly or monthly depending on the demand of the Utility meter reading company.

3.2.1.3 Monitoring On Emergency Basis

Remote Monitoring of the Fire Detector/Carbon Monoxide Detector or Burglar Alarm are few examples of the monitoring on emergency basis.

The following scenario explains the use of an ISDN connection, for monitoring of the detectors/alarms in general (Fire detector/Carbon Monoxide detector or Burglar Alarm)

from a remote location (figure 3.3). This application can let an alarm blow at a remote location because of fire or carbon monoxide leakage or blowing of a burglar alarm at a distant site. The hardware setup (computers, ISDN card and ISDN line) at the fire detector/burglar alarm site and police or fire department site is the same as in previous scenarios. The protocol architecture is the same as in previous scenarios, only the applications have different functionality. The detectors and the alarms are connected to one of the input/output ports of the computer.

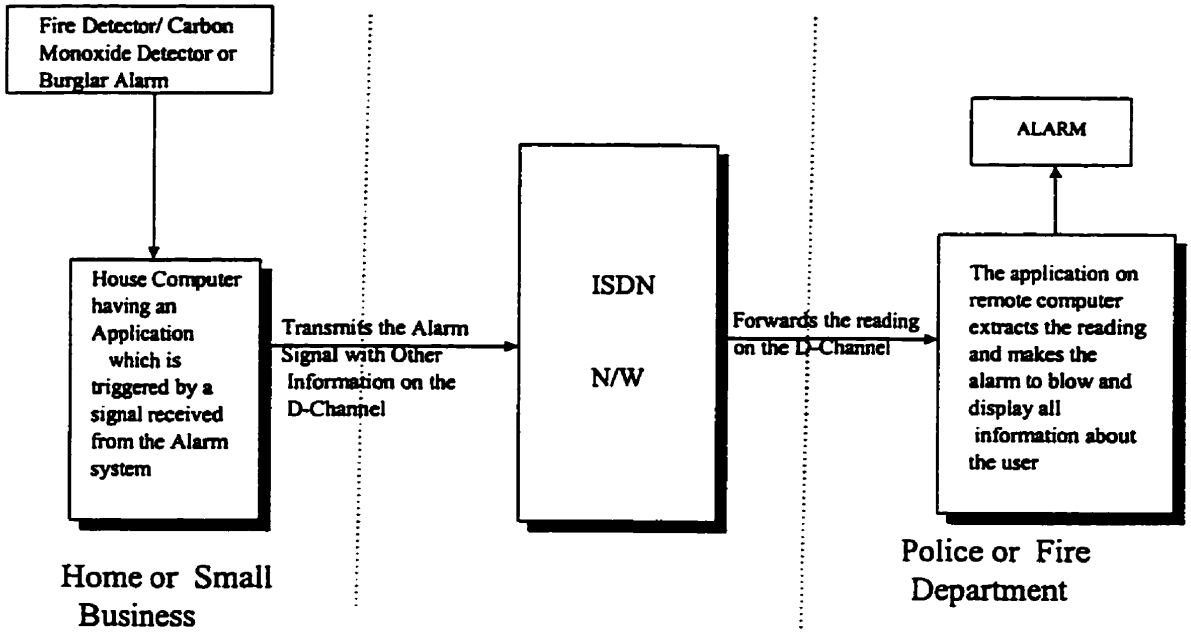


Figure 3.3 : Remote Monitoring of the Fire Detector/Carbon Monoxide Detector or Burglar Alarm on Emergency Basis.

a) As soon as the detector (fire/carbon monoxide) detects smoke/carbon monoxide or the burglar alarm detects the burglar, it creates a digital signal which is passed on to the computer via the input/output port. This signal will trigger the Q.931 entity to establish a temporary signaling connection with the monitoring site via the D-Channel. This connection establishment process involves message communication between the switch

and the calling party. Thus the application sends the setup message to the switch indicating the address of the remote site to which it wants to connect. The digital information of fire/carbon monoxide detection or burglar detection is passed in one of the fields of this message (Implicit User-to-User Signaling).

b) The switch on receiving the message, forwards it to the destination via the D-Channel. The destination LAP-D entity receives the message and passes it on to the Q.931 call control entity.

c) The application at the monitoring end is polling the Q.931 call control entity for that message. Once the Q.931 entity receives that message, the application extracts the relevant digital information about the fire/carbon monoxide detector or the burglar alarm. The application then sends an emergency signal to the alarm, hooked to the computer at the monitoring site, in order to indicate an emergency.

3.2.2 TeleControl

TeleControl allows a user to send the control information to any device at a remote location. The following scenario explains the use of an ISDN connection for controlling any remote device such as an air conditioner. The hardware setup (computers, ISDN card and the ISDN line) and the protocol architecture are the same as in previous scenarios. The applications are interfaced with the Q.931 call control protocol entity. The devices are connected to one of the input/output ports of the computer.

a) When the user wants to send the control information to the air conditioner, he/she invokes the application. He/She passes the destination address of the remote computer to which the air conditioner is hooked as well as the temperature to which the air conditioner is to be set. The application now activates the Q.931 call control entity to

initiate a temporary signaling connection establishment with the destination. The Q.931 entity forms a setup message, and the application inserts the temperature and other control information of the remote device in one of the fields of the message. This setup message is passed on to the LAP-D entity, which forms a LAP-D frame, by placing the setup message in its information field. The LAP-D entity sends this frame via D-Channel to the switch.

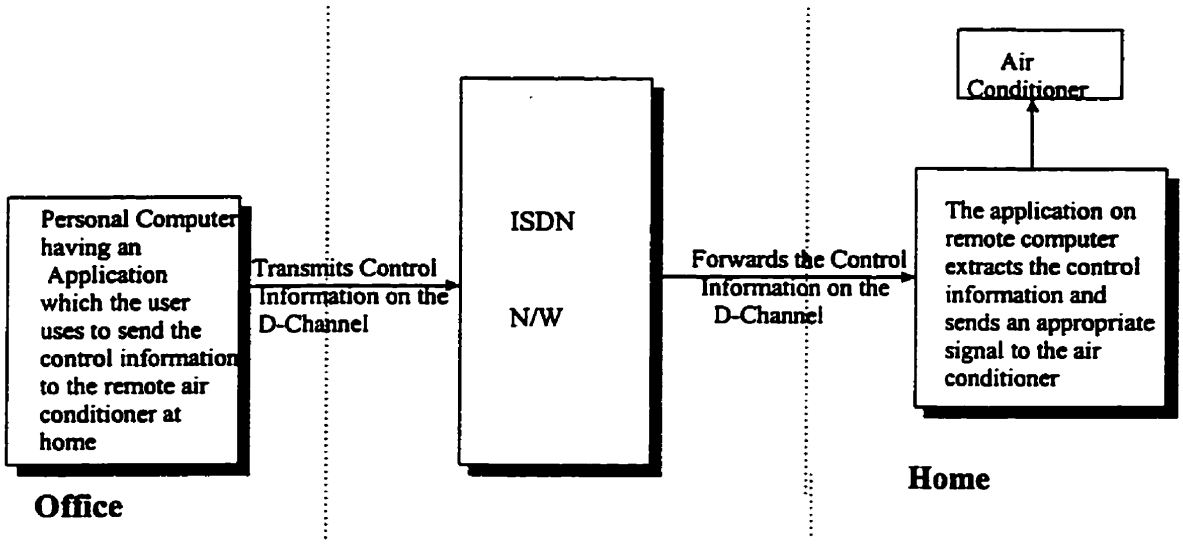


Figure 3.4 : TeleControl, Controlling an air conditioner at home from the work place

b) The switch on receiving the message forwards it to the destination via the D-Channel. The destination LAP-D entity receives the message and passes it on to the Q.931 call control entity.

c) The application at the remote end is polling the Q.931 entity for any setup message to be received, with a request for temporary signaling connection establishment. The LAP-D entity on receiving the setup message, removes the LAP-D header and passes the message to the Q.931 entity. As the Q.931 entity receives the message, the application extracts the

desired control information from the specific field of the message and sends the signal to the device accordingly (Implicit User-to-User Signaling). This scenario is similar to the scenario of TeleMonitoring on emergency basis.

3.2.3 TelePolling

Here we discuss in detail the remote video surveillance of different security cameras installed in different buildings, from a central control room. The surveillance cameras are connected to the input/output port of the computers in the building. The hardware setup (computers, ISDN card and the ISDN line) is the same as in previous scenarios. The protocol architecture is also the same. In this scenario the application in the central monitoring room will establish temporary signaling connections with the computers hooked to different security cameras in the different buildings. The entire snap shot (image) of the camera has to be transmitted from the site of the camera to the central control room. The amount of data is huge and cannot fit into the user-user information field of the setup message (Implicit User-to-User Signaling). Thus, once the temporary signaling connection is established between the two ends, the computers will exchange User-to-User information messages (Explicit User-to-User Signaling). In these messages, the image is transmitted from the security camera end to the central control room.

a) The operator in the central control room invokes the application to initiate a temporary signaling connection establishment with the computer to which the security camera is interfaced. The application passes the ISDN number of the remote computer to the Q.931 entity which forms the SETUP message. In the SETUP message the Q.931 entity will indicate that the D-Channel will be used for a temporary signaling connection. Once the

link is established, unrestricted digital information will be transferred on the D-Channel, between the two users.

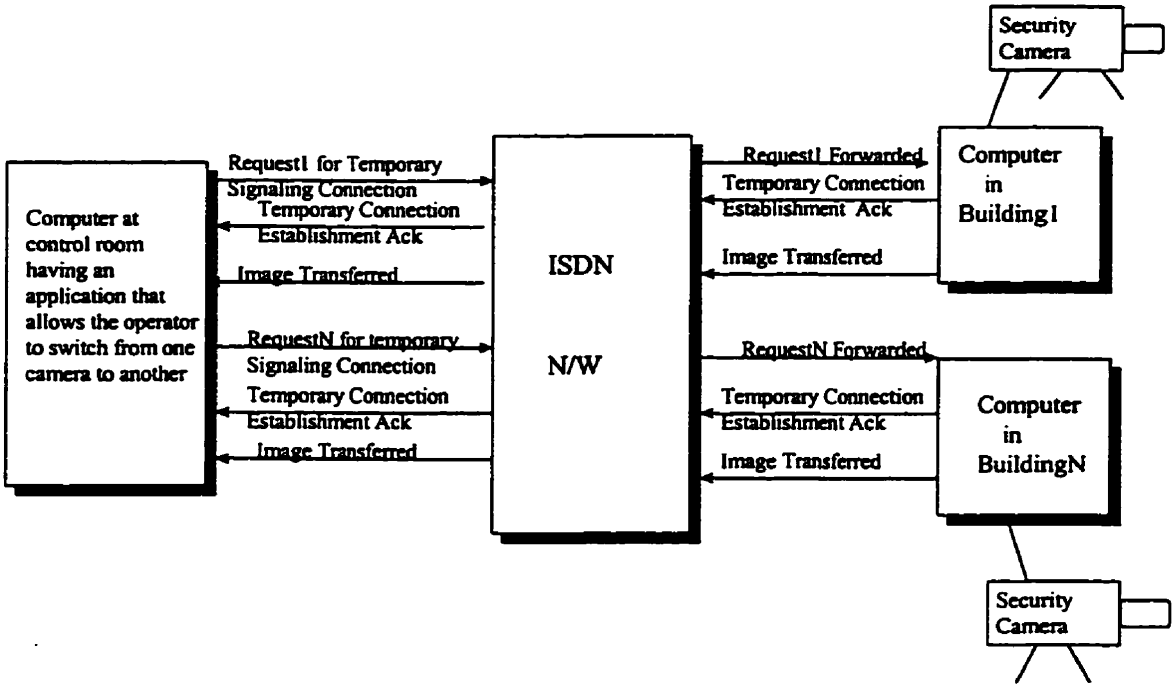


Figure 3.5: TelePolling of Surveillance cameras from the Central Control Room

b) The switch forwards the SETUP message to the destination computer and responds with the CALLPROCEEDING message to the calling Q.931 entity. The called Q.931 entity on getting SETUP will send ALERT to the switch which will forward the ALERT message to the calling Q.931 entity. The called Q.931 entity then sends the CONNECT message to the switch and the switch forwards that CONNECT to the calling Q.931 entity. At this stage the temporary signaling connection has been established between the two users.

c) The application at the security camera end is informed by the Q.931 entity that the temporary signaling connection is established. The application now inserts the digital image into the User-to-User information messages, which the Q.931 entity will pass on to the switch. The switch forwards these messages to the calling Q.931 entity.

d) The application at the central control room end is polling the Q.931 entity for the User-to-User information messages. Once the Q.931 entity receives the User-to-User information messages, the application extracts the digital information from them.

e) The application at the central control room end will initiate the connection release once it has received the entire image in the User-to-User information messages, and the two ends will release the D-Channel.

CHAPTER 4

Detailed Message Level Scenarios for the Real-Time Applications at Homes and Small Businesses

The high level scenarios for the Real-Time Applications at homes and small businesses have been described in the previous chapter. The main focus of this chapter is to describe in detail the ISDN protocol and the message level scenarios for these real-time applications.

4.1 Detailed Message Level Scenario for the TeleMonitoring Applications

The peer Q.931 entities exchange a series of Q.931 call control messages to establish and release a call. As shown in Figure 4.1, the SETUP, CALL PROCEEDING, ALERTING, CONNECT and CONNECT ACKNOWLEDGE messages are used by the peer Q.931 entities to establish the call. The DISCONNECT, RELEASE and RELEASE COMPLETE messages are used to disconnect the call. The bit level details of these messages are specified in [ITUQ931] – ISDN User-Network Interface Layer 3 Specification for Basic Call Control (Q.931).

The bit level details of the messages used in the TeleMonitoring Applications are given in the following sections.

4.1.1 SETUP Message

The application at the Data Monitoring End (DME) triggers the Q.931 entity to initiate a connection establishment with the Data Generating End (DGE). The Q.931 entity of the DME prepares a SETUP message, to send it to the Q.931 entity of the DGE. The applications make use of Implicit User-to-User Signaling (described in Chapter 2). The application at the DME sends the request for remotely generated data to its Q.931 entity.

The Q.931 entity inserts the request string in the User-to-User information field of the SETUP message.

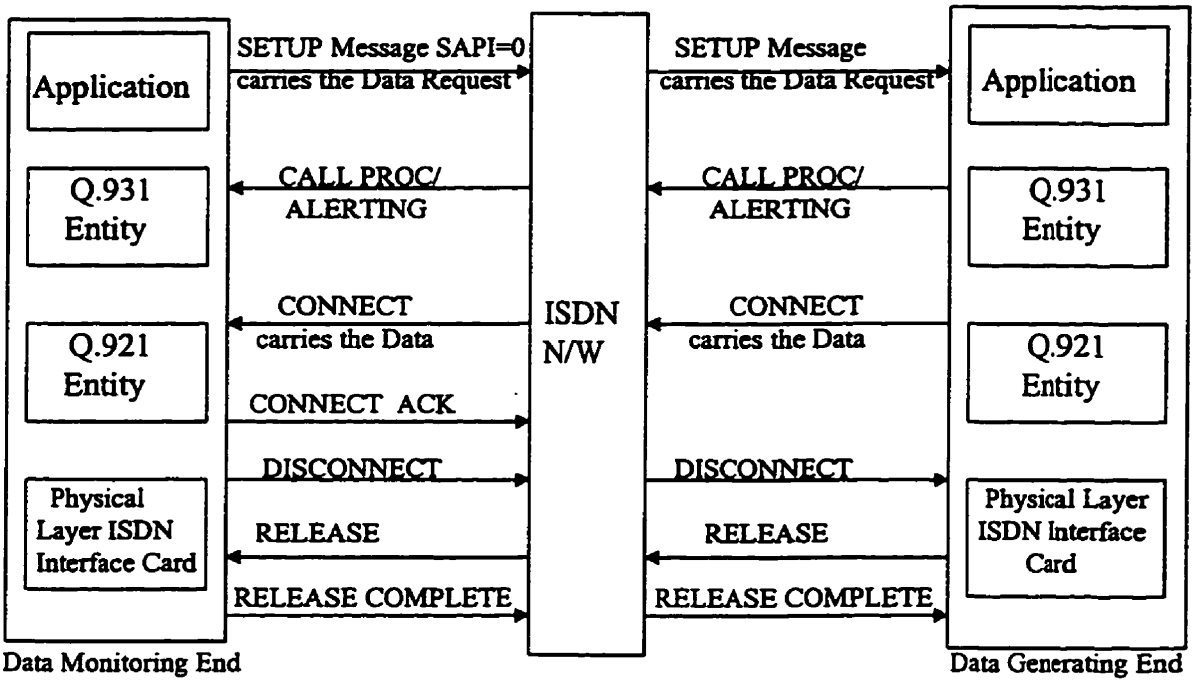


Fig 4.1 Message Level Scenario for TeleMonitoring Applications

The specific details of the fields of the SETUP message are as follows:

Protocol Discriminator: The Protocol Discriminator is one of the Mandatory Fields in every Q.931 message. The purpose of the protocol discriminator is to distinguish messages for user-network call control from other messages. For user-network call control messages the value of the protocol discriminator is one octet and its value is 0000 1000.

Call Reference: It is another mandatory field. The call reference information element is shown in Figure 4.2. The first four bits in the Octet one indicates the length of the call reference value and the next four bits are left for the future use. For the SETUP message in this scenario, the call reference value can be set to any value, though the flag field will

be zero, as it has local significance and it indicates that the message originates from this interface.

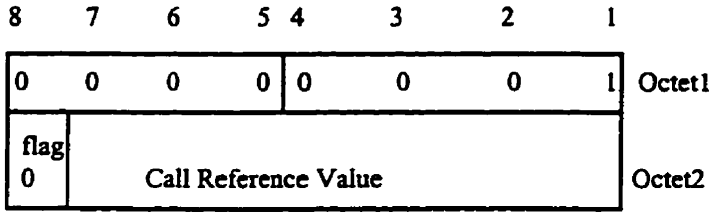


Figure 4.2: Call Reference Information Element

Message Type: It is the third field of every message. It identifies the function of every message being sent. For a setup message as part of call establishment message the value of this field is set to 0000 0101.

Sending Complete: The purpose of the Sending Complete information element is to optionally indicate completion of the called party number and also indicates that all information necessary for call establishment is included in the SETUP message. It is a one octet information field and is coded as 1010 0000. This field is an optional field and is not required in our application.

Bearer Capability: The purpose of bearer capability information element is to indicate a bearer service to be provided by the network. The various octets of the Bearer Capability information field in the SETUP message are encoded as follows :

Octet 1 : Bearer Capability Information Element Identifier - 0000 0100
Octet 2 : Length of bearer capability contents
Octet 3 :
bits 7 & 6 : 00 CCITT Standard coding
bits 5,4,3,2,1 : 01000 Unrestricted Digital information (Information Transfer Capability)
Octet 4:
bits 7 & 6 : 10 Packet mode (Transfer Mode)
bits 5,4,3,2,1 : 00000 (Information transfer rate - in case of packet mode - none)
Octet 5 : is omitted in case of Packet Mode as Transfer Mode

Octet 6 :
bits 5,4,3,2,1 : 00010 Recommendation Q.921 (User Information layer protocol)
Octet 7 :
bits 5,4,3,2,1 : 00010 Recommendation Q.931(User Information layer protocol)

Channel Identification Information Element : The purpose of Channel Identification Information element is to identify a channel within the interface controlled by these signaling procedures. This element includes the following :

Octet 1 : Channel identification information element identifier 0001 1000
Octet 2 : Length of channel identification contents
Octet 3 :
bit 7 : 0 Interface Implicitly identified. The interface which includes the D - Channel carrying this information element is indicated. (Interface Identifier Present)
bit 6 : 0 Basic Interface (Interface type)
bit 4 : 1 Exclusive, Only the indicated channel is acceptable. Preferred/Exclusive has significance only for B-Channel selection. (preferred/exclusive)
bit 3 : 0 Channel identified is not the D-Channel (D-Channel indicator)
bits 2,1 : 11 B Channel - The information channel selection does not apply to the D-Channel (Information Channel Selection)
bit 0 : left for future use

Network Specific facility: It is an optional field and can be omitted in this message. The purpose of this element is to indicate which network facilities are to be invoked.

Display information element: This field is not included in the message because it goes only in one direction, i.e. from the network to the user. The purpose of the Display information element is to supply display information that may be displayed by the user. This information element is included in the message, if the network provides information that can be presented to the user.

Keypad facility Information Element : This is an optional field and can be omitted in this message. This information element includes the Called Party number information

when user wants to convey that number to the network, during overlap sending. In this scenario (TeleMonitoring) the field is not included in the message, as overlap sending is not used.

Calling Party Number Information Element: It is an optional field and is included in the message from both directions i.e. from the user to network and from the network to the user. It may be included by the calling user or the network to identify the calling user. In our scenario (TeleMonitoring), this field is not included in the SETUP message.

Called Party Number Information Element : This is an optional field included in the message if the Called Party Number is to be conveyed by the user to the network or by the network to the user. In this scenario the called party number is conveyed by the application to the Q.931 entity and has to be conveyed to the network, hence this field is included in the message.

User-to-User Information Element : This information element is not a part of the regular message, but is used to enhance the capability of User-to-User signaling. Details on this capability are given in recommendations Q.957 [ITUQ957] (Explicit and Implicit Type 1 user-user signaling). In the TeleMonitoring scenario, we are using the Implicit User-to-User signaling. The request string forwarded by the application to the Q.931 entity for the remotely generated data is inserted into this field. The size of this element is either 35 or 131 octets. This information element is the part of the SETUP message in the TeleMonitoring application.

All the other optional information elements in the SETUP message can be omitted, including the Calling Party Subaddress, Called Party Subaddress, Transit network

selection, Low Layer Compatibility, High Layer Compatibility, as they are not carrying any information relevant to the TeleMonitoring application.

4.1.2 CALL PROCEEDING Message

The SETUP message with the above details is transferred to the network. The network transfers the SETUP message to the Called Party or DGE after analyzing the Called Party Number. The network sends the CALL PROCEEDING to the Calling Party or DME. The Called Party (DGE) also responds with the CALL PROCEEDING message in response to the SETUP message from the network. The details of the CALL PROCEEDING message are as follows:

Protocol Discriminator: This field has the same value (00001000) as that in the SETUP message, as CALL PROCEEDING is also a User-Network call control message.

Call Reference: The Call Reference value will be the same as set in the SETUP message. Only the flag field will be 1 in this case as this message is not coming from the originating side, but is coming from the network to the originating end. When this message is coming from the Called Party to the network, the flag field will still be 1 since it is not coming from the originating end, but a new Call Reference value will be assigned.

Message Type: The message type field for the CALL PROCEEDING message is 00010.

Channel Identification: This field is mandatory since it is the first message in response to the SETUP message, from the network to the user. The field is the same as that in the SETUP message.

User-to-User Information Element: This element is not included in the CALL PROCEEDING message in this scenario as the message is coming from the network and

we do not have any data to be transmitted from the network to the Calling User. When the CALL PROCEEDING message is coming from the Called Party to the network, this field is still not included in the message, as the network does not transfer this message back to the Calling party (Data Monitoring End). Hence any data which the Called Party wants to send to the Calling Party won't be delivered through this field.

4.1.3 CONNECT Message

The Called Party (Data Generating End) now responds with the CONNECT message to the network. The network transfers this CONNECT message to the Calling Party (Data Monitoring End) to indicate the acceptance of the call by the Called Party. As the CONNECT message is transferred by the network to the Calling Party (Data Monitoring End), the application at the Data Generating End (Called Party) inserts the requested data in the User-to-User Information element of this message. The application at the Data Monitoring End extracts the Data from the User-to-User information field of this message. The details of the various fields of the CONNECT message are as follows:

Protocol Discriminator: This field is the same as in earlier messages as it is one of the user-network call control message.

Call Reference: When the message is coming from the Called Party to the network, the Call Reference value will be the same as in the CALL PROCEEDING message sent from the Called Party to the network and the flag value is 1 as the message is not from the originating side. When the message is going from the network to the Calling Party the Call Reference value is the same as set in the SETUP message (from the Calling Party to the network) and the flag value is still 1 as the message is from the network to the Originating side.

Message Type: The message type field for the CONNECT message is 00111.

Channel Identification: By the time this message is transmitted and received by the Called Party and the network, the Channel has already been identified by the Called Party, Calling Party and the network through the SETUP message, thus this field is not required in this message.

User-to-User Information Element: In this scenario, this information element will be included in the message. We are exploring Implicit User-to-User signaling, as the size of the data to be transferred can easily fit in the 131 octets of the User-to-User information Element of the Call Control messages. The generated data forwarded by the Application at the DGE to the Q.931 entity is inserted into this field of the CONNECT message. The CONNECT message from the Called Party to the network carries the remotely generated data. This data is transferred to the Calling User in the User-to-User information field of the CONNECT message transferred from the network to the Calling User. The size of this element is 131 octets.

4.1.4 DISCONNECT Message

Once the application at the DME extracts the data from the User-to-User information field of the CONNECT message, it triggers the Q.931 entity to clear the call. The Q.931 entity at the DME sends a DISCONNECT message to the network. The network forwards the DISCONNECT message to the Called Party (DGE). The various fields of the DISCONNECT message are as follows:

Protocol Discriminator: DISCONNECT is also a user-network call control message, thus its protocol discriminator field is 0000 1000.

Call Reference: The Call Reference value for the DISCONNECT message from the Calling user to the network will be the same as set in the SETUP message from the Calling User to the network. The flag field will be 0 as the message is coming from the call originating end to the network. The Call Reference value for the DISCONNECT message from the network to the Called Party has the same value as set in the CALL PROCEEDING message from the Called Party to the network. The flag value will be 1 in this case as the message is coming to the call originating end.

Message Type: The message type field for the DISCONNECT message has a value of 00101.

4.1.5 RELEASE and RELEASE COMPLETE Messages

The network on receiving the DISCONNECT message from the Calling Party responds with a RELEASE message to the Calling Party. The Calling Party responds with a RELEASE COMPLETE to the network indicating that all the resources have been released and the call has been cleared at this end.

The Called Party responds with the RELEASE message to the network in response to the DISCONNECT message from the network, indicating the release of resources at its end. The network responds with the RELEASE COMPLETE to the Called Party, indicating the clearing of the call from its end.

The Protocol Discriminator, Call Reference values for the RELEASE and RELEASE COMPLETE messages will be the same as in the previous messages. The Message type field of the RELEASE message has a value of 01101 and that of RELEASE COMPLETE is 11010.

4.2 Message Level Scenario for the TeleControl Application

The detailed message level scenario for the TeleControl Application is described in the figure 4.3. This scenario is a subset of the TeleMonitoring application as the control information is transferred in the User-to-User Information element of the SETUP message from the site of control to the site of the remote device. The call is released immediately after the SETUP message is received by the site of the remote device.

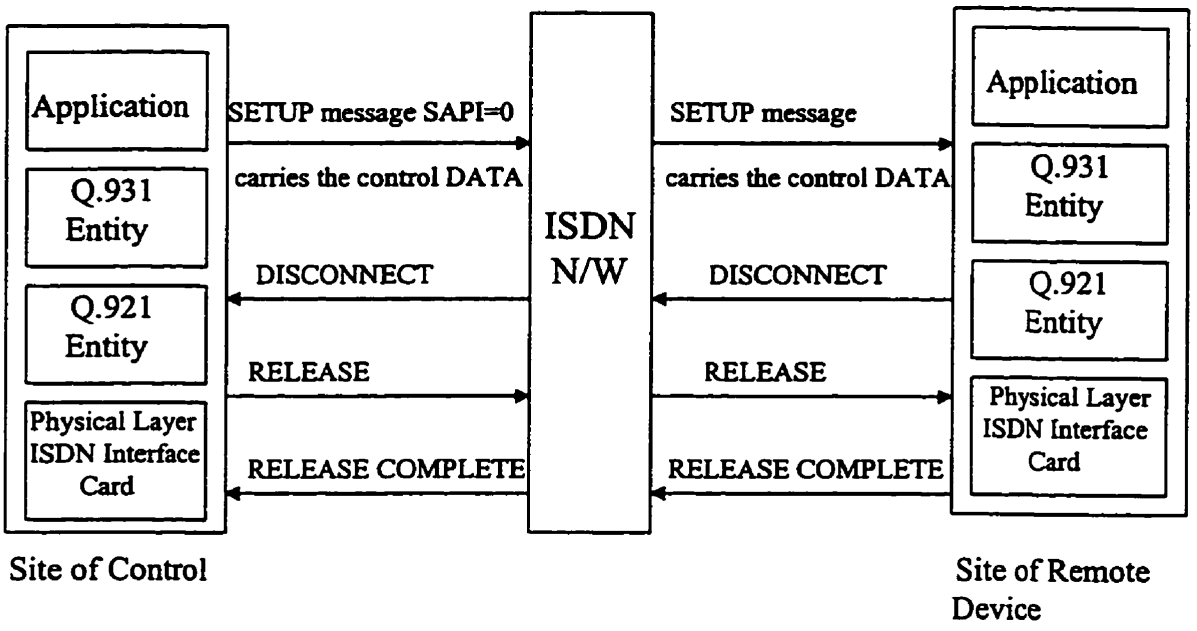


Fig 4.3 Message Level Scenario for TeleControl Application

In the above scenario there is a site from which the remote device is controlled and the site at which the remote device is installed. The configuration at both the sites has been described in the previous chapter. The application at the site of Control triggers the Q.931 entity to initiate the connection establishment with the remote site at the which the device is installed. The Q.931 entity prepares a SETUP message, to send it to the network. The applications make use of Implicit User-to-User Signaling (described in chapter 2). The

application sends the control data for the device to the Q.931 entity. The Q.931 entity inserts this control data in the User-to-User information field of the SETUP message. The bit level details of this SETUP message are exactly the same as in the TeleMonitoring scenario, except that the User-to-User information field contains the Control data for the remote device.

The SETUP message with the above details is transferred to the network. The network transfers the SETUP message to the Called Party (Remote Device End) after analyzing the Called Party Number. It transfers the User-to-User information field received from the Calling Party (Control site) to the Called Party (remote device end). The application at the Remote Device end extracts the Control information from the User-to-User information field of the SETUP message and puts that control information on the port at which the remote device is connected in order to control the device. The application at the remote device end initiates the Call Clearing by sending the DISCONNECT to the network. The network responds with the RELEASE message to the Called Party and forwards the DISCONNECT to the Calling Party. The Called party responds with the RELEASE COMPLETE to the network indicating the resource clearing at its end. The Calling Party in response to the DISCONNECT message, sends RELEASE to the network. The network responds with RELEASE COMPLETE to the Calling Party indicating call clearing at its end. The details of the bits in these messages are the same as in the TeleMonitoring Application.

4.3 Message Level Scenario for the TelePolling Applications

The High level Scenario and the bandwidth requirement of the TelePolling Application has been described in the previous chapter. The detailed message level scenario is described in the figure 4.3. In this scenario we are making use of explicit User-to-User signaling on the D-Channel.

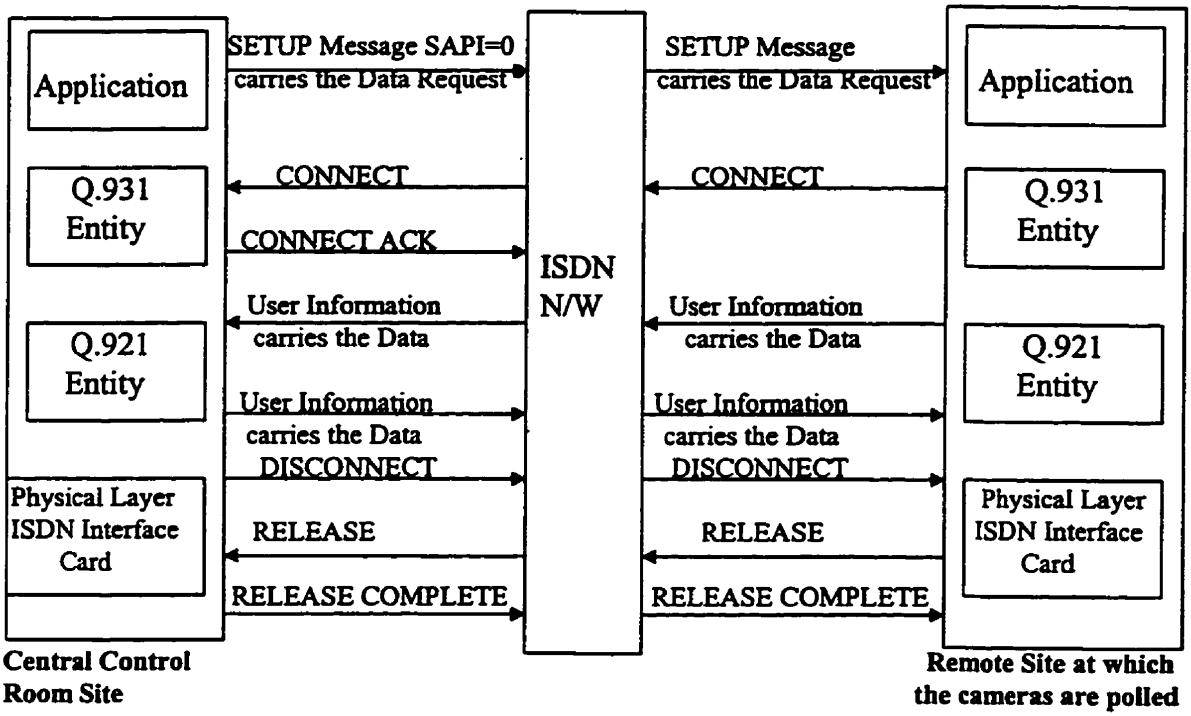


Fig 4.4 Message Level Scenario for TelePolling Application

The operator in the central control room will use the Application to trigger the Q.931 entity, so that the Q.931 entity initiates a temporary signaling connection establishment with the remote site at which the camera is installed. Firstly, the Q.931 entity prepares a SETUP message. The encoding of the SETUP message is similar to what was described in the previous scenarios, excluding the following elements:

Bearer Capability information element

- Unrestricted digital information in the information transfer capability field.
- Packet mode in the transfer mode field.
- User information layer 2 protocol is Recommendation Q.921 and user information layer3 protocol is Q.931, in the layer and protocol identification fields respectively.

Channel Identification information element

- Exclusive in the preferred/exclusive field.
- D-Channel in the Channel indicator field.
- No channel in the channel selection field (B-Channel selection).

The network transfers the SETUP message to the Called Party (remote camera site). The Called Party accepts the temporary signaling connection request by sending a CONNECT message towards the network. The details of the CONNECT message are the same as in the previous scenarios. The network transfers the CONNECT message to the Calling Party (Central Control Room and sends the CONNECT ACKNOWLEDGE to the Called User i.e. the remote camera site). After the called user has received a CONNECT ACKNOWLEDGE and calling user receives CONNECT, the temporary signaling connection is set up. Once the temporary signaling connection is established, both users can transfer information between themselves by transferring User Information messages across the user-network interface. The network provides for the transfer of such messages from the called user to the calling user side and vice versa. The Q.931 User Information message includes the Call reference, the Protocol discriminator, and the User-to-User information elements. This provides entire 16Kbps bandwidth of the D-Channel for the transfer of User-User Information messages. Thus the application at the camera site inserts the digital image in the User Information message which is extracted by the

application at the Central control room. Once the entire image is transmitted the application at the Central control room site initiates the clearing of the connection in the manner similar to the release of a normal connection by following a sequence of DISCONNECT, RELEASE and RELEASE COMPLETE messages between the two users and the network. The application at the Central control room can now initiate the connection establishment with the different camera in the building in order to switch from one camera to another.

CHAPTER 5

The Design and Implementation of the Selected Protocol Model

5.1 Introduction

In order to perform operational testing for the applications described in the previous chapters, an ISDN platform is required. There are three possible approaches for such testing:

1. The first approach requires two workstations interfaced with basic rate ISDN cards. The ISDN cards are connected to the existing ISDN connections that gives connectivity to the Telecommunications networks (ISDN switch). It is important to have an access to the internal details of the software existing on the ISDN card (LAP-D and Q.931 call control protocols) in order to interface our applications with the protocols. Such details would be very useful for our work in order to trace all the activities involved within the ISDN protocols. But manufacturers of these cards do not provide an access to these internal details, which makes it impossible to use this approach. Also, not all the ISDN switches provide telemetry (User-to-User signaling) support, which is usually provided as an optional feature with the switches. Finally, the lack of resources (ISDN lines and interface cards) is another reason, to search for another approach for our testing method.
2. A second approach deals with the building our own protocol model for LAP-D and Q.931 call control protocols. But implementing such protocols is a time consuming process and that is not our primary concern in this research.

3. The failure of using the previous two approaches, led to our search for an available protocol model, with an access to the source code. This gives us the flexibility to interface our applications with the protocols and test our applications.

5.2 The Selected Protocol Model

We have selected an available protocol model that implements the ITU-T (International Telecommunication Union - Telecommunication) defined recommendations, Q.921 and Q.931 in Sun Solaris environment. The protocol model as shown in Figure 5.1 has been developed in Helsinki Institute of Technology by the Sunshine project development team and is freely available with the entire source code for any research, educational or commercial purposes [Suns96]. The protocol model is developed on SUN SPARC workstation in Solaris environment and is interfaced with the Dual Basic Rate ISDN Card. We have studied the implementation of this protocol model in detail, and have modified this model to develop a framework to support the real-time applications.

The original Sunshine protocol model as shown in Figure 5.1 has three main modules, the Q.921 pseudo multiplexing device driver, Data Link Provider Interface (DLPI) and the Q.931 server. The Q.921 module is implemented in the kernel, as a pseudo multiplexing device driver. The pseudo multiplexing device driver provides a pair of queues, a read queue and a write queue. The lower queues of the pseudo multiplexing device driver are linked with the D-Channel of the Dual Basic Rate ISDN interface card. The upper queues of the pseudo multiplexing device driver are linked with Q.931 module via Data Link Provider Interface. The Data Link Provider Interface provides an interface to the services of the Data Link Layer. This interface is the boundary between

the network and the data link layer of the Open System Interconnection (OSI) model. The network layer entity is the Data Link service user, whereas the Data Link Layer entity is the Data Link Service provider. The Q.931 entity is implemented as a server on the TCP/IP network. The Q.931 entity opens a socket port and listens for a connection on that port. The application which wants to connect to this server can reside anywhere on the TCP/IP network, and will connect to the port on which Q.931 server is listening to accept a connection. This allows different applications on the TCP/IP network to connect to the Q.931 server, so several applications can make use of single ISDN connection.

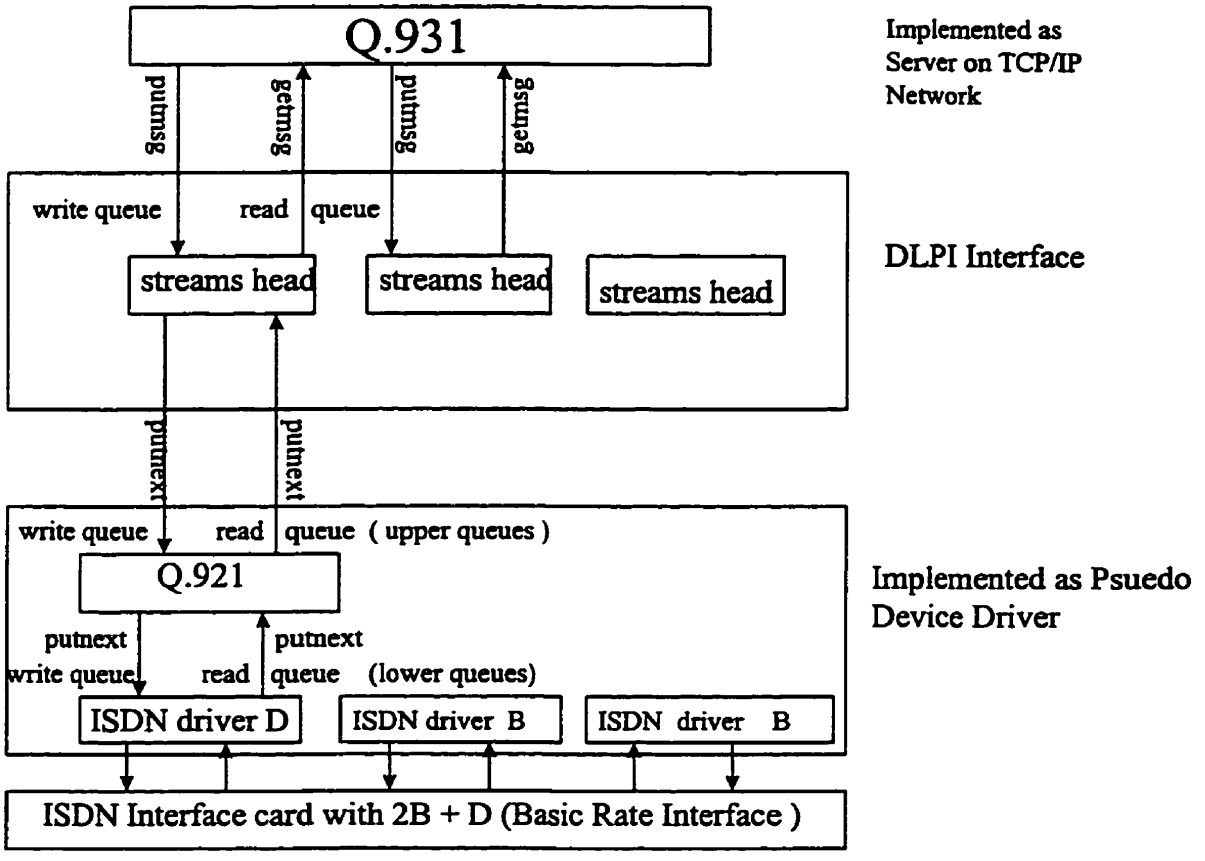


Figure 5.1: ISDN Protocol Model

5.3 Q.921 Multiplexing Device Driver

The Q.921 protocol is built into a STREAMS device driver, within the kernel in the SUN Solaris environment. STREAMS provides a full duplex connection between a user process and a device driver. It provides an alternative to the traditional Unix character input/output. The top portion of the STREAM in the kernel is known as STREAM head. It defines I/O within the kernel and between the kernel and user space. STREAMS supports implementations of the data communication protocols and has mechanisms for implementing flow control and multiplexing. A nice feature about STREAMS is that a process can add modules between the stream head and the device driver. The Q.921 pseudo multiplexing device driver supports multiplexing by cloning. Every time a new clone is opened, a new stream to the user space will be opened. The clones share the same data structures. Therefore every time a new stream is opened from user space, new queue pointers have to be saved. All the clones will have different queue pointers and different state information. In this implementation maximum of three clones is permitted.

The system dependent functions which have to be implemented in all drivers are as follows :

a) **_init** - It initializes the driver when loading it into the kernel. All the data structures that store the queue pointers, the Q.921 state information and the counters in the Q.921 module are also initialized in this function.

b) **_fini** - It removes the driver from the kernel when issuing modunload[1] command.

---

[1] modunload command unloads a loaded module from the kernel.

c) **_info** - It returns a data structure which contains the information about the loadable module.

d) **mux_identify** - It is used to identify if the driver drives a specified device.

e) **mux_probe** - It is used to determine if the driver is loaded into the kernel.

f) **mux_attach** - It is used to attach a device instance to the system.

g) **mux_detach** - It is used to detach a device instance from the system.

STREAMS provide functions for flow control, that will check if the queues are full before putting any message on them. The major functions of the Q.921 pseudo multiplexing device driver that take care of multiplexing and flow control are as follows:

a) **Q921UWPUT** - It gets the messages from the write queue in the user space (upper write queue as shown in figure 5.1). It processes the high priority messages, such as ioctl[2] requests, immediately and puts the remainder of the messages back on the queue, to be serviced by the service procedure. It takes care of flush handling of the upper queues. The device driver sends the M_FLUSH message to itself whenever it receives an unknown message type (error condition). On receiving M_FLUSH message (message type demanding the flushing/clearing of the queue) for the upper queue, Q921UWPUT calls flushq for that queue. This clears the queue of all the messages it was holding.

b) **Q921UWSRV** - It processes all the messages which were put back in the queue by the Q921UWPUT procedure and sends the messages downstream to the Q.921 from DLPI. In other words takes care of the flow control on the write side.

---

[2] ioctl system call performs variety of control functions on terminals, devices, sockets and streams.

c) **Q921LRPUT** - It receives the messages from the D-Channel of the ISDN card and processes all the high priority messages like IOCTL messages. It puts the M_DATA (message type carrying the data coming from the higher layer) and M_PROTO (message type carrying the control information for flow control) messages back on the queue to be serviced by the service procedure. It takes care of the flush handling of the lower queues. On receiving M_FLUSH for the lower queue, Q921LRPUT calls flushq for that queue.

d) **Q921LRSRV** - Takes care of the flow control on the read queue side. It sends M_DATA and M_PROTO messages up to the Q.921 module. The Q.921 module processes the message, and if the message is addressed to the Q.931 module, it passes on the message to the DLPI.

e) **Q921LWSRV** - This procedure is responsible for the flow control. It enables the upper write queue whenever the driver is in a state to receive messages from the user space (Q.931 module).

f) **Q921URSRV** - This procedure is also responsible for the flow control on the read side. It enables the lower read queue whenever the stream is in a state to receive messages from the ISDN card.

5.4 Implementation of ITU-T defined Q.921 Recommendations

The ITU-T defined Q.921 recommendations have been implemented using a modular approach [Suns96]. The major functions which implement the Q.921 recommendations are as follows :

a) dl_establish_request - This function is called when the Q.921 entity receives a DL_ESTABLISH_REQ from the Q.931 entity via the DLPI. If Q.921 module is in TEI (Terminal End Point Identifier) UNASSIGNED state, the function sends a TEI REQUEST (calls send_tei_req function) to the peer LAP-D entity in the switch and changes the state to AWAIT TEI. If the Q.921 module is in TEI ASSIGNED state, it calls the est_data_link function to initiate the data link establishment with the peer LAP-D entity, and changes the state of Q.921 to WAIT_ESTREL.

b) send_tei_req - This function first calls the function phys_activate for the physical activation of the ISDN card. On getting an acknowledgment for the physical activation it sends a request to the network side (switch) for the assignment of the TEI (Terminal End Point Identifier) in Unnumbered Information frame. The function fills the SAPI value in the header and fills MGT_TEI_REQUEST in the message type field. The function also puts the message on the D-Channel queue, in order to deliver it to the network (switch).

c) phys_activate - This function sends a physical activation request to the ISDN card (physical layer) using the M_IOCTL (an IOCTL message) message type and the command set to ISDN_PH_ACTIVATE_REQ.

d) phys_activate_ack - This function processes the acknowledgment received for the physical activation request of the ISDN card. It checks if the message type is

MIOCACK (an ACK for IOCTL message) and indicates whether the physical activation is successful or not.

e) **recv_mgt_msg** - This function processes the management messages coming from the network such as the message concerning TEI MANAGEMENT. It receives the management message which contains the TEI, which was requested by the LAP-D entity. The function extracts the TEI from the message and stores it in the data structures maintained by the Q.921 streams device driver and changes the state of the Q.921 module from the TEI_UNASSIGNED to TEI_ASSIGNED.

f) **est_data_link** - This function sends SABME(Set Asynchronous Balanced Mode Extended) command in an Unnumbered frame to the peer LAP-D entity to initiate the data link establishment between the two LAP-D entities. It starts the timer T200 (2 sec) in order to receive an unnumbered acknowledgment for the SABME from the peer LAP-D entity, within that time.

g) **send_uframe** - This function is used to send Unnumbered frames to the peer LAP-D entity. It prepares the header for the frame by filling the SAPI and TEI values in the header and fills in the command SABME /DISC (Disconnect)/DM (Disconnect Mode)/UA (Unnumbered Acknowledgment)/FRMR (Frame Reject) as per the requirements. The function puts the message on the D-Channel queue to deliver it to the peer LAP-D entity.

h) **recv_uframe** - This function caters to receiving various unnumbered frames such as SABME, Unnumbered Acknowledgment, Disconnect and Frame Reject. On receiving SABME, it sends Unnumbered Acknowledgment to the peer LAP-D entity after confirming that the Q.921 is in TEI ASSIGNED state. It initializes the

variables that keep track of the sequence numbers, and sends DL_ESTABLISH_IND to the Q.931 module via the DLPI module indicating the establishment of multiple frame state. It finally starts the timer T203. T203 is a 10 seconds timer and is a maximum time for which no frames are exchanged between the peer LAP-D entities. The state of Q.921 module is changed to ESTABLISHED TIMER STATE.

Receiving the Unnumbered Acknowledgment indicates that the SABME has been successfully received by the peer LAP-D entity. This function on receiving the UA frame initializes the counters that keep track of multiple frames. It starts the timer T203. Then it informs the Q.931 module via the DLPI that multiple frames state is ESTABLISHED TIMER using the DL_ESTABLISH_CON primitive.

On receiving the Disconnect frame, the function calls dl_release_req function that sends unnumbered acknowledgment to the peer LAP-D entity , stops the timer T200, clears the upper message queues and changes the state of the Q.921 module to TEI ASSIGNED. It sends a DL_RELEASE_IND to the Q.931 entity, indicating the release of data link connection.

On receiving the frame reject, the function calls for re-establishment of the data link with the peer LAP-D entity by calling est_data_link function.

i) **send_iframe** : This function is used to send an information frame to the peer LAP-D entity. The function puts the information frame back on the queue if the previous information frame has not been acknowledged. The function also prepares the header for the information frame by filling in the SAPI, TEI from the data structures maintained by Q.921 streams device driver and C/R (Command/Response)

bit. The function links the header with the data frame received from the Q.931 entity, starts the timer T200 and puts the frame on the D-Channel queue.

j) **receive_iframe** : This function receives the information frame from the peer. It sends the appropriate response to the peer LAP-D entity. If there is a sequence number mismatch, the function sends a Receive not Ready supervisory frame as a response, otherwise it sends a Receive Ready supervisory frame as a response.

k) **receive_sframe** : This function caters to receiving different types of supervisory frames from the peer LAP-D entity. If it receives Receive Ready frame, it resets a flag so that the LAP-D entity can send the next information frame to the peer LAP-D entity. If it receives the Receive not Ready information frame, it sets a flag which prohibits the LAP-D entity from sending the next information frame to peer LAP-D entity.

l) **send_sframe** : This function provides for sending the Receive Ready (RR), Receive not Ready (RNR) or Reject (REJ) supervisory frame to the peer LAP-D entity. It prepares the header for the message by filling up the SAPI and TEI values in the address field from the data structures maintained by the streams device driver and fills up the command (RR, RNR or REJ) as per the condition. It puts the message on the D-Channel queue to deliver it to the peer LAP-D entity.

m) **dlphi_to_q921** - This function is called when a message is received from the physical interface. It checks for the SAPI value within the message. If the SAPI value is SAPI_MGT indicating a management message, it calls recv_mgt_msg. If the SAPI value is SAPI_Q931_8 , the message is a call control message meant for the Q.931 entity. For the call control message, the function further checks the frame

type. If the frame type is Unnumbered frame, it calls recv_uframe function, if the frame type is information frame, it calls recv_iframe function and if the frame type is supervisory frame, it calls recv_sframe function.

5.5 Data Link Provider Interface and STREAMS.

The DLPI module has been implemented both in Q.921 module and the Q.931 module.The major functions of the DLPI as specified in [Steve96] can be described in the following points :

1) The Data Link Provider Interface specifies an interface to the services of Data Link Layer. This interface is the boundary between the network and Data Link Layers of the OSI model. The network layer entity is the Data Link Service (DLS) user; the Data Link Layer entity is the DLS provider.

2) The DLPI primitives are exchanged between the Data Link Service User and the Data Link Service Provider. The DLPI primitives are defined in terms of STREAMS messages.

3) There are three types of STREAMS messages that can be passed from one STREAMS module to another: M_DATA, M_PROTO, M_PCPROTO. M_DATA carries data within a stream. M_PROTO or M_PCPROTO carry both data and control information. Control information includes the attaching and binding the streams to a module, releasing a streams connection etc. The M_PCPROTO message has the same general use as an M_PROTO, but moves faster through a stream as it has a higher priority.

4) STREAMS provides the PUTMSG and GETMSG system calls that are used by the modules to send and receive STREAMS messages. The PUTMSG system call

provides a data buffer which is converted into an M_DATA message, and also provides a separate control buffer to be placed into an M_PROTO or M_PCPROTO block. The GETMSG system call is used by the module to accept the STREAMS messages. It can accept both the data and the control information.

5) The DLPI module is implemented both in the DLS user and DLS provider. The DLPI module in the DLS provider will translate the primitives of the DLS provider to DLPI primitives. These primitives are sent as STREAMS messages to the DLS user. The DLPI module in the DLS user will translate the DLPI primitives received from the DLS provider to the primitives that can be understood by the user.

6) There are four major modules that are supported by DLPI (figure 5.2)

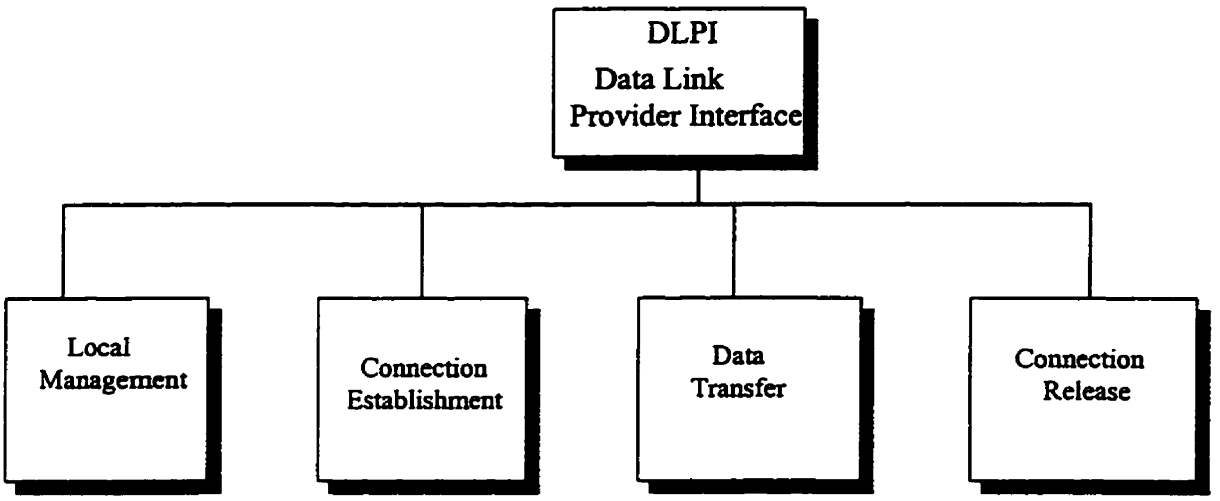


Figure 5.2: DLPI Modules

a) **Local Management** - This module takes care of initializing the STREAMS. It provides services such as attaching and binding. Attaching means providing a

physical point of attachment (PPA - relates to the file descriptor of the pair of queues that are used for message communication between the network and the data link layer) to the STREAMS within a module and Binding means associating the Data Link Service Access Point to STREAM within the module. These primitives are sent using the control buffer of M_PROTO message types supported by STREAMS.

b) **Connection Establishment** - This module establishes a data link connection between a local DLS user and a remote DLS user for the purpose of sending or receiving data. The user issues a DL_ESTABLISH_REQ which is mapped to the DLPI primitive DL_CONNECT_REQ. The provider issues DL_ESTABLISH_CONFIRM which is sent back as DL_CONNECT_CON as an acknowledgment. These primitives are sent using the control buffer of the M_PROTO message type.

c) **Connection Release** - It releases the data link connection between a local DLS user and remote DLS user. The user issues a DL_RELEASE_REQ, which will be sent to the DLS provider as DL_DISCONNECT_REQ. The provider will send back a DL_OK_ACK as an acknowledgment for the release request. These primitives are sent using the control buffer of M_PROTO message type.

d) **Data Transfer** - The DLS user sends the data using the data buffer of M_DATA message type. To receive data, it uses DL_DATA_IND primitive in the M_DATA message type. These primitives are not acknowledged by the DLS provider.

5.5.1 Implementation of DLPI in the Q.921 Module

The major functions that implement the Data Link Provider Interface in the Q.921 module [Suns96] are as follows:

a) dlpi_to_q921 - This function receives the messages from the Q.931 module and calls appropriate DLPI functions or the Q.921 functions based on the message type or the DLPI primitive within the message. If it is M_PROTO type message, the function further checks the DLPI primitive which indicates the control information. Here are all the possibilities that occur due to the check process :

- If DL_CONNECT_REQ is the DLPI primitive, it calls dl_establish_req function the functionality of which has been explained in the Q.921 module.
- If the DLPI primitive is DL_DISCONNECT_REQ, it calls dl_release_req function the functionality of which has been explained in the Q.921 module.
- If DL_ATTACH_REQ is the DLPI primitive, it calls dl_attach_req function which confirms that the Q.921 is in DL_UNATTACHED state and provides a physical point of attachment by updating the data structures maintained by the Q.921 device driver, changes the Q.921 state to DL_UNBOUND and sends a DL_OK_ACK primitive to the DLS user using M_PCPROTO type message.
- If DL_BIND_REQ is the DLPI primitive, it calls dl_bind_req function which binds the service access point to the data link user. It changes the state of Q.921 to

TEI_ASSIGNED and sends the DL_BIND_ACK primitive using the M_PCPROTO type message to the DLS user as an acknowledgment.

- If DL_DETACH_REQ is the DLPI primitive, it calls dl_detach_req function which detaches the physical point of attachment of the DLS user by initializing the data structures maintained by the Q.921 device driver, stops all the timers that are possibly running and changes the state of Q.921 module to DL_UNATTACHED.
- If DL_UNBIND_REQ is the DLPI primitive, it calls dl_unbind_req function which unbinds the DLS user from the service access point and changes the state of Q.921 module to DL_UNBOUND.

b) dlpi_from_q921 - This function receives the messages coming from the Q.921 module, meant for the Q.931 module via the DLPI. The function checks the Q.921 primitive in the message and translates it into a DLPI primitive and calls the appropriate function that sends the primitive along with any data to the Q.931 module. Here are all the possibilities that occur due to the check process.

- If the Q.921 primitive is DL_ESTABLISH_CON, it calls the dl_connect_con function which sends DL_CONNECT_CON (DLPI) primitive to the Q.931 module using the M_PROTO type message. This primitive indicates to the Q.931 module that a data link has been established with the remote end.
- If the Q.921 primitive is DL_RELEASE_CON or DL_RELEASE_IND, it calls dl_disconnect_ind function which sends a DL_DISCONNECT_IND (DLPI) primitive to the Q.931 module using M_PROTO type message. This primitive

indicates to the Q.931 module that the data link has been disconnected with the remote end.

- If the Q.921 primitive is DL_DATA_IND, it calls dl_data_ind function which sends the message up to the Q.931 module using M_DATA type message. This function is used to route data from Q.921 module to Q.931 module via the DLPI.

5.5.2 Implementation of DLPI in Q.931 Module

The major functions that implement DLPI in the Q.931 module are as follows :

a) **dlpi_attach_req** - This function sends a request for a physical point of attachment to the Q.921 module. It sends a DL_ATTACH_REQ primitive to the Q.921 module, in the control buffer using the PUTMSG system call of the STREAMS.

b) **dlpi_bind_req** - This function requests binding of Service Access Point of Q.931 module to the Q.921 module. It sends a DL_BIND_REQ primitive to the Q.921 module, in the control buffer using the PUTMSG system call of the STREAMS.

c) **dlpi_detach_req** - This function sends a request to the Q.921 module to detach the physical point of attachment. It sends DL_DETACH_REQ primitive to the Q.921 module in the control buffer using the PUTMSG system call of the STREAMS.

d) **dlpi_unbind_req** - This function sends a request to the Q.921 module to unbind the Service Access Point of the Q.931 module. It sends a DL_UNBIND_REQ primitive to the Q.921 module in the control buffer using the PUTMSG system call of the STREAMS.

e) **dlpi_open** - This function opens a stream to the Q.921 pseudo multiplexing device

the dl_attach_req function which requests a physical point of attachment for the stream (file descriptor) to the Q.921 module. On getting a positive acknowledgment for this request, the function calls the dl_bind_req function which binds the stream (file descriptor) with the Service Access Point within the Q.921 module.

f) **dlpi_close** - This function closes the stream to the Q.921 module by using the close system call on the file descriptor of the stream. Before closing the stream it calls the dl_unbind_req function that sends a request to the Q.921 module to unbind the stream with the service access point, and then calls the dl_detach_req function which requests the Q.921 module to detach the physical point of attachment of the stream.

g) **dlpi_put** - This function gets the message from the Q.931 module. It translates the Q.931 primitives to DLPI primitives, puts the message in either the control buffer or the data buffer and sends the message down to the Q.921 module using the PUTMSG system call. Depending on the Q.931 primitives three cases arise. If the Q.931 primitive is DL_ESTABLISH_REQ, the dlpi_put function calls the dlpi_connect_req function which sends a DL_CONNECT_REQ (DLPI) primitive to the Q.921 module indicating to the Q.921 module to establish a data link connection with the peer LAP-D entity. If the Q.931 primitive is DL_RELEASE_REQ, the dlpi_put function calls the dlpi_disconnect_req function which sends a DL_DISCONNECT_REQ (DLPI) primitive to the Q.921 module, indicating to the Q.921 module to release the established data Link connection with the peer LAP-D entity. If the Q.931 primitive is

DL_DATA_REQ, it puts the Q.931 message in a data buffer and sends it to the Q.921 module using the PUTMSG system call.

h) dlpi_get - This function gets the message from the Q.921 module using the GETMSG system call. It translates the DLPI primitives received in the message to the Q.931 primitives, in order that they can be understood by the Q.931 module. If the DLPI primitive received is DL_CONNECT_CON, it translates it to the DL_ESTABLISH_CON (Q.931) primitive, which sends a confirmation to the Q.931 module that a data link has been established with the peer LAP-D entity. If the DLPI primitive received is DL_DISCONNECT_IND , it translates it to a DL_RELEASE_IND (Q.931) primitive, which informs the Q.931 module that the data link has been disconnected with the peer LAP-D entity. If the DLPI primitive received is DL_DATA_IND, it extracts the data from the data buffer attached to the message and stores the data in the data structures of Q.931 module.

5.6 Implementation of ITU-T Defined Q.931 Recommendations

The major functions that implement Q.931 recommendations are as follows :

a) q931_call - This function handles the calls made by the local user. It also receives different incoming Call Control messages and depending on the message type and the previous state of the Q.931 entity, it changes the state of the Q.931 entity. Initially the Q.931 entity is in NULL STATE.

When a local user makes a call, this function prepares the Call Control SETUP message. It fills the protocol discriminator and call reference field in the message, sets

changes the Q.931 state to CALL INITIATED state. It finally passes this call record to q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI.

The handling of incoming Call Control messages is as follows.

- If the incoming message type is CALL PROCEEDING , and the current Q.931 state is CALL INITIATED, the function changes the Q.931 state to OUTGOING CALL PROCEEDING.
- If the incoming message type is ALERTING and the Q.931 state is OUTGOING CALL PROCEEDING, the function changes the Q.931 state to CALL_DELIVERED.
- If the incoming message type is CONNECT and the Q.931 state is CALL_DELIVERED, the function changes the Q.931 state to ACTIVE. On reaching the ACTIVE state the function prepares a Q.931 message with message type Q931_CONNACK (CONNECT ACKNOWLEDGE), fills the protocol discriminator and call reference field, and sets the DLPI primitive to DL_DATA_REQ. It passes this message record to the q931_dlpimsg_write function (described in detail later in the chapter) that fills the remainder of the fields and sends it to the Q.921 entity via DLPI.

b) q931_receive - This function receives incoming calls from the network. It receives the Call Control messages for the incoming calls. It checks the message type and the present state of the Q.931 entity and takes action accordingly:

- If the incoming message type is SETUP and the Q.931 state is NULL, this function prepares the Q.931 reply message with message type CALL PROCEEDING and changes the Q.931 state to the CALL PRESENT state. It fills the protocol discriminator, call reference, and the DLPI primitive as DL_DATA_REQ. It passes this message record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI. After sending the CALL PROCEEDING message, it changes the Q.931 state to INCOMING CALL PROCEEDING and prepares another Q.931 message with message type ALERTING. It passes this message record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI. After sending the ALERTING message, it changes the Q.931 state to CONNECT REQ and prepares another Q.931 message with message type CONNECT. It passes this message record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI.
- The Q.931 entity waits for a CONNECT ACK message from the network. On receiving the Q.931 message type CONNACK, the function checks if the Q.931 state is CONNECT REQ and changes the Q.931 state to the ACTIVE state.

c) insert_info - This function fills the information field of the Q.931 call control message. The q931_call function fills the protocol discriminator, call reference and the message type field. The information field is composed of a linked list of records. The first byte of each record indicates the type of the information stored in the record (e.g.

called party number, calling party number), the second byte indicates the length of the information and the remaining bytes are the actual information. The last field is the pointer to the next record. For a new call , the first record contains the calling party number, and the second record contains the called party number. Some of the records can contain User-to-User Information.

d) message_encode - This function converts the complete call record into a stream of bytes, so that they can be inserted into the data buffer of M_DATA message type of DLPI. It allocates enough memory for the data buffer and stores each field of the call record in the consecutive locations of the data buffer in a set pattern, so that the information about each field can extracted by the peer Q.931 entity, by following that pattern.

e) message_decode - This function extracts the stream of bytes from the M_DATA message type of the DLPI and interprets the stream of bytes using a set pattern, as set by the message encode function. From this stream of bytes, it fills the call record i.e. the protocol discriminator, call reference value, the Q.931 message type and the information field. Since the information field is implemented as a linked list of records with each record storing different information, this function prepares the linked list from the stream of bytes. The Q.931 entity only refers the call record to change its state and to take further action.

f) q931_dlpimsg_write - This function gets the partially filled call record from the q931_call function. It calls the insert_info function, which fills the information field of the call record. Then it calls the message_encode function that converts the entire call record into streams of bytes. Finally it calls the dlpi_put function that stores these

streams of bytes into the data buffer of M_DATA message type of DLPI and uses the PUTMSG system call to deliver it to the DLPI of Q.921 module.

g) **q931_disconnect** - This function handles various cases that can lead to call disconnection and release. The various cases are as follows :

- Once Q.931 has sent CONNECT to the network it comes to the CALL RECEIVED state. If it does not receive a CONNECT ACKNOWLEDGE from the network within the time set in the timer (T313), the Q.931 entity initiates call clearing. In this state the function prepares a Q.931 message with message type DISCONNECT, and fills the call reference value and protocol discriminator field. It sets the DLPI primitive to DL_DATA_REQ. It passes this call record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI. It changes the Q.931 state to DISCONNECT REQUEST.
- If the user receives RELEASE from the network, and the function finds Q.931 in the CALL INITIATED state, it prepares a Q.931 message with message type RELEASE COMPLETE. It fills in the protocol discriminator field and the call reference value and sets the DLPI primitive to DL_DATA_REQ. It passes this call record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI. It changes the Q.931 state to the NULL STATE.
- If the Q.931 entity is in the ACTIVE state and the user disconnects, the function prepares the Q.931 message with message type DISCONNECT. It fills the protocol discriminator and call reference value fields and sets the DLPI message type to DL_DATA_REQ. It passes this call record to the q931_dlpimsg_write function that

fills the remainder of the fields and sends it to the Q.921 entity via DLPI. It changes the Q.931 state to the DISCONNECT REQUEST state.

- If the Q.931 entity is in the DISCONNECT REQUEST state and it receives a RELEASE message from the network in response to the DISCONNECT message, the function prepares the Q.931 message with message type RELEASE COMPLETE. It fills the protocol discriminator and call reference value fields and sets the DLPI message type DL_DATA_REQ. It passes this call record to the q931_dlpimsg_write function that fills the remainder of the fields and sends it to the Q.921 entity via DLPI. It changes the Q.931 state to NULL.

CHAPTER 6

The Design and Implementation of the Framework for the Real-Time Applications at Home and Small Businesses

6.1 Description of the Framework

We have developed a framework for real-time applications based on the Sunshine's protocol model described in the previous chapter. In the Sunshine's protocol model, the lower queues of pseudo multiplexing device driver were linked to the D-Channel of the ISDN interface card. The ISDN interface card is connected to the ISDN line (2B+D basic rate interface). The ISDN line gives connectivity to the ISDN switch. The ISDN switch itself has its LAP-D entity and Q.931 entity and it switches the D and B Channel traffic to appropriate D and B channels depending on the ISDN number of the Calling and the Called party. The equipment at the user interface connected to the ISDN line has to be assigned a TEI (Terminal End Point Identifier) by the switch.

In our implementation model we do not have connectivity to the ISDN switch. We have developed a model where we show how the peer LAP-D and peer Q.931 entities interact to support the real-time applications (Figure 6.1). We have implemented a switch simulator and have removed the interface of the Q.921 pseudo multiplexing device driver with the ISDN card. For testing purposes we have put the protocol stack of the calling user and the remote user on the same workstation. We have linked the lower queues of the Q.921 pseudo multiplexing device driver of each user via the switch simulator. These two queues carry the D-Channel traffic between the peer LAP-D entities. The lower

write queue sends the D-Channel messages to the peer LAP-D entity, and the lower read queue receives the messages from the peer LAP-D entity. Thus the write queue of one entity is linked with the read queue of the other, in order to allow full two way communication.

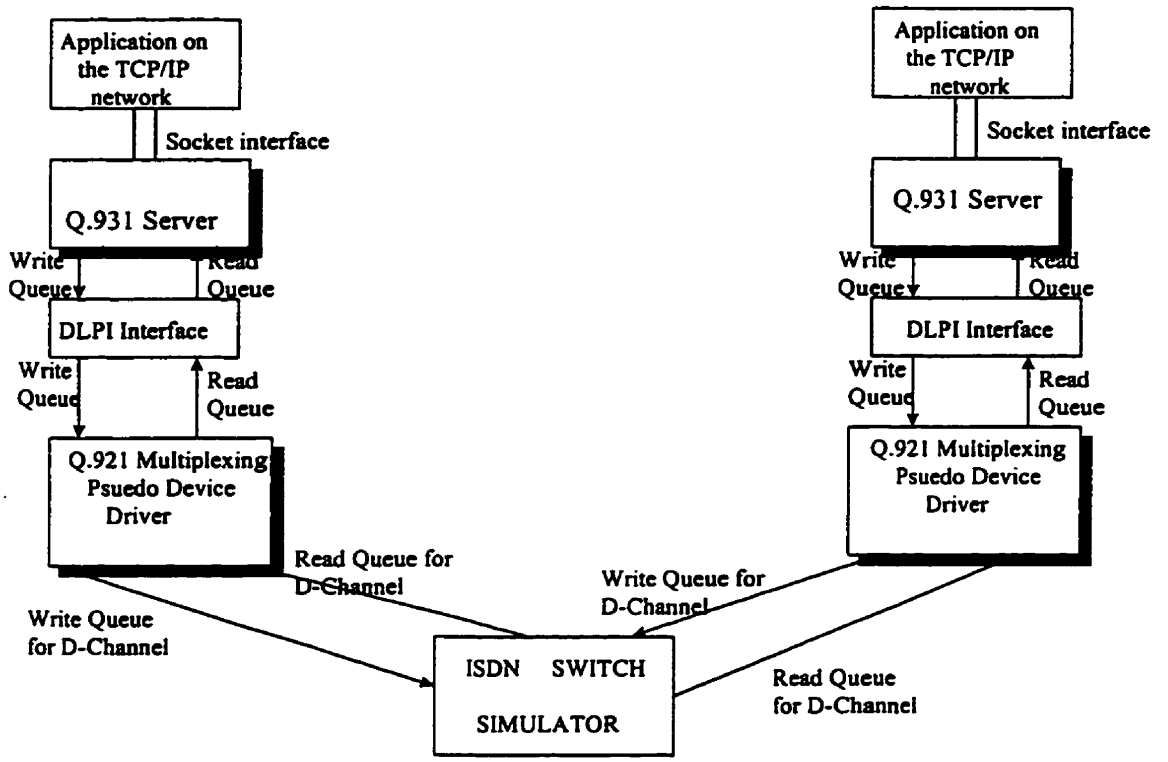


Figure 6.1 Framework for real-time applications on Sun Sparc Station in Solaris Environment

6.2 The Message Flow between the Peer Protocol Entities

The Q.921 entity has been implemented as a pseudo multiplexing device driver: therefore it is installed in the kernel. As per our framework, we have two LAP-D entities residing in the kernel. The Q.931 entities of both the protocol stacks issue an OPEN system call for their respective Q.921 pseudo multiplexing device drivers. As a result, each Q.931 entity gets a pair of queues (file descriptor to the queues) to communicate

with the LAP-D entity residing in the kernel. The read queue and the write queue constitute the pair and are called as upper queues with reference to the Q.921 pseudo multiplexing device driver. The DLPI module, which is implemented in both the Q.921 and Q.931 module, attaches and binds the file descriptor of these queues with a service access point in the LAP-D entity. The write queue carries all the messages from the Q.931 module down to the LAP-D entity via DLPI. The read queue carries all the messages from the LAP-D entity up to the Q.931 entity via the DLPI.

The function that links the lower queues of the Q.921 pseudo multiplexing device driver issues an OPEN system call for each of the Q.921 pseudo multiplexing device drivers. This opens another pair of queues (called lower pair of queues) from each of the Q.921 device drivers and returns file descriptors to each of the queue pairs. These queue pairs are referred to as lower queue pairs for each of the Q.921 device drivers. These lower queue pairs of each Q.921 device driver have to be linked, so that any message coming from one LAP-D entity placed on the lower write queue is delivered to the peer LAP-D entity of the peer pseudo device driver via the lower read queue. In order to link the lower queues of the two device drivers, the function calls an ioctl (system call described in Chapter 5) with an I_LINK parameter and the two file descriptors as parameters. I_LINK links two streams represented by two different file descriptors in order to allow message communication between the different streams. This links the write queue of one device driver with the read queue of the other and vice versa.

6.3 Switch Simulator

In SunShine's model, prior to initiating any communication with the peer entity, the LAP-D entity has to ensure that the physical layer is activated. The LAP-D entity sends

a physical layer activation request to the physical layer (i.e. ISDN card) and waits for an acknowledgment. On receiving the acknowledgment, the LAP-D entity enters the TEI UNASSIGNED state. In our model, we do not have an ISDN card, so we have to provide this message interface to the LAP-D entity from outside.

Initially the LAP-D entity is in the TEI UNASSIGNED state. As stated in Chapter 2, the service access point identifier (SAPI) identifies a layer 3 user of LAP-D and thus corresponds to a layer 3 protocol entity within a user device. The SAPI values are unique within a TEI (Terminal End Point Identifier). That is for given TEI, there is a unique layer 3 entity for a given SAPI. This TEI is assigned by the switch. The LAP-D entity sends a TEI request to the switch and switch responds with a unique TEI for the layer 3 entity, and the LAP-D entity comes to a TEI ASSIGNED state. It is from this state that the LAP-D can communicate with the peer LAP-D entity. In our model, we do not have connectivity to the switch, so we have to provide the TEI ASSIGNMENT message to the LAP-D entity from the external source, so that the LAP-D entity enters the TEI ASSIGNED state.

In order to fulfill the above two requirements, we need a switch simulator. The switch simulator will be an entity which would recognize the messages meant for the switch and the ISDN interface card. It would give the appropriate response back to the LAP-D entity so that it enters a state where it can communicate with the peer LAP-D entity. The functionality of the switch simulator has been built within the pseudo multiplexing device driver. For any message coming from its lower read queue, the device driver checks if the message is meant for the switch. If so, it sends an appropriate response back to the LAP-D entity from which the message was coming. Thus the LAP-D entity

can change its state and enter a state where it can communicate with the peer LAP-D entity.

6.4 Design and Implementation of the TeleMonitoring Application

In the Sunshine's model the Q.931 entity is implemented as a server on a TCP/IP network. The server opens a port and is listening for a connection. The TeleMonitoring application can reside anywhere on the TCP/IP network as long as it knows the server's host address and the port number on which it is listening. The TeleMonitoring application communicates with the Q.931 module via a socket interface. The reason we are having a connection between the application and the Q.931 entity via the TCP/IP network is that the Q.931 entity is implemented as a server in the Sunshine's model and it does not really affect our testing. The application can also be directly integrated with the Q.931 entity without using the socket connection. The Figure 6.2 shows how the application at the Data Monitoring End and the application at the Data Generating End fits in our testing framework for real-time applications. Both the applications can reside anywhere on the TCP/IP network and can connect to their respective servers using the specified port on which the server is listening.

The application and the Q.931 server both call the socket system call to create a new socket that can be used for communication between the two. The parameter in this case indicate TCP/IP as the protocol and STREAM socket as the socket type. The Q.931 server calls the bind system call to specify the local endpoint address for a socket. The Q.931 server then calls the listen system call, in order to listen for a connection from the application. The Q.931 server calls the accept system call after the listen system call in order to receive a connection request from the application. The applications at both ends

call the connect system call, that allows the applications to have a socket connection to their respective Q.931 servers.

The Q.931 server at the Data Monitoring end calls the read system call to accept the ISDN number of the called party and the request string, which will be transmitted over the ISDN connection in order to request data from the Data Generating end. Once the Q.931 server receives the data from the Data Generating end, the Application at the Data Monitoring end calls the read system call to read the data from the server. The Application at the Data Generating end calls the read system call, to receive the Data request from its Q.931 server, which the Q.931 server has received from the Q.931 server at the Data Monitoring end. The Q.931 server at the Data Generating end calls the read system call to receive the Data from the Application, in order to transmit it over the ISDN connection to the Data Monitoring end.

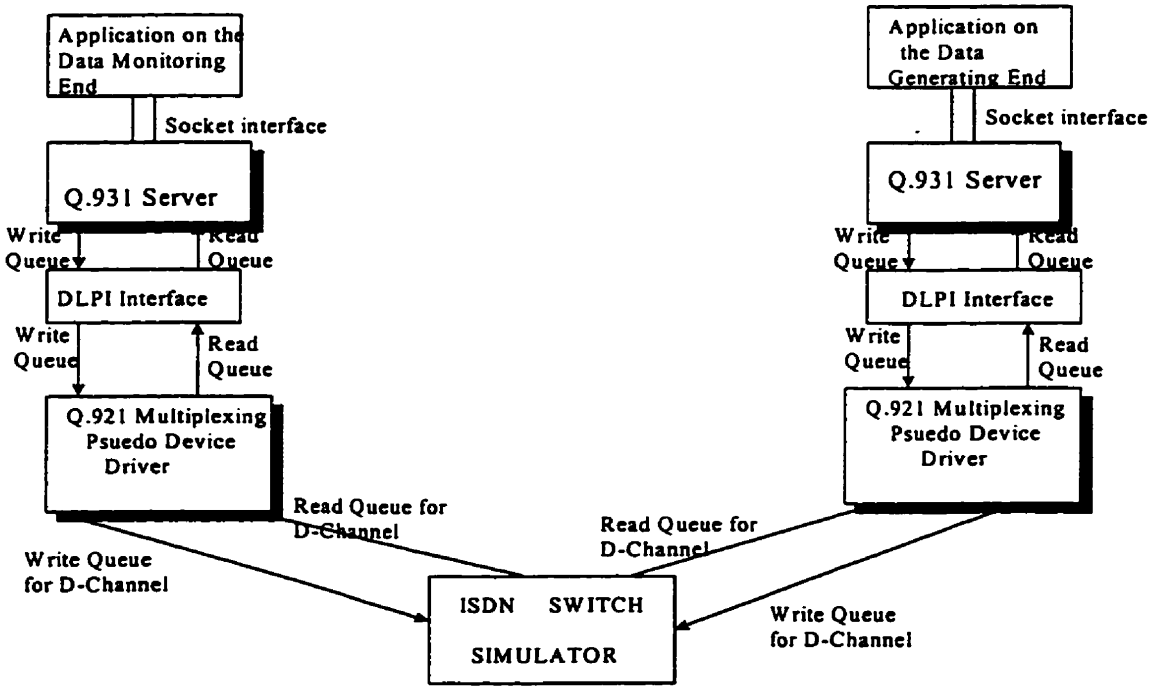


Figure 6.2: Testing Framework for TeleMonitoring Application

The Application at the Data Monitoring end uses the write system call to send the called party number and the request string to its Q.931 server, so that it can transmit that over the ISDN connection. The Q.931 server at the Data Generating end calls write system call, to transfer the Data Request to the Application, which the server received from its peer server. The Application at the Data Generating end calls the write system call to send the generated data to the Q.931 server, so that it can transmit over the ISDN connection. The Q.931 server at the Data Monitoring end calls the write system call, in order to send the data to the Application, that it has received over the ISDN connection from the Q.931 server at the Data Generating end.

6.4.1 Message flow over the ISDN protocol stacks for TeleMonitoring Application

The following sequence of steps explain the functionality of the TeleMonitoring application, using our model explained before. The detailed functionality of all the functions that have been referred to in the following sequence of steps is explained in Chapter 5.

1) The applications at the Data Monitoring end (DME) and at the Data Generating End (DGE) connect to their respective Q.931 servers via the socket connection. The application sends a message containing the ISDN number for the Q.931 server at the DGE. The message also includes a request string to request certain data from the Data Generating End, that has to be delivered to the Q.931 server at DME.

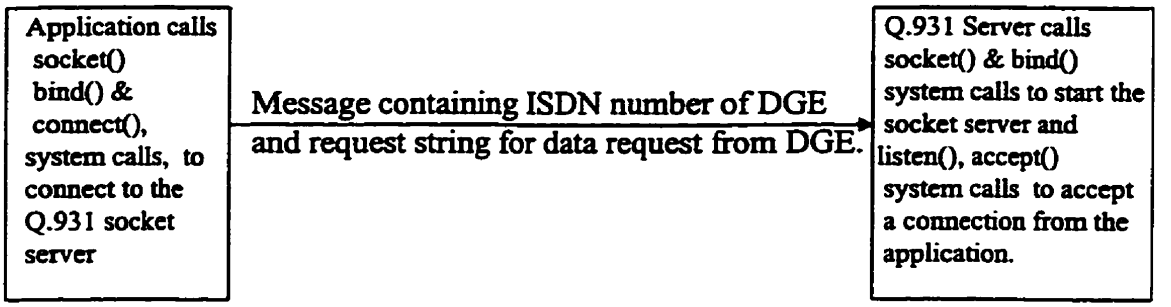


Figure 6.3: Step 1, The activities at the Data Monitoring End

2) The Q.931 server, on receiving the request from the application, initiates the ISDN connection establishment with the DGE. The Q.931 entity opens a streams connection with the Q.921 pseudo multiplexing device driver using the open system call, and it gets a pair of queues (read & write) to communicate with the Q.921 entity.

3) The Q.931 entity attaches and binds the file descriptor of those queues with the Q.921 module using the DLPI interface. The DLPI module in the Q.931 entity calls dlpi_attach_req and dlpi_bind_req (described in chapter 5) functions to send the attach and bind request respectively to the DLPI module in the Q.921 entity. The DLPI module in the Q.921 module on receiving the attach and bind requests calls the dl_attach_req and dl_bind_req functions that allows the Q.931 entity to have a service access point within the Q.921 entity.

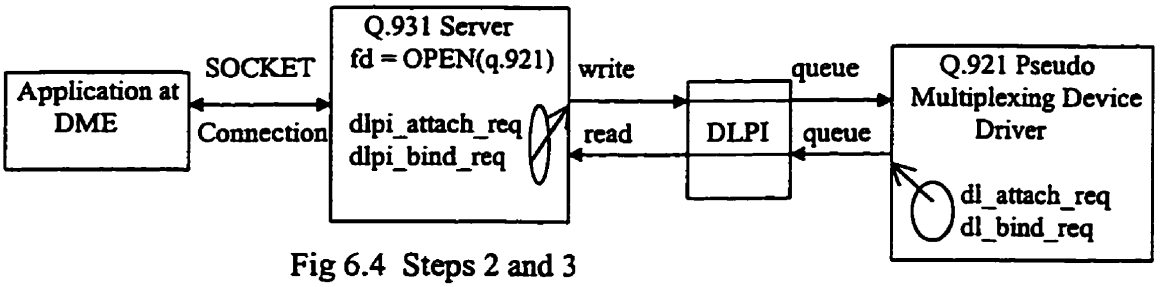


Fig 6.4 Steps 2 and 3

4) The Q.931 entity sends a DL_ESTABLISH_REQ to the Q.921 entity through the DLPI interface. The DLPI module translates the DL_ESTABLISH_REQ primitive into the DLPI primitive called DL_CONNECT_REQ and forwards it to the Q.921 module using the PUTMSG system call.

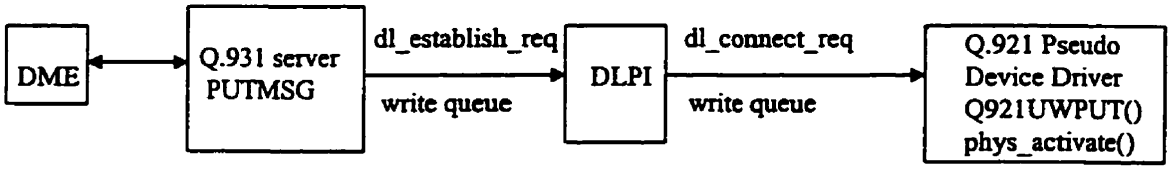


Figure 6.5 Step 4

5) The Q.921 pseudo device driver receives the DL_CONNECT_REQ through its Q921UWPUT procedure and sends a physical activation request to the physical layer (ISDN interface card) by calling phys_activate function. The phys_activate function puts the physical activation request message on the lower write queue of the device driver. In our model we do not have an ISDN interface card, and the lower write queue of one device driver is linked with the lower read queue of the other. Therefore the physical activation request message goes to the peer device driver via the lower read queue of that device driver. The Q921LRPUT procedure of the device driver has been modified to respond to the physical activation request message, and it responds with a positive acknowledgment by calling phys_activate_ack function. The phys_activate_ack function puts the positive acknowledgment message on its lower write queue using the QREPLY system call, and this message reaches the requesting driver via its lower read queue. The QREPLY system call sends the message back on the peer queue i.e. if the message was received on the read queue (write queue), the reply is put back on the write queue (read queue), in order to deliver the reply to the

entity that had sent the message. The Q.921 entity on getting the positive acknowledgment changes its state to TEI UNASSIGNED state.

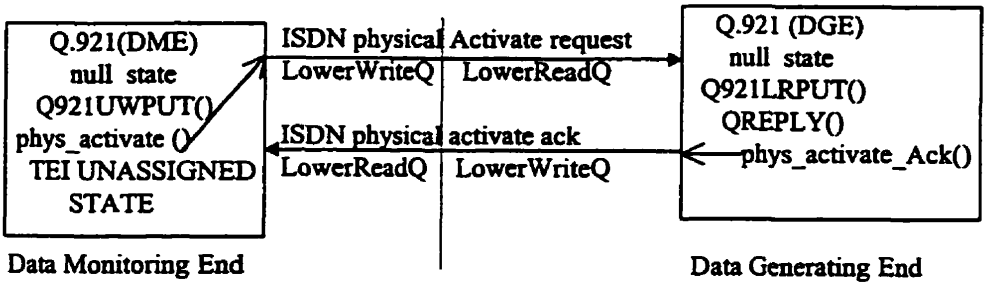


Figure 6.6 Step 5

6) Next the Q.921 pseudo device driver sends a TEI (Terminal End Point) assignment request to the switch. It puts the TEI request message on its lower write queue. A unique TEI has to be assigned to the Q.931 entity, that has a physical point of attachment in the Q.921 entity. The lower write queue of the pseudo device driver is linked with the lower read queue of the peer device driver. The TEI request message goes to the peer device driver. Each device driver has been modified to understand the messages meant for the switch and send back an appropriate reply. The peer device driver on receiving the TEI request, generates a TEI and puts the TEI message on its lower write queue. The write queue of this driver is linked with the read queue of the requesting driver. The message containing the TEI is received by the Q921LRPUT procedure of the requesting Q.921 device driver from its lower read queue. The Q.921 entity changes its state to TEI ASSIGNED state.

7) The Q.921 entity now initiates the data link establishment with the peer LAP-D entity. It calls the send_uframe function which puts the unnumbered message with a SABME command on the lower write queue of the device driver. This message is delivered to the peer Q.921 entity via its lower read queue. The Q921LRPUT procedure on receiving the unnumbered frame, calls the recv_uframe function. The recv_uframe function on finding the command to be SABME, prepares an unnumbered acknowledgment message and puts it on the lower write queue of the device driver, to send it to the peer LAP-D entity. The function initializes the variables that keep track of the sequence numbers, and sends DL_ESTABLISH_IND to the Q.931 entity (DGE) via DLPI. It indicates the establishment of multiple frame state to the Q.931 entity (DGE). It starts the timer T203 (10 SEC -Max time with no frames exchanged). The state of the Q.921 module is changed to ESTABLISHED TIMER state.

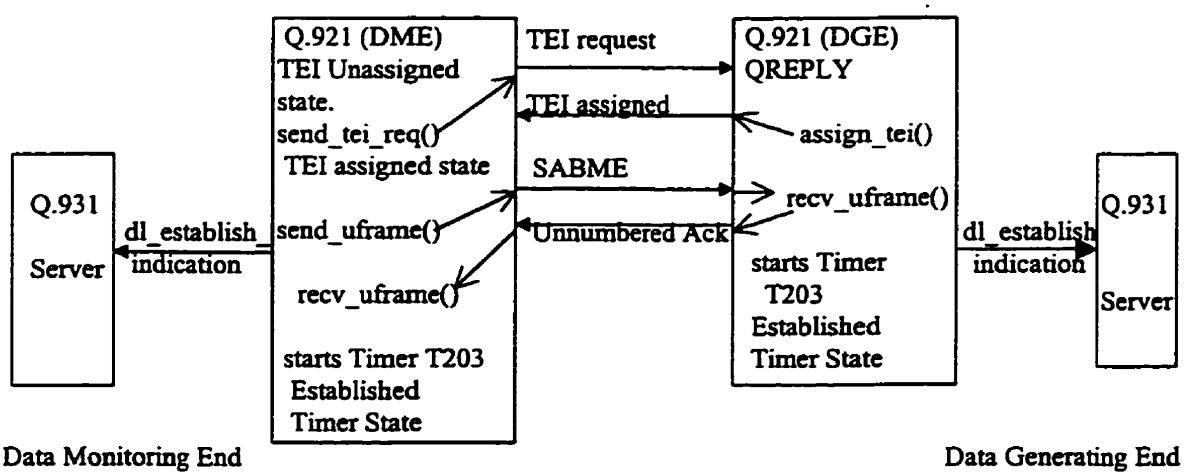


Figure 6.7 Steps 6, 7 and 8

8) The Q.921 entity (DME) that had sent SABME, on receiving the Unnumbered Acknowledgment (UA frame) from the peer LAP-D entity calls the recv_uframe function. This function on receiving the UA frame, initializes the counters that keep track of the multiple frames. It starts the timer T203. It informs the Q.931 entity (DME) via the DLPI that the multiple frame state is ESTABLISHED TIMER using the DL_CONNECT_CON primitive.

9) The dlpi_get function of the DLPI module in Q.931 entity (DME), translates the DL_CONNECT_CON primitive to DL_ESTABLISH_CON primitive and delivers it to the Q.931 entity. The Q.931 entity on receiving this primitive, enters a state where it can send call control messages to the peer Q.931 entity.

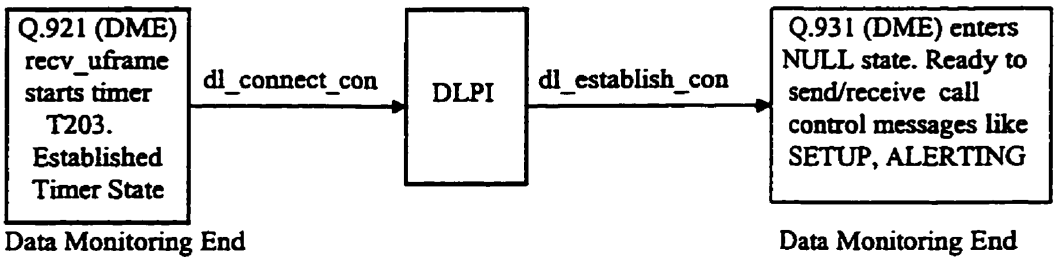


Figure 6.8 Step 9

10) The Q.931 entity (DME) is in the NULL state. It calls the q931_call function that prepares the call control SETUP message to establish a call between the Data Monitoring end and the Data Generating end. The function fills the protocol discriminator and the call reference field in the message, sets the message type field to SETUP and sets the DLPI primitive DL_DATA_REQ. It changes the Q.931 state

to the CALL INITIATED state. It finally passes this call record to the q931_dlpimsg_write function.

11) The q931_dlpimsg_write function calls the insert_info function that fills the Calling and Called party number in the message. The Called party number was sent by the Data Monitoring Application. The insert_info function also fills the request string in the User-to-User information field of the message . The request string was sent by the Data Monitoring application in order to request the data from the Data Generating end. The q931_dlpimsg_write function calls the message_decode function that converts the entire call record into stream of bytes. Finally the q931_dlpimsg_write function calls the dlpi_put function that stores the stream of bytes into the data buffer of M_DATA message type of DLPI and uses the PUTMSG system call to deliver it to the DLPI of Q.921 module.

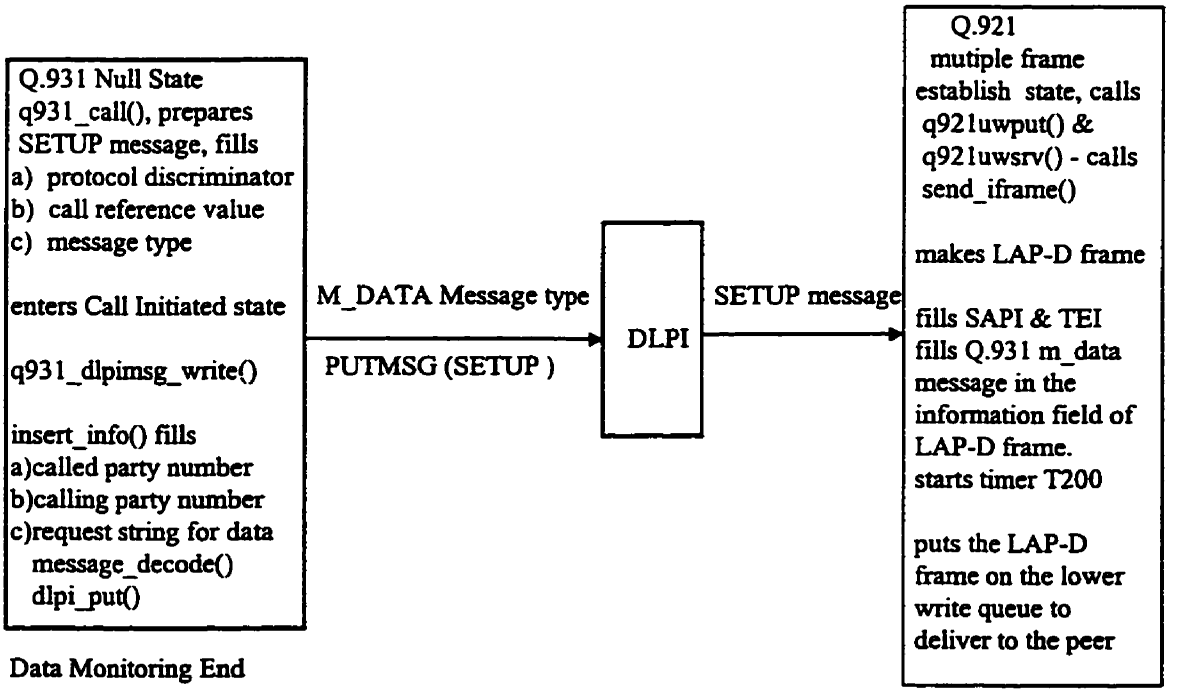


Figure 6.9 Steps 10, 11 and 12

12) The Q.921 pseudo device driver receives the M_DATA message in its Q921UWPUT procedure and puts it back on the queue to be serviced by the Q921UWSRV procedure. This procedure calls the send_iframe function. The send_iframe method prepares a LAP-D information frame. It fills the SAPI and TEI values in the address field of the frame, sets the command bit and inserts the Q.931 databuf information in the information field of the frame. It starts the T200 (1 SEC to wait for an ack) timer and sends the LAP-D frame on the D-Channel queue, which is the lower write queue of the device driver.

13) The peer Q.921 pseudo device driver receives the LAP-D information frame from the lower read queue, through the Q921LRPUT procedure. This procedure calls the dlphi_to_q921 function as the message is coming from the lower read queue. The function checks the SAPI value in the message and finds it to be SAPI_Q931_8 (call processing message). It checks the frame type of the message and on finding it to be an information frame, calls the recv_iframe function. The recv_iframe function matches the sequence number of the Information frame received, updates the counters that keep track of the sequence numbers and calls the send_sframe function to send back the receive ready acknowledgment to the peer LAP-D entity. The send_sframe function fills the SAPI and the TEI values in the address field of the message and RR (Receive Ready) in the command field. It puts the message on the lower write queue (D-Channel queue) to deliver it to the peer LAP-D entity, that had sent the information frame. The recv_iframe function also calls dlpi_from_q921 function to send the Q.931 part of the message to the Q.931 entity.

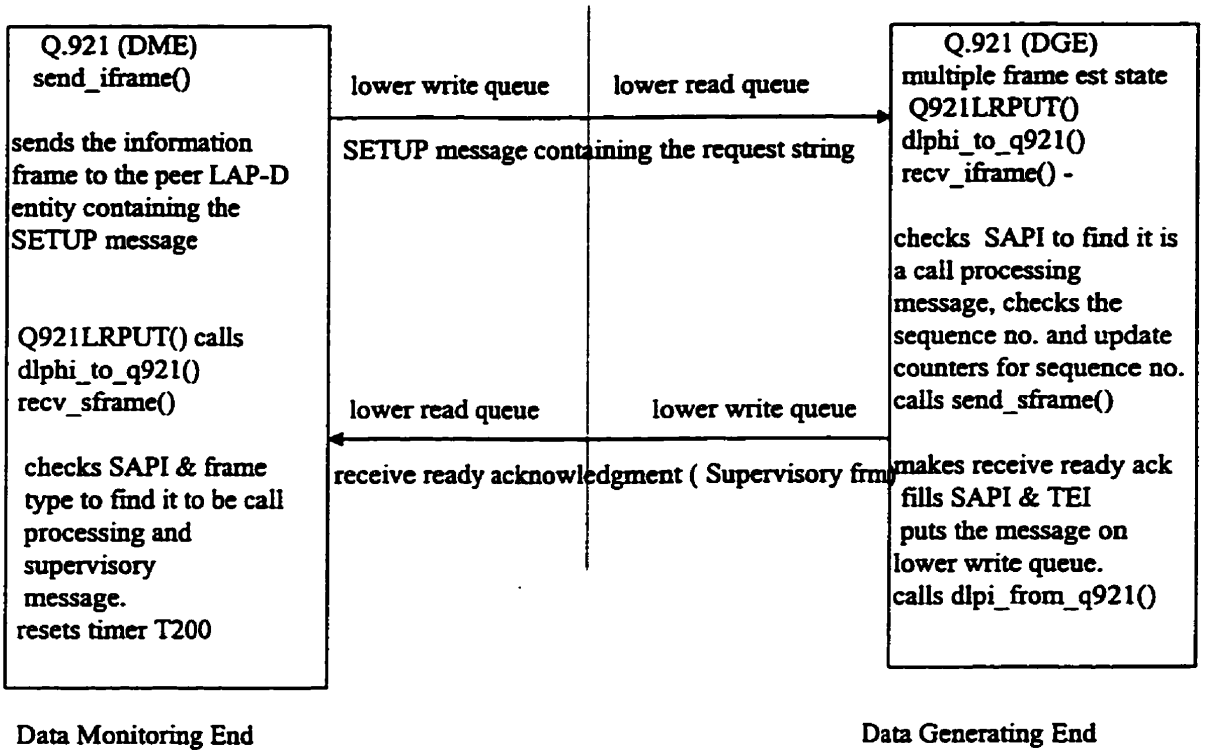


Figure 6.10 Steps 13 & 14

14) The Q.921 entity (DME) that had sent the SETUP message receives the Receive Ready acknowledgment from its lower read queue via the Q921LRPUT message. This procedure calls the dlphi_to_q921 function as the message is coming from the lower read queue. The function checks the SAPI value in the message and finds it to be SAPI_Q931_8 (call processing message). It checks the frame type of the message and on finding it to be a supervisory frame, calls the recv_sframe function. The recv_sframe function on finding the Receive Ready acknowledgment resets a flag so that the LAP-D entity can send the next information frame to the peer LAP-D entity. The function also resets the T200 timer, which was started by the LAP-D entity to wait for a Receive Ready acknowledgment.

15) The dlpi_from_q921 function calls the dl_data_ind function of the DLPI which sends the Q.931 part of the LAP-D frame received, up to the Q.931 module in M_DATA

type message, using the PUTNEXT system call. The primitive used is DL_DATA_IND.

16) The DLPI module implemented in the Q.931 entity (DGE) calls the dlpi_get function to receive the message coming from the DLPI of the Q921 module. The dlpi_get function uses the GETMSG system call to extract the message from the queue. The function on finding the DLPI primitive received to be DL_DATA_IND, extracts the data from the data buffer attached to the message and stores the data in the data structures of the Q.931 module. It calls the q931_receive function to handle the incoming call.

17) The Q.931 entity (DGE) is in NULL state. The q931_receive function finds the message type of the incoming message to be SETUP. It prepares the Q.931 reply message with the message type as CALL PROCEEDING and changes the Q.931 state to CALL PRESENT. It fills the protocol discriminator field, call reference value and the DLPI primitive as DL_DATA_REQ. It passes this call record to the q931_dlpimsg_write function.

18) The q931_dlpimsg_write function (DGE) calls the insert_info function that fills the Calling and Called party number in the message. The Calling party number comes in the SETUP message. The insert_info function also fills the requested data from the Data Generating Application in the User-to-User information field of the message . This data was sent by the Data Generating application and was requested by the Data Monitoring end in the SETUP message.

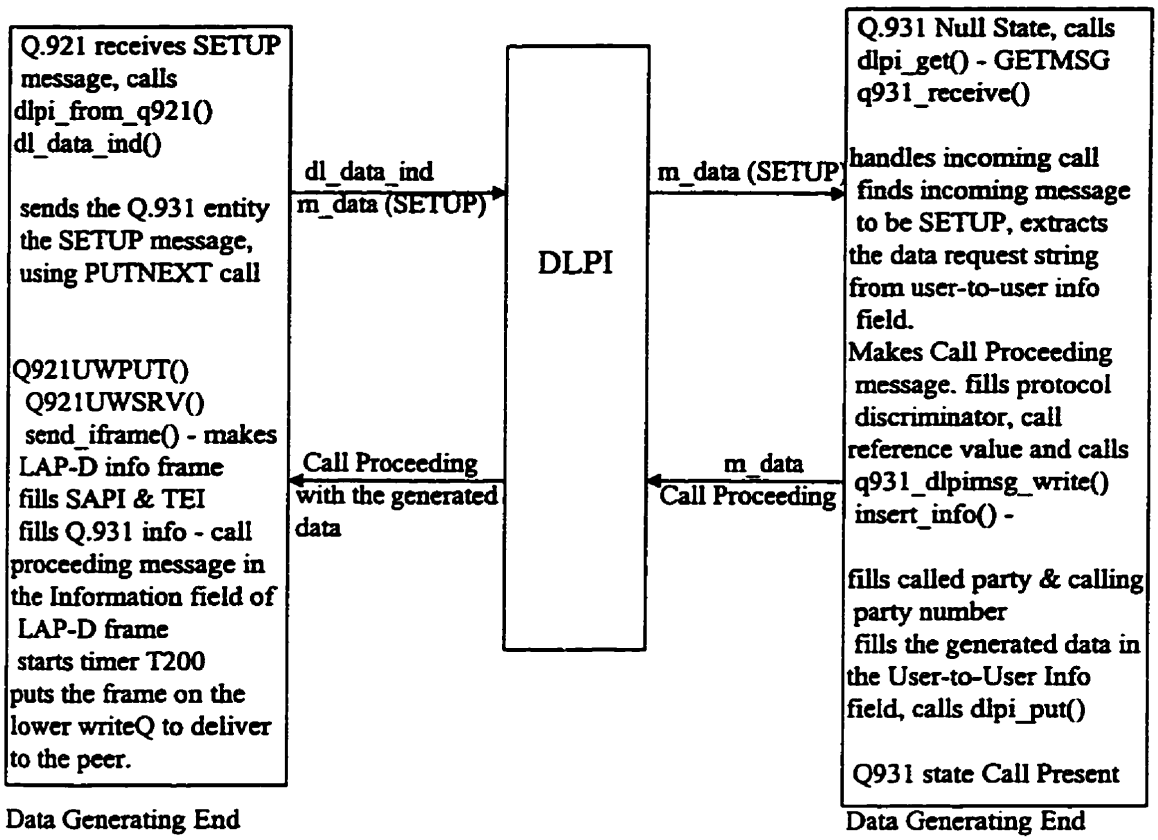


Fig 6.11 Steps 15, 16, 17, 18 & 19

The q931_dlpimsg_write function calls the message_decode function that converts the entire call record into a stream of bytes. Finally the q931_dlpimsg_write function calls the dlpi_put function that stores the stream of bytes into the data buffer of the M_DATA message type of DLPI and uses the PUTMSG system call to deliver it to the DLPI of Q.921 module.

19) The Q.921 pseudo device driver receives the M_DATA message in its Q921UWPUT procedure and puts it back on the queue to be serviced by Q921UWSRV procedure. This procedure calls the send_iframe function. The send_iframe function prepares a LAP-D information frame. It fills the SAPI and TEI values in the address field of the frame, sets the command bit and inserts the Q.931 databuf information in the

information field of the frame. It starts the T200 (1 SEC to wait for an ack) timer and sends the LAP-D frame on the D-Channel queue, which is the lower write queue of the device driver.

20) The peer Q.921 pseudo device driver receives the LAP-D information frame from the lower read queue, through the Q921LRPUT procedure. This procedure calls the dlphi_to_q921 function, as the message is coming from the lower read queue. The function checks the SAPI value in the message and finds it to be SAPI_Q931_8 (call processing message). It checks the frame type of the message and on finding it to be an information frame, calls the recv_iframe function. The recv_iframe function matches the sequence number of the Information frame received, updates the counters that keep track of the sequence numbers and calls the send_sframe function to send back the receive ready acknowledgment to the peer LAP-D entity. The send_sframe function fills the SAPI and the TEI values in the address field of the message and RR (Receive Ready) in the command field. It puts the message on the lower write queue (D-Channel queue) to deliver it to the peer LAP-D entity, that had sent the information frame. The recv_iframe also calls dlpi_from_q921 function to send the Q.931 part of the message to the Q.931 entity.

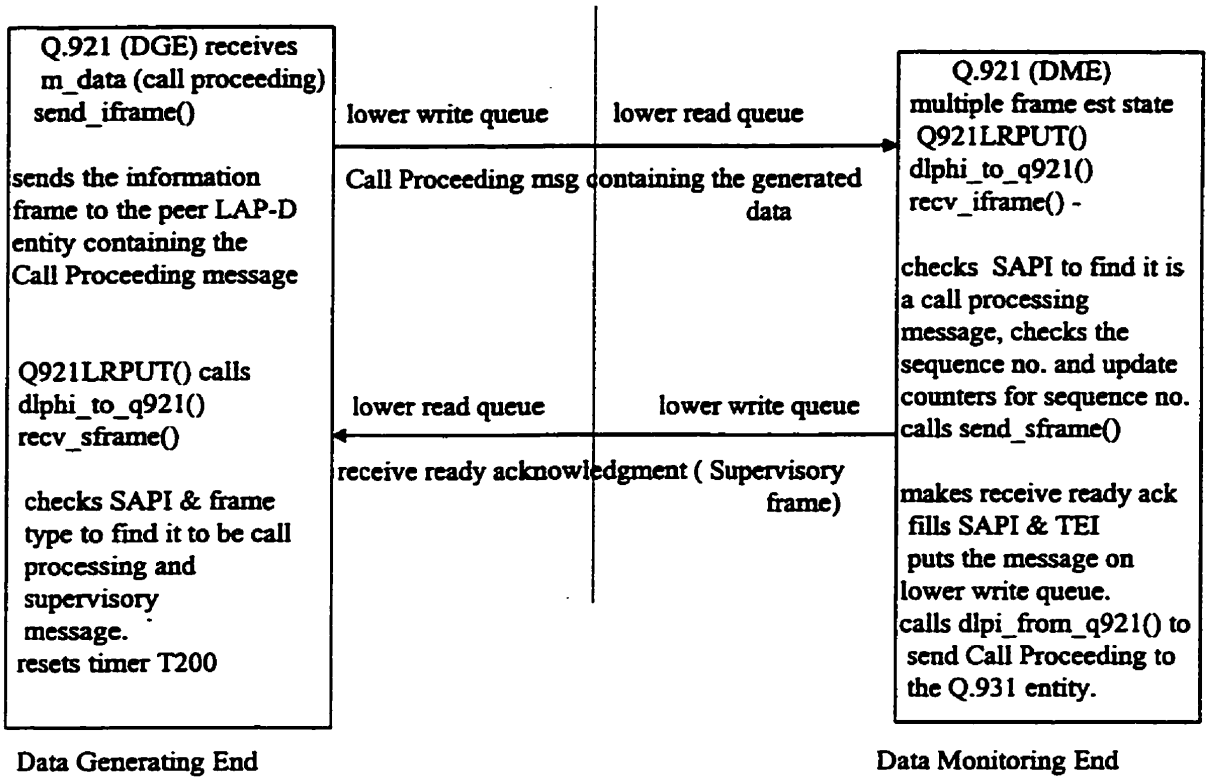


Figure 6.12 Step 20

21) The dlpi_from_q921 function calls dl_data_ind function of the DLPI which sends the Q.931 part of the LAP-D frame received, up to the Q.931 module in M_DATA type message, using the PUTNEXT system call. The primitive used is DL_DATA_IND.

22) The DLPI module implemented in the Q.931 entity calls the dlpi_get function to receive the message coming from the DLPI of the Q.921 module. The dlpi_get function uses the GETMSG system call to extract the message from the queue. The function on finding the DLPI primitive received to be DL_DATA_IND calls the message_decode function, that extracts the data from the data buffer attached to the message and stores the data in the data structures of the Q.931 module. It calls the q931_call function to handle the incoming message for the call initiated by itself.

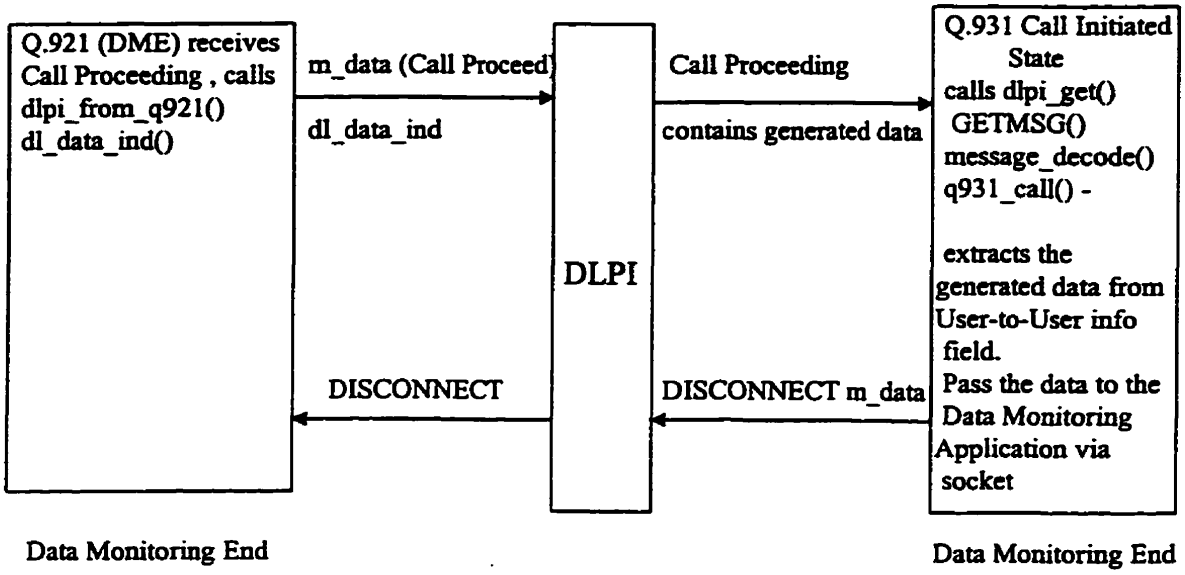


Figure 6.13 Steps 21, 22 & 23

23) The Q.931 entity (DME) is in NULL state. The q931_call function finds the message type of the incoming message to be CALL PROCEEDING. It extracts the remotely generated data from the User-to-User information field of the message and stores it in to the data structure from where the Q.931 entity can send that data to the Data Monitoring Application via the socket connection. The q931_call function now initiates the call disconnection by preparing the DISCONNECT message and sending it to the Q.921 entity using the same procedure as was used to send the SETUP message. The peer Q.931 entity on receiving the DISCONNECT message, will send the RELEASE message to the peer. The peer will again respond with RELEASE COMPLETE to end the connection.

24) The Q.931 server sends the received data to the Data Monitoring application via the socket connection. It uses the WRITE system call for that.

As described in the above steps, we are able to successfully test the TeleMonitoring application. The above steps demonstrate how the protocols (LAP-D & Call Control - Q.931) and the D-Channel can be used to support real-time applications at homes and small businesses. The testing framework is a generalized framework and can easily be used to test the TeleControl and the TelePolling application. The TeleControl application is a sub application of the TeleMonitoring application and the control information can be transferred in the User-to-User information field of the SETUP message. Once the control information is transferred, the call clearing can be initiated as described in the above steps. For the TelePolling application, the same testing framework can be used with different sequence of steps. The Call Control application is used to establish a temporary signaling connection on the D-Channel and once the connection is established, the two Q.931 entities can exchange User Information messages. The applications will insert and extract the image in these messages.

Chapter 7

Conclusion and Future Works

This thesis demonstrates how ISDN can be used to support real-time applications at homes and small businesses. We have explored the ISDN protocols in detail and have shown the protocol features well suited for real-time applications at homes and small businesses. The fast call setup time over the D-Channel makes the D-Channel preferable for real-time applications [Holl97]. The idea of using the D-Channel for real-time applications, leaves the B-Channel free for the circuit switched connections for other services such as voice calls. The thesis focuses on three real-time applications namely TeleMonitoring, TeleControl and TelePolling. The high level scenarios for these applications, the ISDN protocol message and bit level scenarios and the implementation framework for these applications gives us an insight as to how other real-time applications can be supported by the ISDN. These applications strengthen the increasing penetration of narrowband ISDN and the telecommunication industry into the residential markets. The applications also allow the ISDN switch companies to generate revenue from other businesses such as utility meter reading companies. The testing framework developed for these applications provides an ideal platform for the testing of other real-time applications. Therefore, based on the ideas put forth in this thesis, it is hoped that the future work would explore more real-time applications for homes and small businesses.

One of the future projects that has a great potential in residential markets and small businesses is the integration of ISDN with the IEEE1394 serial bus as described in [Wick97] and [Hoff97]. IEEE1394 is a high speed serial bus that provides a high speed method of interconnecting digital devices, consumer electronics products and computers, and transports digital data for these devices. IEEE1394 enables high performance multimedia connections and control of business and consumer electronic devices such as televisions, stereos and CD changers, as well as traditional PC devices such as hard drives, CD-ROM drives, printers and scanners. The bus transports data at rates up to 400 Mb/s and its protocol supports guaranteed delivery of time sensitive information, so that digital data, video and audio can be transmitted in real-time. Most of the companies have started providing the support for IEEE1394 interfaces in their products. Microsoft Corp. is providing the IEEE1394 serial bus interface standard in Microsoft Windows operating systems.

As a future work for this thesis , in order to come out with more real-time applications at homes and small businesses, we propose an ISDN and IEEE1394 bridge. ISDN will provide remote access to the IEEE1394 serial bus connected to several home appliances and remote devices. It is expensive to connect ISDN to every single device at home and offices. IEEE1394 is inexpensive and has a packet structure similar to the ISDN frame, in order to allow bridging with ISDN. Thus the first 10 feet and the last 10 feet of the information super highway can be IEEE1394 and the remaining can be the ISDN network.

BIBLIOGRAPHY

[BellISDN96] Bellsouth Telecommunications., "Integrated Services Digital Networks", http://www.bst.bls.com/products-services/isdn-main.html, 1996

[BellISDNapp96] Bellsouth Small Business District., "Applications – ISDN Means Useful", http://www.smallbiz.bellsouth.com/bssbi2.html, 1996

[FTellISDN97] FranceTelecom North America, Products and Solutions – Voice and Data Services, "ISDN: A User's Guide" http://www.francetelecom.com/ps/ps_voda/isdn/isdnindex.html, 1997

[Hoff95] Hoffman G., and Moore D., "IEEE 1394: A Ubiquitous Bus" COMPCON Conf. in San Francisco, 1995

[ITUQ931] ITU-T., "Digital Subscriber Signaling System No. 1 (DSS 1) – ISDN User-Network Interface Layer 3 Specification for Basic Call Control (Q.931)", International Telecommunication Union, 1993

[ITUQ920] ITU-T., "Digital Subscriber Signaling System No. 1 (DSS 1) – ISDN User-Network Interface Data Link Layer – General Aspects (Q.920)", International Telecommunication Union, 1993

[ITUQ921] ITU-T., "Digital Subscriber Signaling System No. 1 (DSS 1) – ISDN User-Network Interface Layer – Data Link Layer Specification (Q.921)", International Telecommunication Union, 1993

[ITUQ957] ITU-T., "Digital Subscriber Signaling System No. 1 (DSS 1) - Stage 3 description for additional information transfer supplementary services using DSS 1: User-to-User Signalling (UUS)", International Telecommunication Union, 1996

[Holl97] Holliday C. R., "The Residential Gateway", IEEE Spectrum, May 1997

[Kess93] Kessler, C. Gary., "ISDN Concepts, Facilities, and Services", McGraw-Hill, 1993

[Nitz90] Nitzberg K. G. "ISDN Applications and Society" http://www.nitzspace.com/gary/papers/tmisdn.html, 1990

[Stall95] Stalling W., "ISDN and Broadband ISDN with Frame Relay and ATM", Prentice-Hall, 1995

[Stall94] Stalling W., "Data and Computer Communication", Prentice-Hall, 1994

[Stev96] Stevens W. Richard., "Unix Network Programming", Prentice-Hall, 1996

[Suns96] Bengt Sahlin, "SunShine., Implementation of the ISDN Recommendations – Q.921 & Q.931",http://www.tcm.hut.fi/~bos/ISDN/sunshine/SunShine.html, 1996

[Wick97] Wickelgren., "The Facts About Firewire", IEEE Spectrum, April 1997

Appendix – A

The Unix System Calls For Q.931 Server and the Applications

The various system calls that the Q.931 server and the Applications use to communicate with each other are as follows:

a) Socket System Call

int socket(int family, int type, int protocol) ;

The call returns a descriptor to the newly created socket. Arguments to the call specify the protocol family that the application will use (eg., PF_INET assigned for TCP/IP or Internet Protocols). The type is set to SOCK_STREAM for stream socket and SOCK_DGRAM for datagram socket. For a socket that uses the Internet protocol family, the protocol or type of service argument determines whether the socket will use TCP or UDP. For example if the protocol family indicated is PF_INET (Internet Protocol) and the type is set to SOCK_STREAM, it means TCP/IP and stream sockets are used by the application.

b) Bind System Call

int bind(int sockfd, struct sockaddr *myaddr, int addrlen) ;

The bind system call assigns a name to an unnamed socket. The call takes arguments that specify a socket descriptor (*sockfd*) and an endpoint address (*sockaddr*), and the size of this address structure (*addrlen*). For TCP/IP protocols, the endpoint address uses the *sockaddr_in* structure.

```
Struct sock_addr {
 short sa_family ;   /* socket address family */
 union {
```

}

sockaddr_in structure uses the address family as internet protocols (PF_INET), next two bytes are the port address and the next four bytes are the host ID. **c) Listen System Call**

int listen(int sockfd, int backlog) ;

When a socket is created, the socket is neither active, nor passive until the application takes further action. Connection oriented servers call *listen* system call to place a socket in passive mode and make it ready to accept incoming connections. The *sockfd* argument is the same as in the previous system calls. The *backlog* argument specifies how many connection requests can be queued up by the system while it waits for the server to execute the accept system call.

d) Acccept System Call

int accept(int sockfd, struct sockaddr *peer, int *addrlen) ;

After a server calls *socket* system call to create a socket, and *bind* system call to specify a local endpoint address, and *listen* system call to place it in passive mode, the server calls *accept* system call to extract the next incoming connection request from the queue. If there are no connection requests pending, this call blocks the caller until one arrives. *Accept* system call creates a new socket with the same properties as *sockfd*, for each new connection request, and returns the descriptor of the new socket to the caller. The peer and *addrlen* arguments are used to return the address of the connected peer process(the client). *Addrlen* is called a value-result argument: the caller sets its value before the system call, and the system call stores a result in the variable.

e) Connect System Call

int connect(int sockfd, struct sockaddr *servaddr, int addrlen) ;

After creating a socket, a client calls *connect* system call to establish an active connection to a remote server. The second argument to *connect* system call allows the client to specify the remote endpoint, which includes the remote machine's IP address and the port number at which the socket connection is opened. Once a connection has been made, a client can transfer data across it.

f) Read System Call

int read(int sockfd, char *buff, unsigned int nbytes) ;

Both clients and servers use *read* system call to receive data from a TCP connection. Usually, after a connection has been established, the server uses a *read* system call to receive a request that the client sends by calling *write* system call. After sending its request, the client uses *read* system call to receive a reply. To read from a connection, an application calls *read* system call with three arguments. The parameter *sockfd* specifies the socket descriptor to use, the parameter *buff* is the buffer where the data is to be stored. The parameter *nbytes* is the length of the buffer. The *read* system call extracts data bytes that have arrived at that socket, and copies them to the user's buffer area. If more data has arrived, *read* system call only extracts enough to fill the buffer. If less data has arrived, *read* system call extracts all the data and returns the number of data bytes obtained.

g) Write System Call

int write(int sockfd, char *buff, unsigned int nbytes) ;

Both clients and servers use *write* system call to send data across a TCP connection. Clients usually use *write* system call to send requests, while servers use it to send replies. A call to *write* system call requires three arguments. The application passes the

descriptor of a socket to which the data should be sent, the address of the data to be sent and the length of the data. Usually *write* system call copies outgoing data into buffers in the operating system kernel, and allows the application to continue execution while the *write* system call transmits the data across the network.

h) Close System Call

int close(int sockfd)

The *close* system call is used to deallocate a socket once a client or server finishes using it. If only one process is using the socket, *close* system call immediately terminates the connection and deallocates the socket. If several processes share a socket, close system call decrements a reference count (the count keeps track of the number of processes that are actively using the socket connection at present) and deallocates the socket when the reference count reaches zero.

Appendix – B

The Source Code for the Device Drivers

Source code of the device driver that links the lower queues of the Q.921 Multiplexing Device Drivers

```
/*
 *  insertmux.c
 *  Opens the lower queues of  both the Q.921 Multiplexing Device Drivers
 *  Links the lower queues of the Q.921 Multiplexing Device Drivers
 *  i.e Links the read queue of one device driver with the write queue
 *  of the other
 *  Finally Unlinks the lower queues of the two device drivers and closes
 *  them
 */

#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

#include <errno.h>


static char *progname;

/*
  Main gets as argument open or close and calls the appropriate
  function to either open the device drivers or to close the device
  drivers..
*/

void main(int argc, char *argv[])
{
 progname = argv[0];

 if (argc != 2) {
  fprintf(stderr, "Usage: %s open|close\n", argv[0]);
  exit(1);
 }

 if (strcmp (argv[1], "open") == 0)
  muxopen();      /* opens the lower queues of both the device drivers */
 if (strcmp (argv[1], "close") == 0)
  muxclose ();    /* closes the lower queues of the device drivers */
```

```
/*
  muxopen opens the D-Channel queues ( lower queues ) and makes
  links to these in the kernel. The file descriptors for the channels are
  saved in a file - muxvals - for use when closing the  links. The
  persistent links are opened after issuing an I_LINK-ioctl.
*/

void muxopen(void)
{

 int cardfd1, cardd_mgt_fd1, cardmuxdmuxmgt1 ;
 FILE *fpp, *fpp1;

 fpp = fopen("muxvals", "w");

 if (fpp == NULL)
  {
   fprintf(stderr, "%s: Cannot open muxvals for writing\n", progname);
   exit(1);
  }


 fpp1 = fopen("drivervals", "w");

 if (fpp1 == NULL)
  {
   fprintf(stderr, "%s: Cannot open muxvals for writing\n", progname);
   exit(1);
  }

 cardfd1 = cardfd2 = 0;


/* gets the file descriptor to the lower queues of one of  the
   device drivers */

 if ((cardfd1 = open ("/dev/q921", O_RDWR)) < 0)
  {
   perror("Cannot open /dev/q921");
   punlinks(cardfd1, cardmuxdmuxmgt1 );
   closes(cardfd1, cardd_mgt_fd1);
   exit(1);
  }

 printf("/dev/q921 opened, cardd_mgt_fd1 %d\n", cardd_mgt_fd1);
```

```
/* gets the file descriptors to the lower queues of the peer
    device driver */

 if ((cardfd2 = open ("/dev/q922", O_RDWR)) < 0)
   {
    perror("Cannot open /dev/q922 first time\n");
    punlinks(cardfd1, cardmuxdmuxmgt1 );
    closes(cardfd1, cardd_mgt_fd1);
    exit(1);
   }

 printf("/dev/q922 opened, cardfd1 %d\n", cardfd1);

/* Links the lower queues  of both the device drivers */

if((cardmuxdmuxmgt1 = ioctl(cardfd2, I_LINK, cardfd2 )) < 0)
   {
    perror("cardmuxdmuxmgt1 I_PLINK fails");
    printf("error value is : %d ", cardmuxdmuxmgt1 );
    punlinks(cardfd1, cardmuxdmuxmgt1 );
    closes(cardfd1, cardfd2);
    exit(1);
   }

  printf("cardmuxdmuxmgt1 = %d plink\n", cardmuxdmuxmgt1);
 fprintf(fpp, "%d \n", cardmuxdmuxmgt1);
 fprintf(fpp1, "%d \n", cardfd2);
 fclose (fpp);
 fclose (fpp1);
 closes(cardfd1, cardfd2);
 return;
}


/*
  muxclose closes the  links between the lower queues
  of the two device drivers.
*/

void muxclose(void)
{
 int cardfd1;
 int cardmuxdmuxmgt1;
 int cardfd2;

 FILE *fpp;

 cardfd1 = cardfd2  = 0;
 cardmuxdmuxmgt1 = 0;
```

```
 fpp = fopen("muxvals", "r");
 if (fpp == NULL)
   {
    fprintf(stderr, "%s: Cannot open muxvals for reading\n", progname);
    exit(1);
   }
/* reads the descriptor that links the two lower queues from the file
    in which it was saved */

 fscanf(fpp, "%d", &cardmuxdmuxmgt1);
 fclose (fpp);

 punlinks(cardfd1 , cardmuxdmuxmgt1 ); /* unlinks the two lower queues */

 closes(cardfd1 , cardfd2 );    /* closes the file descriptors to the lower queues
                                   of the two device drivers  */

 return;
}


/*
 *
 * punlinks, unlinks the  links that have been made.
 *  between the lower queues of  the two device drivers
 *  This is done in error situations and when exiting.
 *
 */

void  punlinks(int cardfd, int cardmuxdmuxmgt)
{
 int retval = 0;

 if (cardmuxdmuxmgt != 0)
   {
    /* uses the I_UNLINK IOCTL */

    if ((retval = ioctl(cardfd, I_UNLINK, cardmuxdmuxmgt)) < 0)
         perror("cardmuxdmuxmgt punlink failed");
   }

 return;
}
```

```
/*
 *
 * The closes function closes file descriptors that are passed
 * to it as  parameters. The function is called in error situations
 * and when exiting.
 *
 */

void
closes(int cardfd1,  int cardfd2 )
{
 int retval = 0;

 if (cardfd1 != 0)
   {

/* closes the file descriptor of first device driver */

    if((retval = close(cardfd1 )) < 0)
          perror("closing cardfd1 failes");
   }

 if (cardfd2 != 0)
   {

/* closes the file descriptor of the second device driver */

    if((retval = close(cardfd2 )) < 0)
          perror("closing cardfd2 failes");
   }

 return;
}


/* The close_fd function  reads the open file descriptors from the
* file in which they were saved, when the device drivers were
* opened, and close those file descriptors in order to close the
* file descriptors.
*/


void close_fd (void)
{
 int cardfd1 , cardfd2 ;
 int retval = 0;
 FILE *fpp;

 cardfd1  = cardfd2 = 0;


 fpp = fopen("drivervals", "r");
```

```
 if (fpp == NULL)
   {
    fprintf(stderr, "%s: Cannot open muxvals for reading\n", progname);
    exit(1);
   }

 fscanf(fpp, "%d  %d ", &cardfd1, &cardfd2 );

/* reads the file descriptor of the first device driver */

 fclose (fpp);

 if (cardfd1 != 0)
   {
    if((retval = close(cardfd1)) < 0)
          perror("closing cardfd1 failes");
   }


 if (cardfd2 != 0)
   {
    if((retval = close(cardfd2)) < 0)
          perror("closing cardfd2 failes");
   }

 return;
}
```

Appendix – C

Source Code for the Applications

Source code for the Application that connects to the Q.931 Socket Server and sends the request for the call establishment with the remote end along with the request for remotely generated data.

```
/* This file contains the code for the Applications that
 * opens the socket connection to the Q931 server and
* sends to its Q.931 entity, the  phone number and a
 * connection establishment request with the remote
*  Q.931 entity .
 */

#include "q932.h"
#include "isdn_client.h"
#include <stropts.h>
#include <poll.h>

#define RBUF_LEN 1024
#define BUFF 1024

int callopt = FALSE;
int verbose = FALSE;


void main (int argc, char *argv[])
{
 unsigned char
  recv_string[RBUF_LEN];

 char
  *phonenumber,
  *request_msg,
  *host_string;

 int
  port_selected = FALSE,
  first_time = TRUE,
  b_sock = 0,
  s_sock = -1,
  aflg = FALSE,
  pflg = FALSE,
  errflg = FALSE,
  c;

 u_short
  connect_port,
```

```
extern char
 *optarg;



/* Default values */
phonenumber = "80410000";
connect_port = STANDARD_PORT;
host_string = "dragon.acadiau.ca";

while((c = getopt(argc, argv, "P:h:p:va")) != -1)
 {
  switch(c)
        {
        case 'p' :
         if(aflg)
          errflg++;
         else
          { /* phone number to be entered by the user */

           pflg++;
           phonenumber = optarg;
           printf("Phone = %s\n", phonenumber);
           break;
          }
        case 'a' :
         if(pflg)
          errflg++;
         else
          {
           aflg++;
           break;
          }
        case 'h' :
         host_string = optarg;
         break;

        case 'P' :  /* enter the port number of the Q.931 server */

         connect_port = atoi(argv[optind - 1]);
         if (connect_port <=0 )
          errflg++;
         port_selected = TRUE;
         break;
        case 'v' :
         /* take all the default arguments */

         verbose = TRUE;
         break;
        case '?' :

         fprintf(stderr, "Option %c not recognized\n", optopt);
```

```
            errflg++;
            break;

          case ':' :
            fprintf(stderr, "Option %c requires an argument\n", optopt);
            errflg++;
            break;
          default:
            errflg++;
            break;
          }
  }
if(errflg)
  errmsg();


if (verbose)
  {
   printf ("Verbose mode\n");
   printf ("Host = %s\n", host_string);
   printf ("Phonenumber = %s\n", phonenumber);
   printf ("Connect_Port = %i\n", connect_port);

  }

s_sock = server_function(host_string, connect_port);

if (aflg)
  {
   msg = "C_ACTIVE";
   printf("Client: Set server to wait for incoming call = %s\n", msg);
  }
else
  {
   /* number validation */
   if (check_number(phonenumber) < 0)
          {
           printf("Unknown number: Discarding\n");
           exit(0);
          }
   if (verbose)
          printf("Client: Calling phonenumber: %s \n", phonenumber);
   msg = phonenumber;
  }

if (send(s_sock, msg, strlen(msg), 0) == -1)
  {
   perror("Client: cannot write to server");
   exit(1);
  }
```

```
 /*
   Read the data from server until DISCONNECT is  received
   */

 while (TRUE)
  {
   if (verbose)
        printf("Client: reading from server..\n");

   if (recv(s_sock, recv_string, RBUF_LEN, 0) == -1)
        {
         printf("SOCK = %i\n", s_sock);
         perror("RECV fails");
         exit(1);
        }

   if (strcmp(recv_string,"VCONNECT") == 0)
        {
         if (verbose)
           printf("Client: %s recieved - creating new socket connection for B-channel data\n",
recv_string);

         b_sock = server_function(host_string, b_channel_port);
         get_b_channel_data(b_sock);
        }

   else if (strcmp(recv_string,"DISCONNECT") == 0)
        {
         printf("Client: %s recieved - closing connection\n", recv_string);
         /*
           shutdown(s_sock, 2);
           close(s_sock);
           exit(0);
           */
        }
   else
        {
         printf("Hey You !!! %s called at: %s", recv_string);
         get_time();
        }
   memset ((void *)&recv_string, 0, sizeof(recv_string));
  }
}

/*
 * Check the validity of the  phonennumber  number
 */
int check_number(char *nr)
{
 char
  *foo;

 while ( *(nr) != NULL)
```

```
   {
    foo  = *(nr++);
    if (!isdigit(foo))
          {
           printf("ERROR: understanding number %u\n", foo);
           return (-1);
          }
   }
 return (1);
}


/* The function server_function , takes the Q.931
    server address and the port number on which the
    server is listening. It calculates the TCP/IP address
   of the server and creates a socket for the Application.
   Then it tries to connect to the port on which the server
   is listening.
*/


int server_function(char *host_string, u_short port)
{
 struct sockaddr_in server_addr;
 struct hostent *serv_host;

 int
  s_sock,
  s_addr_len = sizeof(server_addr);

 u_long
  host_addr;

 memset((void*)&server_addr, 0, s_addr_len);

 host_addr = inet_addr(host_string); /*  gets the TCP/IP address of Q.931 server*/

 /* GET HOST ENTRY */
 if ((long)host_addr != (long)(-1))
  memcpy ((void *)&server_addr.sin_addr,
            (void *)&host_addr,
            sizeof(host_addr));
 else
  {
   if ((serv_host = gethostbyname(host_string)) == NULL)
         {
          perror("Client error: Cannot get host entry");
          exit(1);
         }

   memcpy(&server_addr.sin_addr,
            serv_host->h_addr,
            serv_host->h_length);
  }
```

```
 server_addr.sin_port = htons(port);
 server_addr.sin_family = htonl(AF_INET);



 /* CREATE SOCKET */
 s_sock = socket(AF_INET, SOCK_STREAM, 0);
 if (s_sock == -1)
  {
   perror("Client: Creating socket fails");
   exit(1);
  }

 /* CONNECT */
 if ((connect(s_sock,
              (struct sockaddr *)&server_addr,
              s_addr_len)) == -1)
  {
   printf("HOST = %s\n", host_string);
   perror("Client error: Connection fails ");
   exit(1);
  }
 return s_sock;
}

/*
 * Gets the  time of incoming call
 * Returns string with date & time "120496 at 14:43"
 */
int get_time(void)
{
 struct timeval tp;
 struct timezone tzp;
 struct tm *time;

 long *clock;

 if (gettimeofday(&tp, &tzp) == -1)
  {
   perror("Error asking for time");
   return (-1);
  }

 clock = &tp.tv_sec;
 time = ctime(clock);
 printf("%s", time);

 return (1);
}
```

```
/*
  Standard Error messages
  */
void errmsg(void)
{
 printf("Usage of isdn_test_client: -h host [-v] -p phonenumber | -a [-P port]\n");
 exit(1);
}
```

Appendix - D

Source code for the Q.931 Server

Source code for the Q.931 Server part that connects to the application and the Q.921 module. It initiates the call establishment with the peer Q.931 entity.

```
/* This file contains the main function for the Q.931 entity.
   The main function starts the socket server. It opens the
   Q.922 device driver i.e. gets the file descriptor to the
   upper queues of the Q.922 device driver. It receives the
   connection establishment request from the Applications
   with the phone number of the remote Q.931 entity and the
   request for data from the remote entity. The function calls
   the protocols functions to initiate the connection establishment
   with the peer Q.931 entity and the remote Application */


#include "q932.h"
#include "my_socket.h"

#define BUFFSIZE 1024

int dlpi_data_req();
void notify_client();
int T308_counter = 0;

struct pollfd pollfds[NPOLL];
struct sockaddr_in client_addr;


int main(int argc, char *argv[])
{

 int
  retval,
  c_sock,
  c_addr_len = sizeof(client_addr),
  count,
  MFS_count = 0,
  i;
 u_short
  port = 0;
```

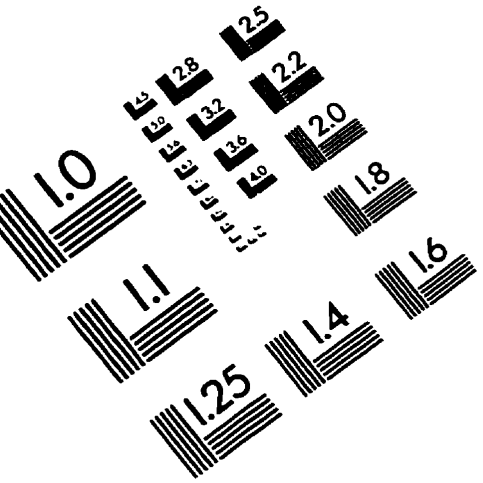
```
c_state->first_call = TRUE;
c_state->bc_sock = -2;
c_state->client_active = FALSE;
c_state->lb_sock = -3;
c_state->client_socket = -2;

qs.timerset = FALSE;

switch(argc)
 {
 case 1:
  break;
 case 2:
  port = atoi(argv[1]);
  break;
 default:
  printf("Usage: %s [PORT]\n", program);
  exit(1);
 }

/* opens q922 device driver and gets file descriptor with the
   pair of queues  */

if ((fd.dlpi_fd = dlpi_open(0)) < 1)
 {
  printf("q932: Unable to open dlpi interface !\n");
  undefined_function();
 }


if((retval = memalloc()) < 0)
 {
  perror("ERROR allocating memory for struct!");
  undefined_function();
 }

/* prepares the Call Establishment record, in case there is request for call
  establishment from the Application */

callinfo.len = 0;
callinfo.primitive = DL_ESTABLISH_REQ;
callinfo.q932_encode = NULL;
qs.polltimeout = ESTABLISH;
qs.q922_state = UFS;
qs.connect_nro = 0;
qs.timerset = FALSE;
qs.state = NULL_STATE;
qs.polltimeout = ESTABLISH;


      count = poll(pollfds, NPOLL,qs.polltimeout);

        callinfo.primitive = DL_ESTABLISH_CON ;
```

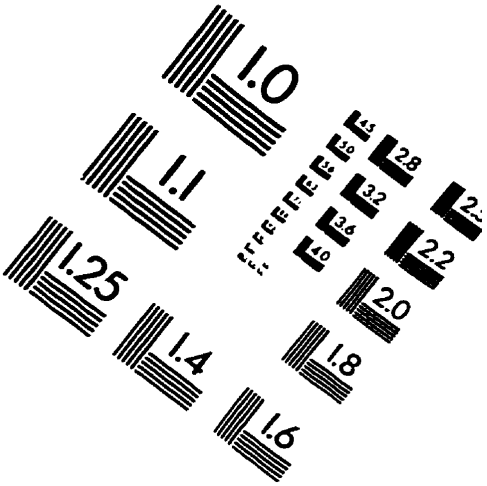
```
            if(callinfo.primitive == DL_ESTABLISH_CON)
              {
               qs.q922_state = MFS;

               /* Starts the socket server on the specified port */



               printf("Q922 now in state: %d\n", qs.q922_state);
               if((fd.socket_fd = start_socket_server(port)) < 0)
                     {
                      perror("Sockserver start");
                      undefined_function();
                     }
              }

    pollfds[1].fd = fd.socket_fd;
    pollfds[1].events = POLLIN;

/* polls the socket's file descriptor for incoming connection request from the
   application */


    if(pollfds[1].revents)
          {
           c_sock = accept(fd.socket_fd,
                           ((struct sockaddr *)&client_addr),
                           &c_addr_len);


 /* calls accept system call to receive any request from the application */


           printf("Client call accepted\n");
           pollfds[2].fd = c_sock;
           pollfds[2].events = POLLIN;
           c_state->client_active = TRUE;
           c_state->client_socket = c_sock;
           c_state->listen_socket = fd.socket_fd;
          }

    free(callinfo.q932_encode);

    pollfds[3].fd = fd.dlpi_fd;
    pollfds[3].events = POLLIN;

 while (TRUE)
  {
      /* in a loop polls on the file descriptors of the socket to
        accept input from the Application and also polls
       on the file descriptor of the Q.921 device driver upper
      queue to check if any input is coming from that side */
```

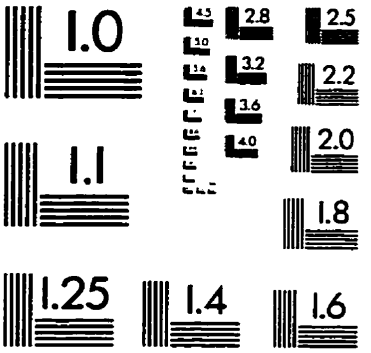
```
      poll(pollfds, NPOLL,qs.polltimeout);
    /* Check for incoming requests, case 0: ==> no data */
    for (i=2;i<NPOLL; i++) {
        printf("POLLING!!\n");
        switch (pollfds[i].revents) {

        default:
         perror("pollerror");
         undefined_function();
         break;

        case 0:
         break;

        case POLLIN:

         if(i == 2)
           {
            if((retval =  client_read(c_sock)) < 0)  /* reads the message from the Application *?

                 {
                  perror("ERROR reading client !");
                  undefined_function();
                 }
            break;
           }

         else if (i == 3)
           {
            printf("POLLFD = 3\n");

            free(callinfo.q932_encode);
            if((retval =  dlpi_get(&callinfo, fd.dlpi_fd)) < 0)  /* receives the message from the Q.921
                                                                    device driver */
                 {
                  perror("Fail to write q922 info into struct !");
                  undefined_function();
                 }

            if((retval = q932_dlpimsg_read(msg)) < 0)
                 {
                  perror("Error reading data from dlpi !");
                  undefined_function();
                 }
            break;
           }
        }
    }
   }
 return(1);
}

int memalloc()
{
```

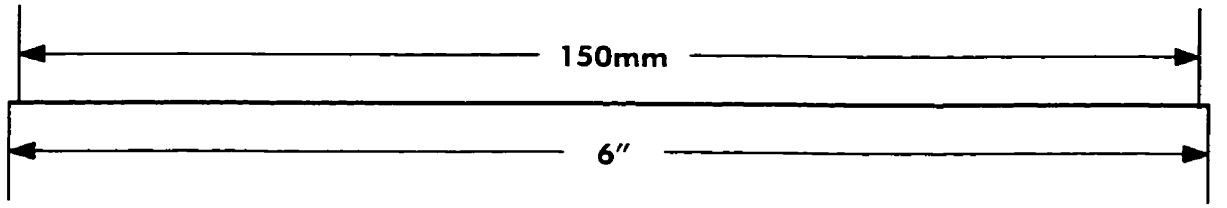
```
  msg = (q932_message_t *) malloc(sizeof(q932_message_t));
  if(msg == NULL)
    return(-1);
  else
    return(1);
}
```

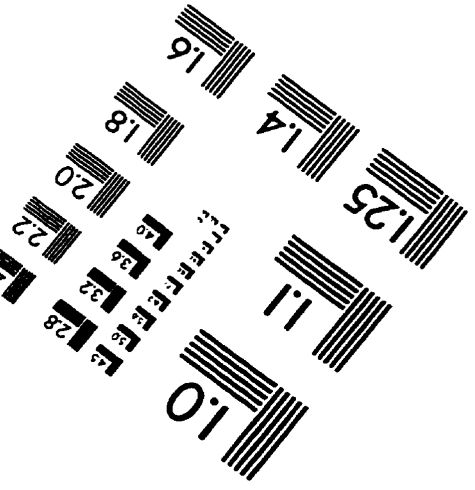
IMAGE EVALUATION
TEST TARGET (QA-3)

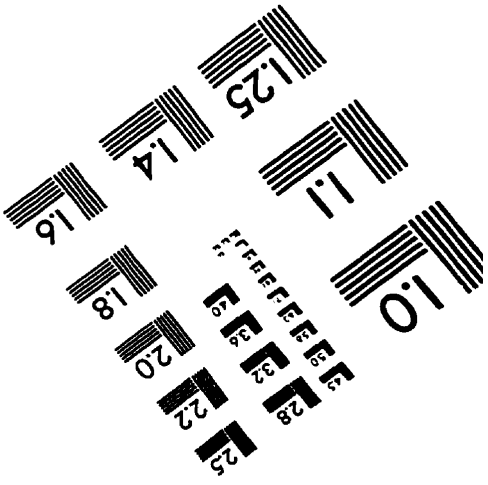












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989